

Jianer Chen  
Pinyan Lu (Eds.)

LNCS 10823

# Frontiers in Algorithmics

12th International Workshop, FAW 2018  
Guangzhou, China, May 8–10, 2018  
Proceedings

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, Lancaster, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Zurich, Switzerland*

John C. Mitchell

*Stanford University, Stanford, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

C. Pandu Rangan

*Indian Institute of Technology Madras, Chennai, India*

Bernhard Steffen

*TU Dortmund University, Dortmund, Germany*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbrücken, Germany*

More information about this series at <http://www.springer.com/series/7407>

Jianer Chen · Pinyan Lu (Eds.)

# Frontiers in Algorithmics

12th International Workshop, FAW 2018  
Guangzhou, China, May 8–10, 2018  
Proceedings



*Editors*  
Jianer Chen  
Texas A&M University  
College Station, TX  
USA

Pinyan Lu  
Shanghai University of Finance  
and Economics  
Shanghai  
China

ISSN 0302-9743                      ISSN 1611-3349 (electronic)  
Lecture Notes in Computer Science  
ISBN 978-3-319-78454-0              ISBN 978-3-319-78455-7 (eBook)  
<https://doi.org/10.1007/978-3-319-78455-7>

Library of Congress Control Number: 2018937383

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG  
part of Springer Nature  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

This volume contains the papers presented at FAW 2018: the 12th International Frontiers of Algorithmics Workshop, held during May 8–10, 2018, at Guangzhou University, in Guangzhou, P. R. China. The workshop provides a focused forum on current trends of research on algorithms, discrete structures, and their applications, and brings together international experts at the research frontiers in these areas to exchange ideas and to present significant new results.

The Program Committee, consisting of 33 top researchers from the field, reviewed 38 submissions and decided to accept 23 papers. Each paper had three reviews, with additional reviews solicited as needed. The review process was conducted entirely electronically via EasyChair. We are grateful to EasyChair for allowing us to handle the submissions and the review process and to the Program Committee for their insightful reviews and discussions, which made our job easier.

Besides the regular talks, the program also included one keynote talk by Andrew Chi-Chih Yao (Tsinghua University) and four invited talks by Yijia Chen (Fudan University), Ran Duan (Tsinghua University), Nick Gravin (ITCS, Shanghai University of Finance and Economics), and Mingji Xia (Institute of Software, Chinese Academy of Sciences).

We are very grateful to all the people who made this meeting possible: the authors for submitting their papers, the Program Committee members and external reviewers for their excellent work, and the five keynote and invited speakers. In particular, we would like to thank Guangzhou University for hosting the conference and providing organizational support and to the Institute for Theoretical Computer science (ITCS) at Shanghai University of Finance and Economics for hosting the conference website and providing some organizational support.

Finally, we would like to thank the members of the Editorial Board of *Lecture Notes in Computer Science* and the editors at Springer for their encouragement and cooperation throughout the preparation of this conference.

February 2018

Jianer Chen  
Pinyan Lu

# Organization

## Program Committee

Hee-Kap Ahn	POSTECH, South Korea
Georgios Barmpalias	Chinese Academy of Sciences, China
Xiaohui Bei	Nanyang Technological University, Singapore
Jianer Chen	Texas A&M University, USA
Wenbin Chen	Guang Zhou University, China
Yukun Cheng	Zhejiang University of Finance and Economics, China
Radu Curticapean	Simons Institute for the Theory of Computing, USA
Ran Duan	Tsinghua University, China
Fedor Fomin	University of Bergen, Norway
Mordecai J. Golin	The Hong Kong University of Science and Technology, SAR China
Heng Guo	The University of Edinburgh, UK
Xin Han	Dalian University of Technology, China
Iyad Kanj	DePaul University, Chicago, USA
Bundit Laekhanukit	Shanghai University of Finance and Economics, China/Max Planck Institute for Informatics, Germany
Lap-Kei Lee	Open University of Hong Kong, SAR China
Minming Li	City University of Hong Kong, SAR China
Bingkai Lin	The University of Tokyo, Japan
Tian Liu	Peking University, China
Pinyan Lu	Shanghai University of Finance and Economics, China
Kuldeep S. Meel	Rice University, USA
Neeldhara Misra	Indian Institute of Science, India
Kim Thang Nguyen	IBISC, University Paris-Saclay, France
Pan Peng	University of Vienna, Vienna, Austria
Venkatesh Raman	The Institute of Mathematical Sciences, India
Dominik Scheder	Shanghai Jiaotong University, Shanghai, China
Dimitrios Thilikos	National and Kapodistrian University of Athens, Greece
Changjun Wang	Beijing University of Technology, China
Ge Xia	Lafayette College, USA
Boting Yang	University of Regina, Canada
Chee Yap	New York University, USA
Deshi Ye	Zhejiang University, China
Chihao Zhang	Shanghai Jiao Tong University, China
Jialin Zhang	Institute of Computing Technology, Chinese Academy of Sciences, China

**Additional Reviewers**

Ahn, Taehoon  
Banik, Aritra  
Bläsius, Thomas  
Chau, Vincent  
Chen, Li-Hsuan  
Chen, Shiteng  
Erlebach, Thomas  
Kim, Mincheol  
Kirousis, Lefteris  
Lee, Seung Jun  
Lee, Seungjoon  
Majumdar, Diptapriyo  
McCartin, Catherine  
Mitsche, Dieter  
Poon, Sheung-Hung  
Raymond, Jean-Florent  
Schewior, Kevin

Srivathsan, B.  
Vishwanathan, Sundar  
Walczak, Bartosz  
Wu, Chenchen  
Xia, Mingji  
Xiao, Mingyu  
Xu, Chenyang  
Xu, Dachuan  
Xue, Yuan  
Yicheng, Pan  
Yu, Liang  
Yu, Wei  
Zhang, Chenhao  
Zhang, Peng  
Zhang, Zhijie  
Zhu, Binhai

# Contents

## Graph Algorithms

Two Kinds of Generalized 3-Connectivities of Alternating Group Networks . . . . .	3
<i>Jou-Ming Chang, Kung-Jui Pai, Jinn-Shyong Yang, and Ro-Yu Wu</i>	
On the Longest Spanning Tree with Neighborhoods . . . . .	15
<i>Ke Chen and Adrian Dumitrescu</i>	
Efficient Algorithms for a Graph Partitioning Problem. . . . .	29
<i>S. Vaishali, M. S. Atulya, and Nidhi Purohit</i>	
On the Minmax Regret Path Center Problem on Trees. . . . .	43
<i>Biing-Feng Wang, Jih-Hong Ye, and Chih-Yu Li</i>	
A Strongly Polynomial Time Algorithm for the Maximum Supply Rate Problem on Trees . . . . .	54
<i>Koki Takayama and Yusuke Kobayashi</i>	
The Maximum Distance- $d$ Independent Set Problem on Unit Disk Graphs . . .	68
<i>Sangram K. Jena, Ramesh K. Jallu, Gautam K. Das, and Subhas C. Nandy</i>	
New Approximation Algorithms for the Minimum Cycle Cover Problem. . . .	81
<i>Wei Yu, Zhaohui Liu, and Xiaoguang Bao</i>	

## Parameterized Algorithms

Parameterized Algorithms for Minimum Tree Cut/Paste Distance and Minimum Common Integer Partition . . . . .	99
<i>Jie You, Jianxin Wang, and Qilong Feng</i>	
Guarding Polyhedral Terrain by $k$ -Watchtowers . . . . .	112
<i>Nitesh Tripathi, Manjish Pal, Minati De, Gautam Das, and Subhas C. Nandy</i>	
Some (in)tractable Parameterizations of Coloring and List-Coloring. . . . .	126
<i>Pranav Arora, Aritra Banik, Vijay Kumar Paliwal, and Venkatesh Raman</i>	
Kernelization for $P_2$ -Packing: A Gerrymandering Approach . . . . .	140
<i>Wenjun Li, Junjie Ye, and Yixin Cao</i>	

Classical Complexity and Fixed-Parameter Tractability of Simultaneous Consecutive Ones Submatrix & Editing Problems . . . . .	154
<i>M. R. Rani, Mohith Jagalmohan, and R. Subashini</i>	
Improved Kernels for Several Problems on Planar Graphs . . . . .	169
<i>Qilong Feng, Beilin Zhuo, Guanlan Tan, Neng Huang, and Jianxin Wang</i>	
<b>Other Algorithms</b>	
On Bayesian Epistemology of Myerson Auction . . . . .	183
<i>Xiaotie Deng and Keyu Zhu</i>	
Low-Weight Superimposed Codes and Their Applications . . . . .	197
<i>Luisa Gargano, Adele A. Rescigno, and Ugo Vaccaro</i>	
Single Vehicle's Package Delivery Strategy with Online Traffic Congestion of Certain Delay Time . . . . .	212
<i>Songhua Li and Yinfeng Xu</i>	
The Complexity of Weak Consistency . . . . .	224
<i>Gaoang Liu and Xiuying Liu</i>	
Balanced Random Constraint Satisfaction: Phase Transition and Hardness . . .	238
<i>Tian Liu, Chaoyi Wang, and Wei Xu</i>	
Exact Algorithms for Allocation Problems . . . . .	251
<i>Sundar Annamalai and N. S. Narayanaswamy</i>	
Exact Algorithms for the Max-Min Dispersion Problem . . . . .	263
<i>Toshihiro Akagi, Tetsuya Araki, Takashi Horiyama, Shin-ichi Nakano, Yoshio Okamoto, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, Takeaki Uno, and Kunihiro Wasa</i>	
Non-orthogonal Homothetic Range Partial-Sum Query on Integer Grids [Extended Abstract]. . . . .	273
<i>Yuan Tang and Haibin Kan</i>	
A Method to Compute the Sparse Graphs for Traveling Salesman Problem Based on Frequency Quadrilaterals . . . . .	286
<i>Yong Wang and Jeffrey Remmel</i>	
Optimal Length Tree-Like Refutations of Linear Feasibility in UTVPI Constraints . . . . .	300
<i>P. Wojciechowski, K. Subramani, and Matthew Williamson</i>	
<b>Author Index</b> . . . . .	315

# **Graph Algorithms**



# Two Kinds of Generalized 3-Connectivities of Alternating Group Networks

Jou-Ming Chang<sup>1</sup>(✉), Kung-Jui Pai<sup>2</sup>, Jinn-Shyong Yang<sup>3</sup>, and Ro-Yu Wu<sup>4</sup>

<sup>1</sup> Institute of Information and Decision Sciences,  
National Taipei University of Business, Taipei, Taiwan  
spade@ntub.edu.tw

<sup>2</sup> Department of Industrial Engineering and Management,  
Ming Chi University of Technology, New Taipei City, Taiwan  
poter@mail.mcut.edu.tw

<sup>3</sup> Department of Information Management,  
National Taipei University of Business, Taipei, Taiwan  
shyong@ntub.edu.tw

<sup>4</sup> Department of Industrial Management,  
Lunghwa University of Science and Technology, Taoyuan, Taiwan  
eric@mail.lhu.edu.tw

**Abstract.** To strengthen the classical connectivity of graphs, two kinds of generalized  $k$ -connectivities of a graph  $G$ , denoted by  $\kappa'_k(G)$  and  $\kappa_k(G)$ , were introduced by Chartrand et al. [1] and Hager [7], respectively. The former is the so-called cut-version definition of connectivity, whereas the latter is the path-version definition of connectivity (a synonym was also called the tree-connectivity by Okamoto and Zhang [32]). Since the underlying topologies of interconnection networks are usually modeled as undirected simple graphs, as applications of these two kinds of generalized connectivities, one can be used to assess the vulnerability of the corresponding network, and the other can serve to measure the capability of connection for a set of  $k$  nodes in the network. So far the exact values of these two types of generalized connectivities are known only for small classes of graphs. In this paper, we study the two kinds of generalized 3-connectivities in the  $n$ -dimensional alternating group networks  $AN_n$ . Consequently, we determine the exact values:  $\kappa'_3(AN_n) = 2n - 3$  for  $n \geq 4$  and  $\kappa_3(AN_n) = n - 2$  for  $n \geq 3$ .

**Keywords:** Interconnection networks · Connectivity  
Generalized connectivity · Alternating group networks

## 1 Introduction

As usual, the underlying topologies of interconnection networks are modeled as undirected simple graphs, where vertices and edges in a graph represent processing elements and their communication channels, respectively. In this paper, we



study two kinds of generalized connectivities of alternating group networks. As applications of the two types of generalized connectivities of graphs, one can be used to assess the vulnerability of the corresponding network, and the other can serve to measure the capability of connection for a set of nodes in the network.

Let  $G$  be a graph with vertex set  $V(G)$  and edge set  $E(G)$ . For any two vertices  $x, y \in V(G)$ , a path joining  $x$  and  $y$  in  $G$  is called an  $(x, y)$ -path, where  $x$  is the *starting vertex* and  $y$  is the *terminal vertex*. We may extend this notion to say that, for two disjoint subsets  $X, Y \subset V(G)$ , an  $(X, Y)$ -path is a path starting at a vertex  $x \in X$ , ending at a vertex  $y \in Y$ , and whose internal vertices belonging to neither  $X$  nor  $Y$ . Also, we simply write  $(x, Y)$ -path instead of  $(X, Y)$ -path if  $X = \{x\}$ . Two  $(x, y)$ -paths (resp.,  $(X, Y)$ -paths) are *internally disjoint* if they have no vertex and edge in common except for the starting vertex and terminal vertex. In particular, a set of  $k$  internally disjoint  $(x, y)$ -paths is also called a  $k$ -path container between  $x$  and  $y$ , and a set of  $k$  internally disjoint  $(x, Y)$ -paths whose terminal vertices are distinct is referred to as a  $k$ -fan from  $x$  to  $Y$ . Note that the term ‘‘container’’ is used to the study of evaluating how much node failures can be tolerated in the network transmission [8].

The *connectivity* of a graph  $G$ , denoted by  $\kappa(G)$ , is the minimum number of vertices whose removal from  $G$  results in a disconnected or trivial graph. A graph  $G$  is  $k$ -connected if  $\kappa(G) \geq k$ . A well-known result by Whitney [35] (as a corollary of Menger’s Theorem [29]) provided an equivalent definition of connectivity as follows: A graph  $G$  is  $k$ -connected if and only if  $G$  admits a  $k$ -path container between any pair of vertices. In addition, the following characterization of  $k$ -connected graphs emerged from [6] is related to the concept of  $k$ -fan and is customarily called the Fan Lemma (e.g. see [34, Theorem 4.2.23]).

**Lemma 1** (see [6]). *A graph  $G$  is  $k$ -connected if and only if  $|V(G)| \geq k+1$  and, for any vertex  $x \in V(G)$  and  $Y \subseteq V(G) \setminus \{x\}$  with  $|Y| \geq k$ , there exists a  $k$ -fan in  $G$  from  $x$  to  $Y$ .*

For a set of vertices  $S \subseteq V(G)$  with  $|S| \geq 2$ , two trees  $T$  and  $T'$  in  $G$  that connect  $S$  are called *internally disjoint trees* (IDTs for short) if  $E(T) \cap E(T') = \emptyset$  and  $V(T) \cap V(T') = S$  (i.e., the two trees are vertex-disjoint in  $G \setminus S$ ). Hereafter, we use  $\psi(S)$  to denote the maximum number of pairwise IDTs that connect  $S$  in  $G$ . By extending from a path container to a tree container, Hager [7] generalized the path-version definition of connectivity as follows: For an integer  $k \geq 2$ , the *generalized  $k$ -connectivity* of a graph  $G$ , denoted by  $\kappa_k(G)$ , is defined as  $\kappa_k(G) = \min\{\psi(S) : S \subseteq V(G) \text{ and } |S| = k\}$ . Note that, for such a generalization of connectivity, a synonym was also called the tree-connectivity by Okamoto and Zhang [32].

Actually, almost at the same time, Chartrand et al. [1] proposed another generalization of the cut-version definition of connectivity as follows: For an integer  $k \geq 2$ , the *generalized  $k$ -connectivity* of a graph  $G$ , denoted by  $\kappa'_k(G)$ , is the minimum number of vertices of  $G$  whose removal produces a disconnected graph with at least  $k$  components or a graph with fewer than  $k$  vertices. Obviously, for any graph  $G$ , we have  $\kappa'_2(G) = \kappa_2(G) = \kappa(G)$ . However, for  $k \geq 3$ , these

two types of generalized  $k$ -connectivities are indeed different. Recently, Sun and Li [33] provided the sharp bounds of the difference between  $\kappa'_k(G)$  and  $\kappa_k(G)$ .

So far the study of  $\kappa'_k(G)$  has received less attention, for more details we refer to [5, 30, 31]. By contrast, there are many research results related to  $\kappa_k(G)$ . First of all, it has been proved in [20] that  $\kappa_3(G) \leq \kappa(G)$  for every connected graph  $G$ , while  $\kappa_k(G) \leq \kappa_{k-1}(G)$  is not true in general. Currently, the exact values of  $\kappa_k(G)$  are known only for small classes of graphs, such as complete graphs [2], complete bipartite graphs [15, 32], complete equipartition 3-partite graphs [16]. In fact, it has pointed out in [18, 20] that the investigation of  $\kappa_k(G)$  for general  $k$  is very difficult (see also [3] for complexity results), and thus succeeding research focused on the study of  $\kappa_3(G)$  (e.g., star graphs and bubble-sort graphs [21], product graphs [12, 14, 23], and others [13, 17, 19, 20]) and  $\kappa_4(G)$  (e.g., hypercubes [28]). For more further investigations of  $\kappa_k(G)$ , see also [24, 27], a survey [26], and a recently published book [25].

Ji [10] introduced the alternating group network (which is defined later in the next section) as an interconnection network topology for computing systems. In this paper, for the  $n$ -dimensional alternating group network  $AN_n$ , we determine the two types of generalized 3-connectivity of  $AN_n$  as follows.

**Theorem 1.** For  $n \geq 4$ ,  $\kappa'_3(AN_n) = 2n - 3$ .

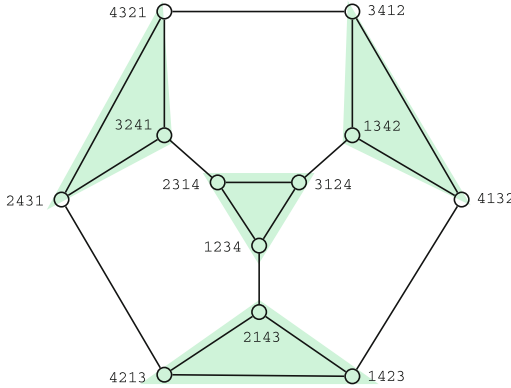
**Theorem 2.** For  $n \geq 3$ ,  $\kappa_3(AN_n) = n - 2$ .

## 2 Background of Alternating Group Networks

Let  $\mathbb{Z}_n = \{1, 2, \dots, n\}$  and  $A_n$  denote the set of all even permutations over  $\mathbb{Z}_n$ . For  $n \geq 3$ , the  $n$ -dimensional alternating group network, denoted by  $AN_n$ , is a graph with the vertex set of even permutations (i.e.,  $V(AN_n) = A_n$ ), and two vertices  $p = (p_1 p_2 \cdots p_n)$  and  $q = (q_1 q_2 \cdots q_n)$  are adjacent if and only if one of the following three conditions holds [10]:

- (i)  $p_1 = q_2, p_2 = q_3, p_3 = q_1$ , and  $p_j = q_j$  for  $j \in \mathbb{Z}_n \setminus \{1, 2, 3\}$ .
- (ii)  $p_1 = q_3, p_2 = q_1, p_3 = q_2$ , and  $p_j = q_j$  for  $j \in \mathbb{Z}_n \setminus \{1, 2, 3\}$ .
- (iii) There exists an  $i \in \{4, 5, \dots, n\}$  such that  $p_1 = q_2, p_2 = q_1, p_3 = q_i, p_i = q_3$ , and  $p_j = q_j$  for  $j \in \mathbb{Z}_n \setminus \{1, 2, 3, i\}$ .

The basic properties of  $AN_n$  are known as follows.  $AN_n$  contains  $n!/2$  vertices and  $n!(n-1)/4$  edges, which is a vertex-symmetric and  $(n-1)$ -regular graph with diameter  $\lceil 3n/2 \rceil - 3$  and connectivity  $n-1$ . For  $n \geq 3$  and  $i \in \mathbb{Z}_n$ , let  $AN_n^i$  be the subnetwork of  $AN_n$  induced by vertices with the rightmost symbol  $i$  in its permutation. It is clear that  $AN_n^i$  is isomorphic to  $AN_{n-1}$ . In fact,  $AN_n$  has a recursive structure, which can be constructed from  $n$  disjoint copies  $AN_n^i$  for  $i \in \mathbb{Z}_n$  such that, for any two subnetworks  $AN_n^i$  and  $AN_n^j$ ,  $i, j \in \mathbb{Z}_n$  and  $i \neq j$ , there exist  $(n-2)!/2$  edges between them. Figure 1 depicts  $AN_4$ , where each part of shadows indicates a subnetwork isomorphic to  $AN_3$ . For more properties on alternating group networks, we refer to [4, 9, 10, 36, 37].



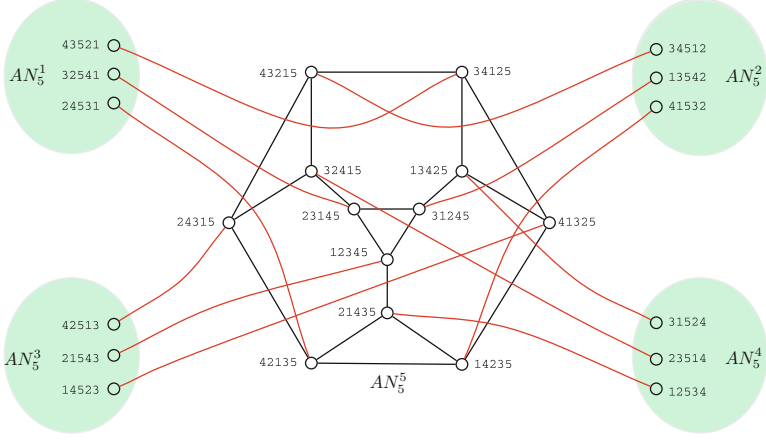
**Fig. 1.** Alternating group network  $AN_4$ .

For notational convenience, if a vertex  $x$  belongs to a subnetwork  $AN_n^i$ , we simply write  $x \in AN_n^i$  instead of  $x \in V(AN_n^i)$ . By  $H = \bigcup_{i=1}^k AN_n^i$ , it means that  $H$  is a subgraph of  $AN_n$  induced by the vertex set  $\bigcup_{i=1}^k V(AN_n^i)$ . Also, the subgraph obtained from  $AN_n$  by removing a set  $F$  of vertices is denoted by  $AN_n - F$ . In particular,  $F$  is called a *vertex-cut* if  $AN_n - F$  is disconnected. An edge  $(x, y) \in E(AN_n)$  with two end vertices  $x \in AN_n^i$  and  $y \in AN_n^j$  for  $i \neq j$  is called an *external edges* between  $AN_n^i$  and  $AN_n^j$ . In this case,  $x$  and  $y$  are called *out-neighbors* to each other. By contrast, edges joining vertices in the same subnetwork are called *internal edges*, and the two adjacent vertices are called *in-neighbors* to each other. Let  $E_n^{i,j} = \{(x, y) : x \in AN_n^i \text{ and } y \in AN_n^j\}$  be the set of external edges between  $AN_n^i$  and  $AN_n^j$  when  $i \neq j$ . The following are some basic properties of  $AN_n$ .

**Lemma 2** (see [9, 36, 37]). *For  $AN_n$  with  $n \geq 4$  and  $i, j \in \mathbb{Z}_n$  with  $i \neq j$ , the following holds:*

- (1)  $AN_n$  has no 4-cycle and 5-cycle.
- (2) Each vertex of  $AN_n$  has a set of  $n - 2$  in-neighbors and exactly one out-neighbor.
- (3) Any two distinct vertices of  $AN_n^i$  have different out-neighbors in  $AN_n - V(AN_n^i)$ .
- (4) There are exactly  $|E_n^{i,j}| = (n - 2)!/2$  edges between  $AN_n^i$  and  $AN_n^j$ .
- (5) Edges that cross between different subnetworks constitute a perfect matching in  $AN_n$ .

According to the property (2) in the above lemma, we use  $out(x)$  to denote the unique out-neighbor of a vertex  $x \in AN_n$ . Also, we denote  $N(x)$  the set of in-neighbors of  $x$ , and let  $N[x] = N(x) \cup \{x\}$ . Figure 2 shows the subnetwork  $AN_5^5$  and all out-neighbors of vertices in  $AN_5^5$ . From definition of  $AN_n$ , it is easy to derive the following properties.



**Fig. 2.** Alternating group subnetwork  $AN_5^5$  and all out-neighbors associated with it.

**Lemma 3.** For  $n \geq 4$ , let  $x \in AN_n$  be any vertex. If  $y, z \in N[x]$ , then  $out(y)$  and  $out(z)$  are contained in different subnetworks.

**Lemma 4.** For  $n \geq 4$  and  $i \in \mathbb{Z}_n$ , if  $x \in AN_n^i$  and  $j \in \mathbb{Z}_n \setminus \{i\}$ , then there is exactly one vertex  $y \in N[x]$  such that  $out(y) \in AN_n^j$ .

**Lemma 5.** For  $n \geq 4$ , let  $H = AN_n - V(AN_n^i)$  for some  $i \in \mathbb{Z}_n$ . Then  $\kappa(H) = n - 2$ .

**Proof.** By Lemma 2, any two distinct vertices of  $AN_n^i$  have different out-neighbors in  $H$ . Thus, the removal of  $AN_n^i$  from  $AN_n$  leads to that the degree of a vertex in  $H$  decreases by at most one. Thus, we have  $\delta(H) = n - 2$ , and it follows that  $\kappa(H) \leq \delta(H) = n - 2$ . To show  $\kappa(H) \geq n - 2$ , we claim that there exists an  $(n - 2)$ -path container between any two vertices  $x, y \in V(H)$ . The assertion is clearly true if  $x, y \in AN_n^j$  for some  $j \in \mathbb{Z}_n \setminus \{i\}$  because  $\kappa(AN_n^j) = \kappa(AN_{n-1}) = n - 2$ . We now consider  $x \in AN_n^j$  and  $y \in AN_n^k$ , where  $j, k \in \mathbb{Z}_n \setminus \{i\}$  and  $j \neq k$ . By Lemma 4, there exists exactly one vertex of  $N[x]$  (resp.  $N[y]$ ) such that its out-neighbor is contained in  $AN_n^\ell$  for each  $\ell \in \mathbb{Z}_n \setminus \{j\}$  (resp. for each  $\ell \in \mathbb{Z}_n \setminus \{k\}$ ). For each  $\ell \in \mathbb{Z}_n \setminus \{i, j, k\}$ , we let  $(u_\ell, v_\ell) \in E_n^{j,\ell}$ , where  $u_\ell \in N[x] \subseteq AN_n^j$  and  $v_\ell \in AN_n^\ell$ , and  $(w_\ell, z_\ell) \in E_n^{k,\ell}$ , where  $w_\ell \in N[y] \subseteq AN_n^k$  and  $z_\ell \in AN_n^\ell$ , be such external edges. Note that it is possible that  $u_\ell = x$  or  $w_\ell = y$  for a certain  $\ell$ . By the property (5) of Lemma 2,  $(u_\ell, v_\ell)$  and  $(w_\ell, z_\ell)$  are independent edges in  $AN_n$ , and thus  $v_\ell \neq z_\ell$ . Since  $AN_n^\ell$  is connected and  $v_\ell, z_\ell \in AN_n^\ell$ , there is a path, say  $P'_\ell$ , that connects  $v_\ell$  and  $z_\ell$  in  $AN_n^\ell$ . Thus, we have  $n - 3$  internally disjoint  $(x, y)$ -paths  $P_\ell$  for  $\ell \in \mathbb{Z}_n \setminus \{i, j, k\}$  by setting

$$P_\ell = \{(x, u_\ell), (u_\ell, v_\ell)\} \cup P'_\ell \cup \{(z_\ell, w_\ell), (w_\ell, y)\}.$$

To complete the proof, we need an extra  $(x, y)$ -path  $P$  such that  $V(P) \cap V(P_\ell) = \{x, y\}$  for all  $\ell \in \mathbb{Z}_n \setminus \{i, j, k\}$ . Let  $U = \{u_\ell : \ell \in \mathbb{Z}_n \setminus \{i, j, k\}\} \setminus \{x\}$  and

$W = \{w_\ell: \ell \in \mathbb{Z}_n \setminus \{i, j, k\}\} \setminus \{y\}$ . Also, define  $H_1 = AN_n^j - U$  and  $H_2 = AN_n^k - W$ . Since  $\kappa(AN_n^j) = \kappa(AN_n^k) = n - 2$  and  $H_1$  is obtained from  $AN_n^j$  (resp.  $H_2$  is obtained from  $AN_n^k$ ) by removing at most  $n - 3$  vertices, this implies that  $H_1$  (resp.  $H_2$ ) is connected. Let  $H$  be the subgraph of  $AN_n$  induced by  $V(H_1) \cup V(H_2)$ . Since the edge connecting a vertex of  $N[x]$  with out-neighbor in  $AN_n^k$  (resp. a vertex of  $N[y]$  with out-neighbor in  $AN_n^j$ ) does not be removed from  $H$ , it follows that  $H$  is connected and it contains an  $(x, y)$ -path  $P$  as our desired.  $\square$

### 3 The Generalized 3-Connectivity $\kappa'_3(AN_n)$

To evaluate the size of the connected components of  $AN_n$  ( $n \geq 4$ ) with a set of faulty vertices, Zhou and Xiao [36] gave the following property.

**Lemma 6** (see [36]). *For  $n \geq 5$ , if  $F$  is a vertex-cut of  $AN_n$  with  $|F| \leq 2n - 5$ , then one of the following conditions holds:*

- (1)  $AN_n - F$  has two components, one of which is a trivial component (i.e., a singleton).
- (2)  $AN_n - F$  has two components, one of which is an edge, say  $(u, v)$ . In particular, if  $|F| = 2n - 5$ ,  $F$  is composed of all neighbors of  $u$  and  $v$ , excluding  $u$  and  $v$ .

Note that if  $F$  is a vertex-cut of  $AN_4$  with  $|F| = 3$ , then  $AN_4 - F$  has two component, one of which is a singleton, an edge, or a 3-cycle. In particular, for the case of taking a 3-cycle as a component, it is easy to check the graph  $AN_4 - \{1342, 2143, 3241\}$  in Fig. 1.

From the definition of generalized connectivity, it is clear that  $\kappa'_3(AN_3) = 1$ . Also, by brute-force checking  $AN_4$  and  $AN_5$ , we found that the removal of no more than four vertices in  $AN_4$  (resp., six vertices in  $AN_5$ ) results in a graph that is connected or contains exactly two components. Thus, the following lemma establishes the lower bound of  $\kappa'_3(AN_n)$  for  $n = 4, 5$ .

**Lemma 7.**  $\kappa'_3(AN_4) \geq 5$  and  $\kappa'_3(AN_5) \geq 7$ .

**Proof of Theorem 1.** We first prove  $\kappa'_3(AN_n) \leq 2n - 3$ . Consider two vertices  $x, y \in AN_n$  where  $x = (p_1 p_2 p_3 \dots p_n) \in AN_n^{p_n}$  and  $y = (p_3 p_2 p_n p_4 \dots p_{n-1} p_1) \in AN_n^{p_1}$ . It is clear that  $x$  and  $y$  are nonadjacent and they have a common neighbor, say  $z = (p_2 p_3 p_1 p_4 \dots p_n) = x g_3^- = y g_n^*$ . Thus,  $z = \text{out}(y) \in N(x)$  and  $\text{out}(x) = (p_2 p_1 p_n p_4 \dots p_{n-1} p_3) \in AN_n^{p_3}$ . Let  $F = N(x) \cup N(y) \cup \{\text{out}(x)\}$ . Then, the removal of  $F$  from  $AN_n$  results in a disconnected graph that contains three components, including two isolated vertices  $x$  and  $y$  as components. By the property (2) of Lemma 2, since every vertex has  $n - 2$  in-neighbors, it follows that  $\kappa'_3(AN_n) \leq |F| = 2(n - 2) + 1 = 2n - 3$ .

Next, we will prove  $\kappa'_3(AN_n) \geq 2n - 3$ . For  $n = 4$  or  $n = 5$ , the results are acquired in Lemma 7. We now consider  $n \geq 6$  and let  $F$  be any vertex-cut in  $AN_n$  such that  $|F| \leq 2n - 4$ . For convenience, vertices in  $F$  (resp., not in  $F$ ) are

called faulty vertices (resp., fault-free vertices). By Lemma 6, if  $|F| \leq 2n - 5$ , then  $AN_n - F$  contains exactly two components, one of which is either a singleton or an edge. To complete the proof, we need to show that the same result holds for  $|F| = 2n - 4$ . Let  $F_i = F \cap V(AN_n^i)$  and  $f_i = |F_i|$  for each  $i \in \mathbb{Z}_n$ . We claim that there exists some subnetwork, say  $AN_n^i$ , such that it contains  $f_i \geq n - 2$  faulty vertices. Since  $|F| = 2n - 4$ , if it is so, then there are at most two such subnetworks. Suppose not, i.e., every subnetwork  $AN_n^j$  for  $j \in \mathbb{Z}_n$  has  $f_j \leq n - 3$  faulty vertices. Since  $AN_n^j$  is  $(n - 2)$ -connected,  $AN_n^j - F_j$  remains connected for each  $j \in \mathbb{Z}_n$ . Recall the property (4) of Lemma 2 that there are  $(n - 2)!/2$  independent edges between  $AN_n^j$  and  $AN_n^{j'}$  for each pair  $j, j' \in \mathbb{Z}_n$  with  $j \neq j'$ . Since  $(n - 2)!/2 > 2(n - 3) \geq f_j + f_{j'}$  for  $n \geq 6$ , it guarantees that the two subgraphs  $AN_n^j - F_j$  and  $AN_n^{j'} - F_{j'}$  are connected by an external edge in  $AN_n - F$ . Thus,  $AN_n - F$  is connected, and this contradicts to the fact that  $F$  is a vertex-cut in  $AN_n$ . Moreover, for such subnetworks  $AN_n^i$  such that  $f_i \geq n - 2$ , it is sure that some of  $F_i$  must be a vertex-cut of  $AN_n^i$ ; otherwise,  $AN_n - F$  is connected, a contradiction. We now consider the following two cases:

**Case 1:** There is exactly one such subnetwork, say  $AN_n^i$ , such that it contains  $f_i \geq n - 2$  faulty vertices. In this case, we have  $f_j \leq n - 3$  for all  $j \in \mathbb{Z}_n \setminus \{i\}$  and  $F_i$  is a vertex-cut of  $AN_n^i$ . Let  $H$  be the subgraph of  $AN_n$  induced by the fault-free vertices outside  $AN_n^i$ , i.e.,  $H = AN_n - (V(AN_n^i) \cup F)$ . Since every subnetwork  $AN_n^j$  in  $H$  has  $f_j \leq n - 3$  faulty vertices, it is sure that  $H$  is connected. We denote by  $C$  the component of  $AN_n - F$  that contains  $H$  as its subgraph. Consider the following scenarios:

**Case 1.1:**  $f_i = 2n - 4$ . In this case, there are no faulty vertices outside  $AN_n^i$ . That is,  $H = AN_n - V(AN_n^i)$ . Indeed, this case is impossible because if it is the case, then every vertex of  $AN_n^i - F_i$  has the fault-free out-neighbor in  $H$ . Thus,  $AN_n^i - F_i$  belongs to  $C$ , and it follows that  $AN_n - F$  is connected, a contradiction.

**Case 1.2:**  $f_i = 2n - 5$ . Let  $u \in F \setminus F_i$  be the unique faulty vertex outside  $AN_n^i$ . That is,  $H = AN_n - (V(AN_n^i) \cup \{u\})$ . Since  $F_i$  is a vertex-cut of  $AN_n^i$ , we assume that  $AN_n^i - F_i$  is divided into  $k$  disjoint connected components, say  $C_1, C_2, \dots, C_k$ . For each  $j \in \mathbb{Z}_k$ , if  $|C_j| \geq 2$ , then there is at least one vertex of  $C_j$  with its out-neighbor in  $H$ , and thus  $C_j$  belongs to  $C$ . We now consider a component that is a singleton, say  $C_j = \{v\}$ . If  $\text{out}(v) \neq u$ , then  $\text{out}(v)$  must be contained in  $H$ , and thus  $C_j$  belongs to  $C$ . Clearly, there exists at most one component  $C_j = \{v\}$  such that  $\text{out}(v) = u$ . In this case,  $AN_n - F$  has exactly two components  $\{v\}$  and  $C$ .

**Case 1.3:**  $f_i = 2n - 6$ . Let  $u_1, u_2 \in F \setminus F_i$  be the two faulty vertices outside  $AN_n^i$ . That is,  $H = AN_n - (V(AN_n^i) \cup \{u_1, u_2\})$ . Since  $F_i$  is a vertex-cut of  $AN_n^i$ , we assume that  $AN_n^i - F_i$  is divided into  $k$  disjoint connected components, say  $C_1, C_2, \dots, C_k$ . For each  $j \in \mathbb{Z}_k$ , if  $|C_j| \geq 3$ , then there is at least one vertex of  $C_j$  with its out-neighbor in  $H$ , and thus  $C_j$  belongs to  $C$ . We now consider a component  $C_j$  with  $|C_j| = 2$ , i.e.,  $C_j$  is an edge, say  $(v_1, v_2)$ . If  $\{\text{out}(v_1), \text{out}(v_2)\} \neq \{u_1, u_2\}$ , then at least one of  $\text{out}(v_1)$  and  $\text{out}(v_2)$  must be

contained in  $H$ , and thus  $C_j$  belongs to  $C$ . Since  $f_i = 2n - 6$  and  $(v, w)$  has  $2n - 6$  neighbors (not including  $v$  and  $w$ ) in  $AN_n^i$ , there exists at most one component  $C_j = \{(v_1, v_2)\}$  such that  $\{\text{out}(v_1), \text{out}(v_2)\} = \{u_1, u_2\}$ . If it is the case of existence, then  $AN_n - F$  has exactly two components  $\{(v_1, v_2)\}$  and  $C$ . Finally, we consider a component that is a singleton. Since  $f_i = 2n - 6$  and every vertex has degree  $n - 2$  in  $AN_n^i$ , we have  $n - 2 < 2n - 6 < 2(n - 2)$  for  $n \geq 6$ . Thus, at most one such component exists, say  $C_j = \{v\}$ . If  $\text{out}(v) \notin \{u_1, u_2\}$ , then  $\text{out}(v)$  is contained in  $H$ , and thus  $C_j$  belongs to  $C$ . Otherwise,  $AN_n - F$  has exactly two components  $\{v\}$  and  $C$ .

**Case 1.4:**  $n - 2 \leq f_i \leq 2n - 7$ . In this case, there exist at least three faulty vertices outside  $AN_n^i$  (i.e.,  $(2n - 4) - (2n - 7) = 3$ ). Since  $AN_n^i$  is isomorphic to  $AN_{n-1}$  and  $F_i$  is a vertex-cut of  $AN_n^i$  with no more than  $2(n - 1) - 5$  vertices, by Lemma 6,  $AN_n^i - F_i$  has exactly two components, one of which is either a singleton or an edge. Let  $C_1$  and  $C_2$  be such two components for which  $1 \leq |C_1| \leq 2 < |C_2|$ . More precisely,  $|C_2| = |V(AN_n^i)| - f_i - |C_1| \geq (n - 1)!/2 - f_i - 2 > (2n - 4) - f_i = |F| - f_i$  for  $n \geq 6$ , where the last term  $|F| - f_i$  is the number of faulty vertices outside  $AN_n^i$ . Clearly, the above inequality indicates that there exist some vertices of  $C_2$  such that their out-neighbors are contained in  $H$ , even if all out-neighbors of vertices in  $F \setminus F_i$  are contained in  $C_2$ . Thus,  $C_2$  belongs to  $C$ . Also, if there is a vertex  $v \in C_1$  with its out-neighbor in  $H$ , then  $C_1$  belongs to  $C$ . Otherwise,  $AN_n - F$  has exactly two components, one of which is either a singleton or an edge.

**Case 2:** There exist exactly two subnetworks, say  $AN_n^i$  and  $AN_n^j$ , such that  $f_i \geq n - 2$  and  $f_j \geq n - 2$ . Since  $F$  is a vertex-cut of  $AN_n$ , it implies that at least one of the subgraphs  $AN_n^i - F_i$  and  $AN_n^j - F_j$  must be disconnected. Also, since  $|F| = 2n - 4$ , we have  $f_i = f_j = n - 2$  and  $f_k = 0$  for all  $k \in \mathbb{Z}_n \setminus \{i, j\}$ . Moreover, since both  $AN_n^i$  and  $AN_n^j$  are isomorphic to  $AN_{n-1}$  and  $f_i = f_j = n - 2 \leq 2(n - 1) - 5$  for  $n \geq 6$ , by Lemma 6, if  $AN_n^i - F_i$  (resp.,  $AN_n^j - F_j$ ) is disconnected, then it contains exactly two components. For the case that exactly one of  $AN_n^i - F_i$  and  $AN_n^j - F_j$  is disconnected, through an argument similar to the analysis of Case 1.4 for  $f_i = n - 2$  or  $f_j = n - 2$ , we can show that  $AN_n - F$  contains exactly two components, one of which is either a singleton or an edge. Now, we consider that both  $F_i$  and  $F_j$  are vertex-cuts of  $AN_n^i$  and  $AN_n^j$ , respectively, such that  $AN_n^i - F_i = C_i \cup C'_i$  and  $AN_n^j - F_j = C_j \cup C'_j$ , where  $C_i, C'_i, C_j, C'_j$  are disjoint components with  $1 \leq |C_i| \leq 2 < |C'_i|$  and  $1 \leq |C_j| \leq 2 < |C'_j|$ . Let  $H = AN_n - (V(AN_n^i) \cup V(AN_n^j))$  and denote by  $C$  the component of  $AN_n - F$  that contains  $H$  as its subgraph. Since  $|C'_i| = |V(AN_n^i)| - f_i - |C_i| \geq (n - 1)!/2 - (n - 2) - 2 > n - 2 = f_j$  for  $n \geq 6$ , there is at least one vertex of  $C'_i$  with its out-neighbor in  $H$ , and thus  $C'_i$  belongs to  $C$ . Similarly, we can show that  $C'_j$  belongs to  $C$ . Next, we consider  $|C_i| = 2$  or  $|C_j| = 2$ . Suppose that  $C_i$  (resp.,  $C_j$ ) is an edge, say  $(u, v)$ . By Lemma 3,  $\text{out}(u)$  and  $\text{out}(v)$  must be contained in different subnetworks, and thus at least one of which must be in  $H$ . It follows that  $C_i$  (resp.,  $C_j$ ) belongs to  $C$ . Finally, we consider  $C_i = \{u\}$  and  $C_j = \{v\}$ . In this case, since  $f_i = f_j = n - 2$ , we have  $F_i = N(u)$  and  $F_j = N(v)$ . We describe all subcases as follows: If



$\text{out}(u) \in H$  (resp.,  $\text{out}(v) \in H$ ), then  $C_i$  (resp.,  $C_j$ ) belongs to  $C$ . If  $\text{out}(u) \in F_j$  and  $\text{out}(v) \in F_i$ , then  $\{u, \text{out}(u), v, \text{out}(v)\}$  forms a 4-cycle, a contradiction to the property (1) of Lemma 2. If exactly one of the conditions  $\text{out}(u) \in F_j$  and  $\text{out}(v) \in F_i$  holds, then  $AN_n - F$  has exactly two components, one of which is a singleton  $\{u\}$  or  $\{v\}$ . If  $\text{out}(u) = v$ , then  $AN_n - F$  has exactly two components, one of which is an edge  $(u, v)$ .  $\square$

Note that the lower bound of  $\kappa'_3(AN_n)$  in the above proof mainly concern the condition that  $n \geq 6$  and  $|F| = 2n - 4$ . As a remark, if  $F$  is a vertex-cut of  $AN_5$  with  $|F| = 6$ , then  $AN_5 - F$  has two component, one of which is a singleton, an edge, or a 3-cycle. In particular, for the case of taking a 3-cycle as a component, we can consider  $F = \{13425, 13542, 21435, 21543, 32415, 32541\}$  in Fig. 2. Also, if  $F$  is a vertex-cut of  $AN_4$  with  $|F| = 4$ , then  $AN_4 - F$  has two component, one of which is a singleton, an edge, a 3-cycle, a 2-path (i.e.,  $P_3$ ) or a paw (i.e.,  $K_{1,3} + e$ ). For the case of taking a 2-path (resp., a paw) as a component, we can consider  $F = \{2431, 3241, 1342, 1423\}$  (resp.,  $F = \{3241, 1342, 4213, 1423\}$ ) in Fig. 1. Thus, from the proof of Theorem 1 together with the remark, we obtain the following result, which is an extension of Lemma 6.

**Corollary 1.** *Let  $F$  is a vertex-cut of  $AN_n$  with  $|F| \leq 2n - 4$ . Then, the following conditions hold:*

- (1) *If  $n = 4$ , then  $AN_n - F$  has two components, one of which is a singleton, an edge, a 3-cycle, a 2-path, or a paw.*
- (2) *If  $n = 5$ , then  $AN_n - F$  has two components, one of which is a singleton, an edge, or a 3-cycle.*
- (3) *If  $n \geq 6$ , then  $AN_n - F$  has two components, one of which is either a singleton or an edge.*

## 4 The Generalized 3-Connectivity $\kappa_3(AN_n)$

The following two properties establish an upper bound and a lower bound of  $\kappa_3(G)$  for a connected graph  $G$ , respectively.

**Lemma 8** (see [20]). *Let  $G$  be a connected graph with minimum degree  $\delta$ . Then  $\kappa_3(G) \leq \delta$ . In particular, if there exist two adjacent vertices of degree  $\delta$  in  $G$ , then  $\kappa_3(G) \leq \delta - 1$ .*

**Lemma 9** (see [20]). *Let  $G$  be a connected graph. For every two nonnegative integers  $k$  and  $r \in \{0, 1, 2, 3\}$ , if  $\kappa(G) = 4k + r$ , then  $\kappa_3(G) \geq 3k + \lceil r/2 \rceil$ .*

**Proof of Theorem 2.** Since  $AN_n$  is  $(n - 1)$ -regular, by Lemma 8, we have  $\kappa_3(AN_n) \leq \delta(AN_n) - 1 = n - 2$ . We now prove that  $\kappa_3(AN_n) \geq n - 2$  by induction. For  $n = 3$ , since  $AG_3$  is isomorphic to a 3-cycle, it is obvious that  $\kappa_3(AN_3) \geq 1 = n - 2$ . For  $n = 4$ , since  $\kappa(AN_4) = n - 1 = 3$ , by Lemma 9 we have  $\kappa_3(AN_4) \geq \lceil 3/2 \rceil = n - 2$ . Suppose that  $n \geq 5$  and the assertion holds for  $AN_{n-1}$ . Let  $S = \{x, y, z\}$ , where  $x, y$  and  $z$  are three distinct vertices of  $AN_n$ . Without loss of generality, we may consider the following three cases:



**Case 1:**  $x, y, z \in AN_n^1$  (i.e.,  $x, y$  and  $z$  belong to the same subnetwork). Let  $H = AN_n - V(AN_n^1)$ . By Lemma 5,  $H$  is connected. Since  $AN_n^1$  is isomorphic to  $AN_{n-1}$ , by induction hypothesis we have  $\kappa_3(AN_n^1) \geq (n-1) - 2 = n-3$ . Thus, there exist  $n-3$  IDTs that connect  $S$  in  $AN_n^1$ . Let  $S' = \{x', y', z'\}$ , where  $x', y'$  and  $z'$  are the out-neighbors of  $x, y$  and  $z$  respectively. Thus, it follows from the property (3) of Lemma 2 that all vertices of  $S'$  are distinct. Since  $S' \subset V(H)$  and  $H$  is connected, there is a tree  $T'$  that connects  $S'$  in  $H$ . Let  $T$  be the tree obtained from  $T'$  by adding three pendant edges  $(x, x'), (y, y')$  and  $(z, z')$ . Then,  $T$  is a tree connecting  $S$  in  $AN_n$  and  $V(T) \cap V(AN_n^1) = S$ . Now, counting the  $n-3$  IDTs we mentioned before together with  $T$ , there exist at least  $n-2$  IDTs connecting  $S$  in  $AN_n$ , and hence  $\kappa_3(AN_n) \geq n-2$ .

**Case 2:**  $x, y \in AN_n^1$  and  $z \in AN_n^2$  (i.e.,  $x, y$  and  $z$  belong to two subnetworks). Let  $H = AN_n - V(AN_n^1)$ . By Lemma 5 and since  $AN_n^1$  is isomorphic to  $AN_{n-1}$ , we have  $\kappa(H) = \kappa(AN_n^1) = n-2$ , and thus there is an  $(n-2)$ -path container between  $x$  and  $y$  in  $AN_n^1$ . Let  $P_1, P_2, \dots, P_{n-2}$  be such internally disjoint  $(x, y)$ -paths. For each path  $P_i$ , we choose a vertex  $w_i \in V(P_i)$ , and let  $W = \{w_1, w_2, \dots, w_{n-2}\}$ . Note that at most one of these paths, say  $P_1$ , has length 1. If so, we choose  $w_1 = x$ . For each  $i \in \mathbb{Z}_{n-2}$ , we let  $w'_i = \text{out}(w_i)$ . By the property (3) of Lemma 2, we have  $w'_i \neq w'_j$  for  $i, j \in \mathbb{Z}_{n-2}$  and  $i \neq j$ . Let  $W' = \{w'_1, w'_2, \dots, w'_{n-2}\}$ . Since  $W' \subset V(H)$  and  $\kappa(H) = n-2$ , by Lemma 1, if  $z \notin W'$ , there exists an  $(n-2)$ -fan from  $z$  to  $W'$  in  $H$ . For  $i \in \mathbb{Z}_{n-2}$ , let  $P'_i$  be such  $(z, w'_i)$ -path in this fan. On the other hand, if  $z = w'_i$  for some  $i \in \mathbb{Z}_{n-2}$  (i.e.,  $z \in W'$ ), we may consider that  $P'_i$  contains exactly one vertex  $z$ . Now, for each  $i \in \mathbb{Z}_{n-2}$ , we can construct a tree  $T_i$  in  $AN_n$  by setting  $T_i = P_i \cup \{(w_i, w'_i)\} \cup P'_i$ . As a result,  $T_1, T_2, \dots, T_{n-2}$  are  $n-2$  IDTs that connect  $S$  in  $AN_n$ , and hence  $\kappa_3(AN_n) \geq n-2$ .

**Case 3:**  $x \in AN_n^1, y \in AN_n^2$  and  $z \in AN_n^3$  (i.e.,  $x, y$  and  $z$  belong to different subnetworks). For each  $i \in \{4, 5, \dots, n\}$ , we consider the following external edges between subnetworks:  $(u_i, u'_i) \in E_n^{1,i}$  where  $u_i \in N[x] \subseteq AN_n^1$  and  $u'_i \in AN_n^i$ ,  $(v_i, v'_i) \in E_n^{2,i}$  where  $v_i \in N[y] \subseteq AN_n^2$  and  $v'_i \in AN_n^i$ , and  $(w_i, w'_i) \in E_n^{3,i}$  where  $w_i \in N[z] \subseteq AN_n^3$  and  $w'_i \in AN_n^i$ . Note that it is possible that  $u_i = x, v_i = y$  or  $w_i = z$  for a certain  $i$ . By the property (5) of Lemma 2,  $(u_i, u'_i), (v_i, v'_i)$  and  $(w_i, w'_i)$  are independent edges in  $AN_n$ , and thus the three vertices  $u'_i, v'_i$  and  $w'_i$  are distinct. Since  $AN_n^i$  is connected and  $u'_i, v'_i, w'_i \in AN_n^i$ , let  $T'_i$  be a tree in  $AN_n^i$  that connects the three vertices  $u'_i, v'_i$  and  $w'_i$ . Now, we can construct a tree  $T_i$  in  $AN_n$  by setting

$$T_i = T'_i \cup \{(x, u_i), (u_i, u'_i), (y, v_i), (v_i, v'_i), (z, w_i), (w_i, w'_i)\}.$$

Thus,  $T_4, T_5, \dots, T_n$  are  $n-3$  IDTs that connect  $S$  in  $AN_n$ . To complete the proof, we need an extra tree  $T$  to connect  $S$  such that  $V(T) \cap V(T_i) = S$  for all  $i \in \{4, 5, \dots, n\}$ . Let  $H_1 = AN_n^1 - (\{u_4, u_5, \dots, u_n\} \setminus \{x\})$ . Since  $\kappa(AN_n^1) = n-2$  and  $H_1$  is obtained from  $AN_n^1$  by removing at most  $n-3$  vertices, this implies that  $H_1$  is connected. Similarly, the two subgraphs  $H_2 = AN_n^2 - (\{v_4, v_5, \dots, v_n\} \setminus \{y\})$  and  $H_3 = AN_n^3 - (\{w_4, w_5, \dots, w_n\} \setminus \{z\})$  are also connected. Moreover, since each  $H_j, j \in \{1, 2, 3\}$ , is obtained from  $AN_n^j$  by removing vertices without out-neighbors in  $AN_n^k$  for  $k \in \{1, 2, 3\} \setminus \{j\}$ , by Lemma 4,

the subgraph of  $AN_n$  induced by  $\cup_{i=1}^3 V(H_i)$  is connected and it contains a tree  $T$  connecting  $S$  as our desired.  $\square$

## 5 Concluding Remarks

In this paper, we determine the exact values of  $\kappa'_3(AN_n)$  and  $\kappa_3(AN_n)$ . A natural counterpart of the generalized connectivity is the so-called generalized edge-connectivity introduced in [27]. For  $S \subseteq V(G)$ , let  $\psi'(S)$  denote the maximum number of edge-disjoint trees that connect  $S$  in  $G$ . The *generalized  $k$ -edge-connectivity* of  $G$  is defined by  $\lambda_k(G) = \min\{\psi'(S) : S \subseteq V(G) \text{ and } |S| = k\}$ . An easy observation shows that  $\kappa_k(G) \leq \lambda_k(G) \leq \delta(G)$ . Also, similar to Lemma 8, a result in [22] showed that if there exist two adjacent vertices with the minimum degree  $\delta$  in  $G$ , then  $\lambda_3(G) \leq \delta - 1$ . Accordingly, by Theorem 2, we conclude that  $\lambda_3(AN_n) = n - 2$  for  $n \geq 3$ .

So far it seems that less effort has been devoted on the study of the two types of generalized  $k$ -connectivity for larger  $k$ . As a future work, we would like to study in this direction, especially, for some popular interconnection networks such as hypercubes, star graphs, high dimensional tori, and their variants.

**Acknowledgments.** This research was partially supported by MOST grants 104-2221-E-141-002-MY3 (Jou-Ming Chang), 105-2221-E-131-027 (Kung-Jui Pai), 106-2221-E-141-001 (Jinn-Shyong Yang) and 104-2221-E-262-005 (Ro-Yu Wu) from the Ministry of Science and Technology, Taiwan.

## References

1. Chartrand, G., Kapoor, S.F., Lesniak, L., Lick, D.R.: Generalized connectivity in graphs. *Bull. Bombay Math. Colloq.* **2**, 1–6 (1984)
2. Chartrand, G., Okamoto, F., Zhang, P.: Rainbow trees in graphs and generalized connectivity. *Networks* **55**, 360–367 (2010)
3. Chen, L., Li, X., Liu, M., Mao, Y.: A solution to a conjecture on the generalized connectivity of graphs. *J. Comb. Optim.* **33**, 275–282 (2017)
4. Chen, B., Xiao, W., Parhami, B.: Internode distance and optimal routing in a class of alternating group networks. *IEEE Trans. Comput.* **12**, 1645–1648 (2006)
5. Day, D.P., Oellermann, O.R., Swart, H.C.: The  $l$ -connectivity function of trees and complete multipartite graphs. *J. Comb. Math. Comb. Comput.* **10**, 183–192 (1991)
6. Dirac, G.A.: In abstrakten Graphen vorhandene vollstndige 4-Graphen und ihre Unterteilungen. *Math. Nachr.* **22**, 61–85 (1960)
7. Hager, M.: Pendant tree-connectivity. *J. Comb. Theory Ser. B* **38**, 179–189 (1985)
8. Hsu, D.F.: On container width and length in graphs, groups, and networks. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E77-A**, 668–680 (1994)
9. Hao, R.-X., Zhou, J.-X.: Characterize a kind of fault tolerance of alternating group network. *Acta Math. Sinica (Chin. Ser.)* **55**, 1055–1066 (2012)
10. Ji, Y.-H.: A class of Cayley networks based on the alternating groups. *Adv. Math.* **4**, 361–362 (1998). (in Chinese)
11. Jwo, J., Lakshmivarahan, S., Dhall, S.K.: A new class of interconnection networks based on the alternating group. *Networks* **23**, 315–326 (1993)

12. Li, H., Li, X., Sun, Y.: The generalized 3-connectivity of Cartesian product graphs. *Discrete Math. Theor. Comput. Sci.* **14**, 43–54 (2012)
13. Li, H., Li, X., Mao, Y., Sun, Y.: Note on the generalized connectivity. *ARS Comb.* **114**, 193–202 (2014)
14. Li, H., Ma, Y., Yang, W., Wang, Y.: The generalized 3-connectivity of graph product. *Appl. Math. Comput.* **295**, 77–83 (2017)
15. Li, S., Li, W., Li, X.: The generalized connectivity of complete bipartite graphs. *ARS Comb.* **104**, 65–79 (2012)
16. Li, S., Li, W., Li, X.: The generalized connectivity of complete equipartition 3-partite graphs. *Bull. Malays. Math. Sci. Soc.* **37**, 103–121 (2014)
17. Li, S., Li, W., Shi, Y., Sun, H.: On minimally 2-connected graphs with generalized connectivity  $\kappa_3 = 2$ . *J. Comb. Optim.* **34**, 141–164 (2017)
18. Li, S., Li, X.: Note on the hardness of generalized connectivity. *J. Comb. Optim.* **24**, 389–396 (2012)
19. Li, S., Li, X., Shi, Y.: The minimal size of a graph with generalized connectivity  $\kappa_3 = 2$ . *Australas. J. Comb.* **51**, 209–220 (2011)
20. Li, S., Li, X., Zhou, W.: Sharp bounds for the generalized connectivity  $\kappa_3(G)$ . *Discrete Math.* **310**, 2147–2163 (2010)
21. Li, S., Tu, J., Yu, C.: The generalized 3-connectivity of star graphs and bubble-sort graphs. *Appl. Math. Comput.* **274**, 41–46 (2016)
22. Li, X., Mao, Y.: The minimal size of a graph with given generalized 3-edge-connectivity. [arXiv:1201.3699v2](https://arxiv.org/abs/1201.3699v2) (2013)
23. Li, X., Mao, Y.: The generalized 3-connectivity of lexicographic product graphs. *Discrete Math. Theor. Comput. Sci.* **16**, 339–354 (2014)
24. Li, X., Mao, Y.: Graphs with large generalized (edge-) connectivity. *Discuss. Math. Graph Theor.* **36**, 931–958 (2016)
25. Li, X., Mao, Y.: *Generalized Connectivity of Graphs*. Springer Briefs in Mathematics. Springer, New York (2016). <https://doi.org/10.1007/978-3-319-33828-6>
26. Li, X., Mao, Y.: A survey on the generalized connectivity of graphs. [arXiv:1207.1838v9](https://arxiv.org/abs/1207.1838v9) (2014)
27. Li, X., Mao, Y., Sun, Y.: On the generalized (edge-) connectivity of graphs. *Australas. J. Comb.* **58**, 304–319 (2014)
28. Lin, S., Zhang, Q.: The generalized 4-connectivity of hypercubes. *Discrete Appl. Math.* **220**, 60–67 (2017)
29. Menger, K.: Zur allgemeinen Kurventheorie. *Fund. Math.* **10**, 96–115 (1927)
30. Oellermann, O.R.: On the  $l$ -connectivity of a graph. *Graph Comb.* **3**, 285–291 (1987)
31. Oellermann, O.R.: A note on the  $l$ -connectivity function of a graph. *Congressus Numerantium* **60**, 181–188 (1987)
32. Okamoto, F., Zhang, P.: The tree connectivity of regular complete bipartite graphs. *J. Comb. Math. Comb. Comput.* **74**, 279–293 (2010)
33. Sun, Y., Li, X.: On the difference of two generalized connectivities of a graph. *J. Comb. Optim.* **33**, 283–291 (2017)
34. West, D.B.: *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River (2001)
35. Whitney, H.: Congruent graphs and the connectivity of graphs and the connectivity of graphs. *Am. J. Math.* **54**, 150–168 (1932)
36. Zhou, S., Xiao, W.: Conditional diagnosability of alternating group networks. *Inf. Process. Lett.* **110**, 403–409 (2010)
37. Zhou, S., Xiao, W., Parhami, B.: Construction of vertex-disjoint paths in alternating group networks. *J. Supercomput.* **54**, 206–228 (2010)



# On the Longest Spanning Tree with Neighborhoods

Ke Chen and Adrian Dumitrescu<sup>(✉)</sup>

University of Wisconsin–Milwaukee, Milwaukee, WI, USA  
{kechen,dumitres}@uwm.edu

**Abstract.** We study a maximization problem for geometric network design. Given a set of  $n$  compact neighborhoods in  $\mathbb{R}^d$ , select a point in each neighborhood, so that the longest spanning tree on these points (as vertices) has maximum length. Here we give an approximation algorithm with ratio 0.511, which represents the first, albeit small, improvement beyond  $1/2$ . While we suspect that the problem is NP-hard already in the plane, this issue remains open.

**Keywords:** Maximum (longest) spanning tree · Neighborhood  
Geometric network · Metric problem · Approximation algorithm

## 1 Introduction

In the *Euclidean Maximum Spanning Tree Problem* (EMST), given a set of points in the Euclidean space  $\mathbb{R}^d$ ,  $d \geq 2$ , one seeks a tree that connects these points (as vertices) and has maximum length. The problem is easily solvable in polynomial time by Prim's algorithm or by Kruskal's algorithm; algorithms that take advantage of the geometry are also available [13]. In the *Longest Spanning Tree with Neighborhoods* (MAX-ST-N), each point is replaced by a point-set, called *region* or *neighborhood*, and the tree must connect  $n$  representative points, one chosen from each region (duplicate representatives are allowed), and the tree has maximum length. The tree edges are straight line segments connecting pairs of points in distinct regions; for obvious reasons we refer to these edges as *bichromatic*. As one would expect, the difficulty lies in choosing the representative points; once these points are selected, the problem is reduced to the graph setting and is thus easily solvable.

The input  $\mathcal{N}$  consists of  $n$  (possibly disconnected) neighborhoods. For simplicity, it is assumed that each neighborhood is a union of polyhedral regions; the total vertex complexity of the input is  $N$ . However, it will be apparent from the context that our methods extend to a broader class of regions, those approximable by unions of polyhedral regions within a prescribed accuracy (for instance unions of balls of arbitrary radii, etc.).

*Examples.* It is worth noting that a greedy algorithm does not necessarily find an optimal tree. Let  $\mathcal{N} = \{X_1, X_2, X_3\}$ , where  $X_1 = \{a, b\}$ ,  $X_2 = \{a, c\}$ ,  $X_3 = \{d\}$ ,  $\Delta abc$  is a unit equilateral triangle and  $d$  is the midpoint of  $bc$ ; see Fig. 1 (left).



**Fig. 1.** Left: an example on which the greedy algorithm is suboptimal. Right: an example of a long (still suboptimal) spanning tree with 10 regions  $\mathcal{N} = \{A, S \cup S, E \cup E \cup E, T \cup T, O \cup O, F, N \cup N, R, G, I\}$  (some regions are disconnected); the blue segments form a spanning tree on  $\mathcal{N}$  and the green dots are the chosen representative points. (Color figure online)

A (natural) greedy algorithm chooses two points attaining a maximum inter-point distance with points in distinct regions, and then repeatedly chooses a point in each new region as far as possible from some selected point. Here the selection  $b \in X_1, c \in X_2, d \in X_3$  yields a spanning tree in the form of a star centered at  $v_1 = b$  of length  $3/2$ ; on the other hand, selecting vertices  $v_i \in X_i, i = 1, \dots, 3$  at  $a, a, d$ , respectively, yields a spanning tree in the form of a 2-edge star centered at  $v_3 = d$  of length  $2 \times \sqrt{3}/2 = \sqrt{3}$  (the edge lengths in the underlying complete graph are  $\sqrt{3}/2, \sqrt{3}/2$ , and  $0$ ). Another example appears in Fig. 1 (right).

We start by providing a factor  $1/2$  approximation to MAX-ST-N. We then offer two refinement steps achieving a better ratio. The last refinement step proves Theorem 1.

**Theorem 1.** *Given a set  $\mathcal{N}$  of  $n$  neighborhoods in  $\mathbb{R}^d$  (with total vertex complexity  $N$ ), a ratio 0.511 approximation for the maximum spanning tree for the regions in  $\mathcal{N}$  can be computed in polynomial time.*

Although our improvement in the approximation ratio for spanning trees is very small, it shows that the “barrier” of  $1/2$  can be broken. On the other hand, we show that every algorithm that always includes a bichromatic diameter pair in the solution (as the vertices of the corresponding regions) is bound to have an approximation ratio at most  $\sqrt{2 - \sqrt{3}} = 0.517\dots$  (via Fig. 4 in Sect. 3).

*Definitions and notations.* A *geometric graph*  $G$  is a graph whose vertices (a finite set) are points in  $\mathbb{R}^d$  and whose edges consist of straight line segments. For two points  $p, q \in \mathbb{R}^d$ , the Euclidean distance between them is denoted by  $|pq|$ . The *length* of  $G$ , denoted  $\text{len}(G)$ , is the sum of the Euclidean lengths of all edges in  $G$ .

For a neighborhood  $X \in \mathcal{N}$ , let  $V(X)$  denote its set of vertices. Let  $V = \cup_{X \in \mathcal{N}} V(X)$  denote the union of vertices of all neighborhoods in  $\mathcal{N}$ ; put  $N = |V|$ .

Given a set  $\mathcal{N}$  of  $n$  neighborhoods, we define the following parameters. A *monochromatic diameter* pair is a pair of points in the same region attaining a maximum distance. A *bichromatic diameter* pair is a pair of points from two regions attaining a maximum distance, i.e.,  $p_i \in X_i, p_j \in X_j$ , where  $X_i, X_j \in \mathcal{N}$ ,  $i \neq j$ , and  $|p_i p_j|$  is maximum. For  $X \in \mathcal{N}$  and  $p \in X$ , let  $d_{\max}(p)$  denote the maximum distance between  $p$  and any point of a neighborhood  $Y \in \mathcal{N} \setminus \{X\}$ . It is well known and easy to prove that both a monochromatic diameter and bichromatic diameter pair are attained by pairs of vertices in the input instance. An optimal (longest) Spanning Tree with neighborhoods is denoted by  $T_{\text{OPT}}$ ; it is a geometric graph whose vertices are the representative points of the  $n$  regions.

*Preliminaries and related work.* Computing the minimum or maximum Euclidean spanning trees of a point set are classical problems in a geometric setting [13, 14]. A broad collection of problems in geometric network design, including the classical *Euclidean Traveling Salesman Problem* (ETSP), can be found in the surveys [9, 11, 12]. While past research has primarily focused on minimization problems, the maximization variants usually require different techniques and so they are interesting in their own right and pose many unmet challenges; e.g., see the section devoted to longest subgraph problems in the survey of Bern and Eppstein [5]. The results obtained in this area in the last 20 years are rather sparse; the few articles [4, 8, 10] make a representative sample.

Spanning trees for systems of neighborhoods have also been studied. For instance, given a set of  $n$  (possibly disconnected) compact neighborhoods in  $\mathbb{R}^d$ , select a point in each neighborhood so that the minimum spanning tree on these points has minimum length [7, 18], or maximum length [7], respectively. In the cycle version first studied by Arkin and Hassin [3], called *TSP with neighborhoods* (TSPN), given a set of neighborhoods in  $\mathbb{R}^d$ , one must find a shortest closed curve (tour) intersecting each region.

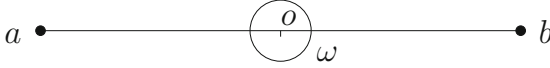
## 2 Approximation Algorithms

For simplicity, we present our algorithms for the plane, namely  $d = 2$ ; the extension to higher dimensions is straightforward, and is briefly discussed at the end.

Let  $S = \{p_1, \dots, p_n\}$ , where  $p_i = (x_i, y_i)$ . Given a point  $p \in S$ , the *star centered at  $p$* , denoted  $S_p$ , is the spanning tree on  $S$  whose edges connect  $p$  to the other points. Using a technique developed in [8] (in fact a simplification of an earlier approach used in [2]), we first obtain a simple approximation algorithm with ratio  $1/2$ .

*Algorithm A1.* Compute a bichromatic diameter of the point set  $V$ , pick an arbitrary point (vertex) from each of the other  $n - 2$  neighborhoods, and output the longest of the two stars centered at one of the endpoints of the diameter.

*Analysis.* Let  $ab$  be a bichromatic diameter pair, and assume without loss of generality that  $ab$  is a horizontal unit segment, where  $a = (0, 0)$  and  $b = (1, 0)$ .



**Fig. 2.** A bichromatic diameter pair  $a, b$  and the disk  $\omega$ .

We may assume that  $a \in X_1$  and  $b \in X_2$ ; refer to Fig. 2. The ratio  $1/2$  (or  $\frac{n}{2n-2}$  which is slightly better) follows from the next lemma in conjunction with the obvious upper bound

$$\text{len}(T_{\text{OPT}}) \leq n - 1. \quad (1)$$

The latter is implied by the fact that each edge of  $T_{\text{OPT}}$  is bichromatic and thus of length at most 1.

**Lemma 1.** *Let  $S_a$  and  $S_b$  be the stars centered at the points  $a$  and  $b$ , respectively. Then  $\text{len}(S_a) + \text{len}(S_b) \geq n$ .*

*Proof.* Assume that  $a = p_1, b = p_2$ . For each  $i = 3, \dots, n$ , the triangle inequality for the triple  $a, b, p_i$  gives

$$|ap_i| + |bp_i| \geq |ab| = 1.$$

By summing up we have

$$\text{len}(S_a) + \text{len}(S_b) = \sum_{i=3}^n (|ap_i| + |bp_i|) + 2|ab| \geq (n - 2) + 2 = n,$$

as required.  $\square$

We next refine this algorithm to achieve an approximation ratio of 0.511. The technique uses two parameters  $x$  and  $y$ , introduced below. The smallest value of the ratio obtained over the entire range of admissible  $x$  and  $y$  is determined and output as the approximation ratio of Algorithm **A2**.

Let  $o$  be the midpoint of  $ab$ , and  $\omega$  be the disk centered at  $o$ , of minimum radius, say,  $x$ , containing at least  $\lfloor n/2 \rfloor$  of the neighborhoods  $X_3, \dots, X_n$ ; in particular, this implies that we can consider  $\lfloor n/2 \rfloor$  neighborhoods as contained in  $\omega$  and  $\lceil n/2 \rceil$  neighborhoods having points on the boundary  $\partial\omega$  or in the exterior of  $\omega$ . We may assume that  $x \leq 0.2$ ; if  $x \geq 0.2$ , the 0.511 approximation ratio easily follows (with room to spare): Since for each of the regions not contained in  $\omega$ , one of the connections from an arbitrary point of the region to  $a$  or  $b$  is at least  $\sqrt{\frac{1}{4} + x^2}$ . If  $T$  is the spanning tree consisting of all such longer connections together with  $ab$ , then

$$\begin{aligned} \text{len}(T) &\geq 1 + \frac{1}{2} \left\lfloor \frac{n}{2} \right\rfloor + \left( \left\lceil \frac{n}{2} \right\rceil - 2 \right) \sqrt{\frac{1}{4} + x^2} \\ &\geq \frac{1 + \sqrt{1 + 4x^2}}{4} (n - 1) + 1 - \frac{3\sqrt{1 + 4x^2}}{4} \\ &\geq \frac{5 + \sqrt{29}}{20} (n - 1) + 1 - \frac{3\sqrt{29}}{20} \geq \frac{5 + \sqrt{29}}{20} (n - 1). \end{aligned}$$

So the approximation ratio is at least  $(5 + \sqrt{29})/20 = 0.519\dots$

Let the monochromatic diameter of  $V$  be  $1 + y$ , for some  $y \in [-1, \infty)$ ; the next lemma shows that  $y \leq 1$ , and so the monochromatic diameter of  $V$  is  $1 + y$ , for some  $y \in [-1, 1]$ .

**Lemma 2.** *For every  $X \in \mathcal{N}$ ,  $\text{diam}(X) \leq 2$ .*

*Proof.* Let  $pq$  be a diameter pair of  $X$ . Let  $r$  be an arbitrary point of an arbitrary neighborhood  $Y \in \mathcal{N} \setminus \{X\}$ . By the triangle inequality, we have  $|pq| \leq |pr| + |rq| \leq 1 + 1 = 2$ , as required.  $\square$

If  $y \geq 0.2$ , let  $a_1, b_1 \in X_1$  be a corresponding diameter pair; choose a point in every other region and connect it to  $a_1$  and  $b_1$ . Since  $|a_1b_1| = 1 + y \geq 1.2$ , the longer of the two stars centered at  $a_1$  and  $b_1$  has length at least  $(n-1)(1+y)/2 \geq 0.6(n-1)$ ; this candidate spanning tree offers thereby this ratio of approximation. We will subsequently assume that  $y \in [-1, 0.2]$ .

As shown above a constant approximation ratio better than  $1/2$  can be obtained if  $x$  or  $y$  is sufficiently large. In the complementary case (both  $x$  and  $y$  are small), an upper bound of  $cn$ , for some constant  $c < 1$ , on the length of  $T_{\text{OPT}}$  can be derived. We continue with the technical details.

*Algorithm A2.* The algorithm computes one or two candidate solutions. The first candidate solution  $T_1$  for the spanning tree is only relevant for the range  $y \geq 0$  (if  $y < 0$  its length could be smaller than  $n/2$ ). Assume that one of the regions, say,  $X_1$  achieves a diameter pair:  $a_1, b_1 \in X_1$ ; recall that  $|a_1b_1| = 1 + y$ . Choose an arbitrary point in every other region and connect it to  $a_1$  and  $b_1$ . Let  $T_1$  be the longer of the two stars centered at  $a_1$  and  $b_1$ . By the triangle inequality,

$$\text{len}(T_1) \geq (n-1) \frac{1+y}{2}. \quad (2)$$

The second candidate solution  $T_2$  for the spanning tree connects each of the regions contained in  $\omega$  with either  $a$  or  $b$  at a cost of at least  $1/2$  (based on the fact that  $\max\{|ap_i|, |bp_i|\} \geq |ab|/2 = 1/2$ ). For each region  $X_i$ ,  $i \geq 3$ , select the vertex of  $X_i$  that is farthest from  $o$  and connect it with  $a$  or  $b$ , whichever yields the longer connection. As such, if  $X_i$  is not contained in  $\omega$ , the connection length is at least  $\sqrt{\frac{1}{4} + x^2}$ . Finally add the unit segment  $ab$ . Then,

$$\text{len}(T_2) \geq 1 + \left\lfloor \frac{n}{2} \right\rfloor \frac{1}{2} + \left( \left\lceil \frac{n}{2} \right\rceil - 2 \right) \sqrt{\frac{1}{4} + x^2}. \quad (3)$$

The above expression can be simplified as follows. If  $n$  is even, (3) yields

$$\begin{aligned} \text{len}(T_2) &\geq 1 + \frac{n}{4} + \left( \frac{n}{4} - 1 \right) \sqrt{1 + 4x^2} \\ &= \frac{n-1}{4} \left( 1 + \sqrt{1 + 4x^2} \right) + \left( \frac{5}{4} - \frac{3}{4} \sqrt{1 + 4x^2} \right) \\ &\geq \frac{n-1}{4} \left( 1 + \sqrt{1 + 4x^2} \right). \end{aligned}$$



If  $n$  is odd, (3) yields

$$\begin{aligned} \text{len}(T_2) &\geq 1 + \frac{n-1}{4} + \left(\frac{n+1}{4} - 1\right) \sqrt{1+4x^2} \\ &= \frac{n-1}{4} \left(1 + \sqrt{1+4x^2}\right) + \left(1 - \frac{2}{4}\sqrt{1+4x^2}\right) \\ &\geq \frac{n-1}{4} \left(1 + \sqrt{1+4x^2}\right). \end{aligned}$$

Consequently, for every  $n$  we have

$$\text{len}(T_2) \geq \frac{n-1}{4} \left(1 + \sqrt{1+4x^2}\right). \quad (4)$$

*Upper bound on  $\text{len}(T_{OPT})$ .* Let  $\Omega$  be the disk of radius  $R(y)$  centered at  $o$ , where

$$R(y) = \begin{cases} \frac{\sqrt{3}}{2} & \text{if } y \leq 0 \\ \frac{\sqrt{3}}{2} + \frac{2}{\sqrt{3}}y & \text{if } y \geq 0 \end{cases}$$

**Lemma 3.**  *$V$  is contained in  $\Omega$ .*

*Proof.* Assume for contradiction that there exists a point  $p_i \in X_i$  at distance larger than  $R(y)$  from  $o$ . By symmetry, we may assume that  $|ap_i| \leq |bp_i|$  and that  $p_i$  lies in the closed halfplane above the line containing  $ab$ .

First consider the case  $y \leq 0$ ; it follows that  $|bp_i| > \sqrt{\frac{1}{4} + \frac{3}{4}} = 1$ . If  $i = 2$ , then  $b, p_i \in X_2$ , which contradicts the definition of  $y$ ; otherwise  $b \in X_2$  and  $p_i \in X_i$  are points in different neighborhoods at distance larger than 1, in contradiction with the original assumption on the bichromatic diameter of  $V$ .

Next consider the case  $y \geq 0$ ; it follows that  $|bp_i| \geq \sqrt{\frac{1}{4} + \left(\frac{\sqrt{3}}{2} + \frac{2}{\sqrt{3}}y\right)^2} > 1 + y$ . If  $i = 2$ , then  $b, p_i \in X_2$ , which contradicts the definition of  $y$ ; otherwise  $b \in X_2$  and  $p_i \in X_i$  are points in different neighborhoods at distance larger than 1, in contradiction with the original assumption on the bichromatic diameter of  $V$ .

In either case (for any  $y$ ) we have reached a contradiction, and this concludes the proof.  $\square$

Recall that for a point  $p \in X \in \mathcal{N}$ ,  $d_{\max}(p)$  is the maximum distance between  $p$  and any point of a neighborhood  $Y \in \mathcal{N} \setminus \{X\}$ .

**Lemma 4.** *Let  $\mathcal{N} = \{X_1, \dots, X_n\}$  be a set of  $n$  neighborhoods and  $T_{OPT}$  be an optimal spanning tree assumed to connect points (vertices)  $p_i \in X_i$  for  $i = 1, \dots, n$ . For every  $j \in [n]$ , we have*

$$\text{len}(T_{OPT}) \leq \sum_{i \neq j} d_{\max}(p_i).$$

*Proof.* Consider  $T_{\text{OPT}}$  rooted at  $p_j$ . Let  $\pi(v)$  denote the parent of a (non-root) vertex  $v$ . Uniquely assign each edge  $\pi(v)v$  of  $T_{\text{OPT}}$  to vertex  $v$ . The inequality  $\text{len}(\pi(v)v) \leq d_{\max}(v)$  holds for each edge of the tree. By adding up the above inequalities, the lemma follows.  $\square$

**Lemma 5.** *If  $X \in \mathcal{N}$  is contained in  $\omega$ , and  $p \in X$ , then  $d_{\max}(p) \leq \min(1, x + R(y))$ .*

*Proof.* By definition,  $d_{\max}(p) \leq 1$ . By Lemma 3, the vertex set  $V$  is contained in  $\Omega$ ; equivalently, all neighborhoods in  $\mathcal{N}$  are contained in  $\Omega$ . By the triangle inequality,  $d_{\max}(p) \leq |po| + R(y) \leq x + R(y)$ , as claimed.  $\square$

**Lemma 6.** *The following holds:*

$$\text{len}(T_{\text{OPT}}) \leq (n-1) \cdot \min\left(1, \frac{1+x+R(y)}{2}\right). \quad (5)$$

*Proof.* Let  $T_{\text{OPT}}$  be a longest spanning tree of  $p_1, \dots, p_n$ , where  $p_i \in X_i$ , for  $i = 1, \dots, n$ . View  $T_{\text{OPT}}$  as rooted at  $p_1 \in X_1$ ; recall that  $a \in X_1$ . By Lemma 4,

$$\text{len}(T_{\text{OPT}}) \leq \sum_{i=2}^n d_{\max}(p_i).$$

If  $X_i$  is not contained in  $\omega$ ,  $d_{\max}(p_i) \leq 1$ ; otherwise, by Lemma 5,  $d_{\max}(p_i) \leq \min(1, x + R(y))$ . By the setting of  $x$  in the definition of  $\omega$ , we have

$$\text{len}(T_{\text{OPT}}) \leq \left(\left\lceil \frac{n}{2} \right\rceil - 1\right) \cdot 1 + \left\lfloor \frac{n}{2} \right\rfloor \cdot \min(1, x + R(y)).$$

If  $n$  is even, the above inequality yields

$$\begin{aligned} \text{len}(T_{\text{OPT}}) &\leq \left(\frac{n}{2} - 1\right) + \frac{n}{2} \min(1, x + R(y)) \\ &= \frac{n-1}{2} (1 + x + R(y)) + \frac{\min(1, x + R(y)) - 1}{2} \\ &\leq \frac{n-1}{2} (1 + x + R(y)), \end{aligned}$$

while if  $n$  is odd, it yields

$$\text{len}(T_{\text{OPT}}) \leq \frac{n-1}{2} + \frac{n-1}{2} (x + R(y)) = \frac{n-1}{2} (1 + x + R(y)).$$

Therefore  $\text{len}(T_{\text{OPT}}) \leq \frac{n-1}{2} (1 + x + R(y))$  in both cases. Then the lemma follows by adjoining the trivial upper bound in Eq. (1).  $\square$

### 3 Analysis of Algorithm A2

We start with a preliminary argument for ratio 0.506 that comes with a simpler proof. We then give a sharper analysis for ratio 0.511.

A first bound on the approximation ratio of **A2**. First consider the case  $y < 0$ . Then  $R(y) = \sqrt{3}/2$ , so the ratio of **A2** is at least

$$\min_{\substack{0 \leq x \leq 0.2 \\ y < 0}} \frac{\text{len}(T_2)}{\text{len}(T_{\text{OPT}})} \geq \min_{0 \leq x \leq 0.2} \frac{1 + \sqrt{1 + 4x^2}}{\min(4, 2 + \sqrt{3} + 2x)}.$$

A standard analysis shows that this ratio achieves its minimum  $(1 + 2\sqrt{2 - \sqrt{3}})/4 = 0.508\dots$  when  $x = 1 - \sqrt{3}/2$ .

When  $y \geq 0$ , the ratio of **A2** is at least

$$\min_{0 \leq x, y \leq 0.2} \max\left(\frac{\text{len}(T_1)}{\text{len}(T_{\text{OPT}})}, \frac{\text{len}(T_2)}{\text{len}(T_{\text{OPT}})}\right).$$

The inequalities (2), (4), (5) imply that this ratio is at least

$$\frac{\max(1 + y, (1 + \sqrt{1 + 4x^2})/2)}{\min(2, 1 + x + R(y))} = \frac{\max(1 + y, (1 + \sqrt{1 + 4x^2})/2)}{\min\left(2, 1 + \frac{\sqrt{3}}{2} + x + \frac{2}{\sqrt{3}}y\right)}.$$

Since the analysis is similar to that for deriving the refined bound we give next, we state without providing details that this piecewise function reaches its minimum value

$$\left(4\sqrt{3} - 1 - 2\sqrt{9 - 3\sqrt{3}}\right)/4 = 0.506\dots$$

when

$$y = \left(4\sqrt{3} - 3 - 2\sqrt{9 - 3\sqrt{3}}\right)/2 = 0.0137\dots$$

and

$$x = \sqrt{3}/2 - 3 + 2\sqrt{3 - \sqrt{3}} = 0.1180\dots$$

This provides a preliminary ratio 0.506 in Theorem 1.

*A refined bound.* Let  $m = \lfloor n/2 \rfloor$ . Assume for convenience that the regions  $X_3, \dots, X_n$  are relabeled so that  $X_3, \dots, X_{m+2}$  are contained in  $\omega$  and  $X_{m+3}, \dots, X_n$  are not contained in the interior of  $\omega$ . Recall that  $p_i \in X_i$  are the representative points in an optimal solution  $T_{\text{OPT}}$ . Let  $x_i = |op_i|$ , for  $i = 3, \dots, m+2$ ; as such,  $x_3, \dots, x_{m+2} \leq x$ . Let the average of  $x_3, \dots, x_{m+2}$  be  $\lambda x$ , where  $\lambda \in [0, 1]$ , i.e.,  $\sum_{i=3}^{m+2} x_i = m\lambda x$ .

Observe that  $d_{\max}(p_i) \leq |op_i| + R(y) = x_i + R(y)$ , for  $i = 3, \dots, m+2$ . Consequently, the upper bound in (5) can be improved to

$$\text{len}(T_{\text{OPT}}) \leq \frac{n-1}{2} (1 + \lambda x + R(y)). \quad (6)$$

We next obtain an improved lower bound on  $\text{len}(T_2)$ . Recall that Algorithm **A2** selects the vertex of  $X_i$  that is farthest from  $o$  for every  $i \geq 3$ , and

connects it with  $a$  or  $b$ , whichever yields the longer connection. In particular, the length of this connection is at least  $\sqrt{\frac{1}{4} + x_i^2}$  for  $i = 3, \dots, m+2$ . Since the function  $\sqrt{x}$  is concave, Jensen's inequality yields:

$$\sum_{i=3}^{m+2} \sqrt{1 + 4x_i^2} \geq m \sqrt{1 + 4\lambda^2 x^2},$$

hence we obtain the following sharpening of the lower bound in (4):

$$\text{len}(T_2) \geq \frac{n-1}{4} \left( \sqrt{1 + 4\lambda^2 x^2} + \sqrt{1 + 4x^2} \right). \quad (7)$$

In order to handle (6) and (7) we make a key substitution  $z = \lambda x$  and simplify the lower bound in (7). Recall that  $0 \leq \lambda \leq 1$ , and so  $0 \leq z \leq x$  and  $z \in [0, 0.2]$ . We now deduce from (6) and (7) that

$$\text{len}(T_{\text{OPT}}) \leq \frac{n-1}{2} (1 + z + R(y)), \quad (8)$$

and

$$\text{len}(T_2) \geq \frac{n-1}{2} \sqrt{1 + 4z^2}. \quad (9)$$

To analyze the approximation ratio we distinguish two cases:

*Case 1:*  $y \leq 0$ . Then  $R(y) = \sqrt{3}/2$ , so the ratio of **A2** is at least

$$\min_{0 \leq z \leq 0.2} \max \left( \frac{\text{len}(T_2)}{\text{len}(T_{\text{OPT}})} \right) \geq \min_{0 \leq z \leq 0.2} \frac{2\sqrt{1 + 4z^2}}{\min(4, 2 + 2z + \sqrt{3})}.$$

When  $4 \leq 2 + 2z + \sqrt{3}$ , we have  $z \geq 1 - \sqrt{3}/2$ . Then

$$\frac{\sqrt{1 + 4z^2}}{2} \geq \frac{\sqrt{8 - 4\sqrt{3}}}{2} = \sqrt{2 - \sqrt{3}} = 0.517\dots$$

When  $4 \geq 2 + 2z + \sqrt{3}$ , or  $z \leq 1 - \sqrt{3}/2$ , let

$$f(z) = \frac{2\sqrt{1 + 4z^2}}{2 + \sqrt{3} + 2z}.$$

Then

$$f'(z) = \frac{8(2 + \sqrt{3})z - 4}{\sqrt{1 + 4z^2} (2 + \sqrt{3} + 2z)^2}.$$

Since  $8(2 + \sqrt{3})z - 4 \leq 4(2 + \sqrt{3})(2 - \sqrt{3}) - 4 = 0$ , the function is non-increasing on  $[0, 1 - \sqrt{3}/2]$  and so

$$f(z) \geq f\left(1 - \sqrt{3}/2\right) = \sqrt{2 - \sqrt{3}} = 0.517\dots$$

This concludes the proof for the first case.

Case 2:  $y \geq 0$ . Then the ratio of **A2** is at least

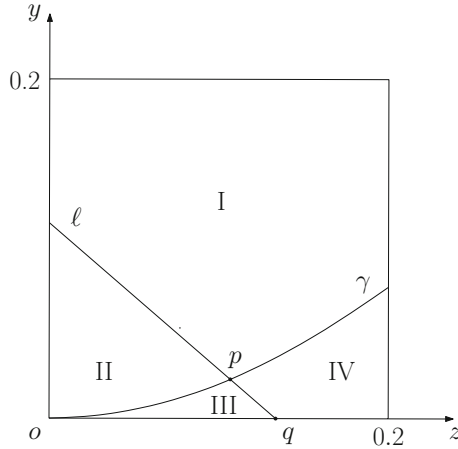
$$0 \leq y, z \leq 0.2 \quad \max \left( \frac{\text{len}(T_1)}{\text{len}(T_{\text{OPT}})}, \frac{\text{len}(T_2)}{\text{len}(T_{\text{OPT}})} \right).$$

For  $0 \leq y, z \leq 0.2$ , let

$$g(z, y) = \frac{\max(1 + y, \sqrt{1 + 4z^2})}{\min(2, 1 + z + R(y))} = \frac{\max(1 + y, \sqrt{1 + 4z^2})}{\min\left(2, 1 + \frac{\sqrt{3}}{2} + z + \frac{2}{\sqrt{3}}y\right)}.$$

The inequalities (2), (8), (9) imply that the ratio of **A2** is at least

$$\min_{0 \leq y, z \leq 0.2} g(z, y).$$



**Fig. 3.** The feasible region of the function  $g(z, y)$ .

The curve  $\gamma : 1 + y = \sqrt{1 + 4z^2}$  and the line  $\ell : 2 = 1 + \frac{\sqrt{3}}{2} + z + \frac{2}{\sqrt{3}}y$  split the feasible region  $[0, 0.2] \times [0, 0.2]$  into four subregions; see Fig. 3. The curve  $\gamma$  intersects line  $\ell$  at point  $p = (z_0, y_0)$ , where  $z_0 = (8\sqrt[4]{3} - \sqrt{3} - 6) / 26 = 0.1075\dots$  and  $y_0 = (8\sqrt{3} - 2\sqrt[4]{27} - 9) / 13 = 0.0228\dots$  Set

$$\rho := (1 + y_0)/2 = \left(4\sqrt{3} + 2 - \sqrt[4]{27}\right) / 13 = 0.5114\dots \tag{10}$$

In region I,  $g(z, y) = (1 + y)/2$ . It reaches the minimum value  $\rho$  when  $y$  is minimized, i.e.,  $y = y_0$ .

In region II,  $g(z, y) = \frac{1 + y}{1 + \sqrt{3}/2 + z + 2y/\sqrt{3}}$ . Its partial derivative is positive, i.e.,

$$\frac{\partial g}{\partial y} = \frac{1 - \sqrt{3}/6 + z}{(1 + \sqrt{3}/2 + z + 2y/\sqrt{3})^2} > 0,$$

so  $g(z, y)$  reaches its minimum value on the curve  $\gamma$ . On this curve, let

$$G(z) = g(z, y(z)) = \frac{\sqrt{1 + 4z^2}}{1 - \sqrt{3}/6 + z + 2\sqrt{1 + 4z^2}/\sqrt{3}}.$$

Its derivative is

$$G'(z) = \frac{(4 - 2\sqrt{3}/3)z - 1}{\sqrt{1 + 4z^2} (1 - \sqrt{3}/6 + z + 2\sqrt{1 + 4z^2}/\sqrt{3})^2}.$$

Note that the numerator of  $G'(z)$  is negative, i.e.,  $(4 - 2\sqrt{3}/3)z - 1 < 4z - 1 < 0$  for  $z \in [0, 0.2]$ , thus  $G'(z) < 0$ . So the minimum value is  $\rho$ , and is achieved when  $z$  is maximized, i.e.,  $z = z_0$ .

In region IV,  $g(z, y) = \sqrt{1 + 4z^2}/2$  which increases monotonically with respect to  $z$ . So the minimum value is again  $\rho$  and is achieved when  $z$  is minimized, i.e.,  $z = z_0$ .

In region III,

$$g(z, y) = \frac{\sqrt{1 + 4z^2}}{1 + \sqrt{3}/2 + z + 2y/\sqrt{3}}.$$

Its partial derivative is negative, i.e.,

$$\frac{\partial g}{\partial y} = \frac{-2\sqrt{1 + 4z^2}}{\sqrt{3} (1 + \sqrt{3}/2 + z + 2y/\sqrt{3})^2} < 0,$$

so  $g(z, y)$  reaches its minimum value on the arc  $op \subset \gamma$  or the segment  $pq \subset \ell$ , where  $q = (1 - \sqrt{3}/2, 0)$  is the intersection point of  $\ell$  and the  $z$ -axis. Since these two curves are shared with region II and IV respectively, by previous analyses,  $g(z, y)$  reaches its minimum value  $\rho$  at point  $p$ .

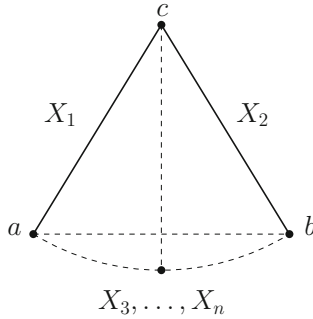
In summary, we showed that

$$\min_{0 \leq y, z \leq 0.2} g(z, y) \geq \rho = 0.511\dots,$$

establishing the approximation ratio in Theorem 1.

*Remark.* The algorithm can be adapted to work in  $\mathbb{R}^d$  for any  $d \geq 3$ . In the analysis, the disk  $\omega$  becomes the ball of radius  $x$  with the same defining property; the disk  $\Omega$  becomes the ball of radius  $R(y)$ . All arguments and relevant bounds still hold since they only rely on the triangle inequality; the verification is left to the reader. Consequently, the approximation guarantee remains the same.

*An almost tight example.* Let  $\Delta abc$  be an isosceles triangle with  $|ca| = |cb| = 1 - \varepsilon$ ,  $|ab| = 1$ , for a small  $\varepsilon > 0$ ; e.g., set  $\varepsilon = 1/(n-1)$ . Let  $\mathcal{N} = \{X_1, \dots, X_n\}$ , where  $X_1 = ac$ ,  $X_2 = bc$ , and  $X_3, \dots, X_n$  are  $n-2$  points at distance  $1 - \varepsilon$  from  $c$ , below  $ab$  and whose projections onto  $ab$  are close to the midpoint of  $ab$  (see Fig. 4). The spanning tree constructed by **A2** is of length close to  $\sqrt{2 - \sqrt{3}}n = 0.517\dots n$ , while the longest spanning tree has length at least  $(1 - \varepsilon)(n - 1) = n - 2$ ; as such, the approximation ratio of **A2** approaches  $\sqrt{2 - \sqrt{3}} = 0.517\dots$  for large  $n$ . Note that this is a tight example for the case  $y \leq 0$ , for which the ratio of **A2** is at least  $\sqrt{2 - \sqrt{3}}$ ; and an almost tight example in general, since the overall approximation ratio of **A2** is 0.511. Moreover, the example shows that every algorithm that always includes a bichromatic diameter pair in the solution (as the vertices of the corresponding regions) is bound to have an approximation ratio at most  $\sqrt{2 - \sqrt{3}}$ .



**Fig. 4.** A tight example.

*Time complexity of Algorithm A2.* It is straightforward to implement the algorithm to run in quadratic time for any fixed  $d$ . All interpoint distances can be easily computed in  $O(N^2)$  time. Similarly the farthest point from  $o$  in each region (over all regions) can all be computed in  $O(N)$  time. Subquadratic algorithms for computing the diameter and farthest bichromatic pairs in higher dimensions can be found in [1, 6, 15–17]; see also the two survey articles [9, 11].

## 4 Conclusion

We gave two approximation algorithms for MAX-ST-N: a very simple one with ratio  $1/2$  and another simple one (with slightly more elaborate analysis but equally simple principles) with ratio 0.511. The following variants represent extensions of the Euclidean maximum TSP for the neighborhood setting.

In the *Euclidean Maximum Traveling Salesman Problem*, given a set of points in the Euclidean space  $\mathbb{R}^d$ ,  $d \geq 2$ , one seeks a cycle (a.k.a. *tour*) that visits these

points (as vertices) and has maximum length; see [4]. In the *Maximum Traveling Salesman Problem with Neighborhoods* (MAX-TSP-N), each point is replaced by a point-set, called *region* or *neighborhood*, and the cycle must connect  $n$  representative points, one chosen from each region (duplicate representatives are allowed), and the cycle has maximum length. Since the original variant with points is NP-hard when  $d \geq 3$  (as shown in [4]), the variant with neighborhoods is also NP-hard for  $d \geq 3$ . The complexity of the original problem in the plane is unsettled, although the problem is believed to be NP-hard [10]. In the *path* variant, one seeks a path of maximum length.

The following problems are proposed for future study:

1. What is the computational complexity of MAX-ST-N?
2. What approximations can be obtained for the *cycle* or *path* variants of MAX-TSP-N?

## References



1. Agarwal, P.K., Matoušek, J., Suri, S.: Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom.* **1**, 189–201 (1992)
2. Alon, N., Rajagopalan, S., Suri, S.: Long non-crossing configurations in the plane. *Fundam. Inf.* **22**, 385–394 (1995)
3. Arkin, E.M., Hassin, R.: Approximation algorithms for the geometric covering salesman problem. *Discret. Appl. Math.* **55**, 197–218 (1994)
4. Barvinok, A.I., Fekete, S., Johnson, D.S., Tamir, A., Woeginger, G.J., Woodroffe, R.: The geometric maximum traveling salesman problem. *J. ACM* **50**(5), 641–664 (2003)
5. Bern, M., Eppstein, D.: Approximation algorithms for geometric problems. In: Hochbaum, D.S. (ed.) *Approximation Algorithms for NP-hard Problems*, pp. 296–345. PWS Publishing Company, Boston (1997)
6. Bhattacharya, B.K., Toussaint, G.T.: Efficient algorithms for computing the maximum distance between two finite planar sets. *J. Algorithms* **4**, 121–136 (1983)
7. Dorrigiv, R., Fraser, R., He, M., Kamali, S., Kawamura, A., López-Ortiz, A., Seco, D.: On minimum- and maximum-weight minimum spanning trees with neighborhoods. *Theory Comput. Syst.* **56**(1), 220–250 (2015)
8. Dumitrescu, A., Tóth, C.D.: Long non-crossing configurations in the plane. *Discret. Comput. Geom.* **44**(4), 727–752 (2010)
9. Eppstein, D.: Spanning trees and spanners. In: Sack, J.-R., Urrutia, J. (eds.) *Handbook of Computational Geometry*, pp. 425–461. Elsevier Science, Amsterdam (2000)
10. Fekete, S.: Simplicity and hardness of the maximum traveling salesman problem under geometric distances. In: *Proceedings of 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 337–345. ACM/SIAM (1999)
11. Mitchell, J.S.B.: Geometric shortest paths and network optimization. In: Sack, J.-R., Urrutia, J. (eds.) *Handbook of Computational Geometry*, pp. 633–701. Elsevier Science, Amsterdam (2000)
12. Mitchell, J.S.B.: Shortest paths and networks. In: Goodman, J.E., O’Rourke, J. (eds.) *Handbook of Computational Geometry*, 2nd edn, pp. 607–641. Chapman & Hall/CRC, Boca Raton (2004)



13. Monma, C., Paterson, M., Suri, S., Yao, F.: Computing Euclidean maximum spanning trees. *Algorithmica* **5**, 407–419 (1990)
14. Preparata, F.P., Shamos, M.I.: *Computational Geometry*. Springer, New York (1985). <https://doi.org/10.1007/978-1-4612-1098-6>
15. Ramos, E.A.: An optimal deterministic algorithm for computing the diameter of a three-dimensional point set. *Discret. Comput. Geom.* **26**(2), 233–244 (2001)
16. Robert, J.M.: Maximum distance between two sets of points in  $\mathbb{R}^d$ . *Pattern Recogn. Lett.* **14**, 733–735 (1993)
17. Toussaint, G.T., McAlear, M.A.: A simple  $O(n \log n)$  algorithm for finding the maximum distance between two finite planar sets. *Pattern Recogn. Lett.* **1**, 21–24 (1982)
18. Yang, Y., Lin, M., Xu, J., Xie, Y.: Minimum spanning tree with neighborhoods. In: Kao, M.-Y., Li, X.-Y. (eds.) *AAIM 2007*. LNCS, vol. 4508, pp. 306–316. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-72870-2\\_29](https://doi.org/10.1007/978-3-540-72870-2_29)



# Efficient Algorithms for a Graph Partitioning Problem

S. Vaishali<sup>1</sup>(✉) , M. S. Atulya<sup>1</sup>(✉) , and Nidhi Purohit<sup>2</sup>(✉)

<sup>1</sup> PSG College of Technology, Coimbatore, India

svaishali.psg@gmail.com, atulyasusheel@gmail.com

<sup>2</sup> The Institute of Mathematical Sciences, HBNI, Chennai, India

nidhipurohit95@gmail.com

**Abstract.** Motivated by an expensive computation performed by a computational topology software RIVET [9], Madkour et al. [1] introduced and studied the following graph partitioning problem. Given an edge weighted graph and an integer  $k$ , partition the vertex set of the graph into  $k$  connected components such that the weight of the heaviest component is as small as possible, where the weight of each component is the weight of a minimum spanning tree of the graph induced by the vertices in that component. They showed that the problem is  $NP$ -hard even for  $k = 2$  and provided some heuristic algorithms. They asked whether the problem is polynomial time solvable when the input is a tree. Our first result is an affirmative answer to their question. We give a polynomial time algorithm to provide such a partition in a tree. We also give an exact exponential algorithm taking  $O^*(2^n)$  time on general graphs (improving on the naive  $O^*(k^n)$  algorithm) ( $O^*$  notation ignores polynomial factors). We also prove that the problem remains  $NP$ -complete even when the weights on all the edges are the same and give a linear time algorithm for this version of the problem when the graph is a tree.

## 1 Introduction

We investigate and propose efficient algorithms for a tree partitioning problem that has been introduced recently. A *spanning  $k$ -forest* of an edge-weighted, undirected graph  $G$  is a collection of  $k$  trees,  $T_1, \dots, T_k$ , each a subgraph of  $G$ , such that each vertex of  $G$  is contained in exactly one  $T_i$  for some  $i \in \{1, \dots, k\}$ . A *minimum spanning  $k$ -forest* of  $G$  is a spanning  $k$ -forest such that the quantity  $\max\{w(T_1), \dots, w(T_k)\}$  is minimum among all the spanning  $k$ -forests of  $G$ . Here  $w(T_i)$  is the total weight of the edges in  $T_i$ . If  $F$  is a spanning  $k$ -forest of  $G$ , then the quantity  $w(F) = \max\{w(T_1), \dots, w(T_k)\}$  is the weight of  $F$ .

Madkour et al. [1] considered the following problem.

**Input:** An edge-weighted, undirected graph  $G$  on  $n$  vertices, integer  $k > 1$   
**Output:** Minimum spanning  $k$ -forest of  $G$ .

Work done while the authors were visiting IMSc Chennai.

## 1.1 Related Work

Partitioning a graph into  $k$ -connected components is a problem well studied in literature, having multiple variants. The special case where each connected component is a tree forms a small subset of this problem [2–8]. Madkour et al. [1] introduced and showed the problem  $NP$ -complete on general graphs even for  $k = 2$  and provided two heuristic algorithms for the problem, one based on spectral clustering and the other based on dynamic programming. They left open the complexity of the problem on trees.

A few variants of this problem are studied in [2,7,8]. The complexity of the problem of partitioning a tree into  $k$  balanced parts is studied in [7]. The problem of finding a  $k$ -partition of a graph in which the weight of each component is defined based on the weight of its minimum spanning tree and the sum of weights of its outgoing edges is studied in [8]. The closest variant to our problem that has been previously studied is the problem that asks to find a minimum spanning  $k$ -forest of a vertex weighted graph, here the weight of each component in a spanning  $k$ -forest of a graph is the sum of weights of all the vertices in that component. It is known [2] that this problem on trees can be solved in linear time. In Sect. 2, we reduce our problem on trees having edges of equal weight to this vertex weighted problem to obtain a linear time algorithm. However, for the more general edge weighted version of the problem, this reduction fails, and we provide a greedy based polynomial algorithm in Sect. 3. Our algorithm, not only works for the general edge weighted version, but for the unweighted version, it is, we believe, much simpler than the linear time algorithm obtained via the vertex weighted version. Our algorithm follows a greedy local search strategy.

## 1.2 Paper Organization

Our main result of the paper is a polynomial time algorithm for the problem on trees. As  $k - 1$  edges are necessary and sufficient to remove from the tree to get  $k$  connected components, this immediately gives an  $O(n^k)$  algorithm for the problem by trying all possible  $k - 1$  subsets of the edge set. This already suggests that the problem is quite different on trees, as for general graphs the problem was  $NP$ -complete even for  $k = 2$ . In Sect. 3, we provide an  $O(kn^3)$  algorithm for the problem. Our algorithm follows the popular greedy local search method similar to the one used in [10] for an approximation algorithm for the minimum degree spanning tree problem. Starting from a partition obtained by removing the  $k - 1$  heaviest weight edges, iteratively we find an improved solution through local swaps of edges until no improvement is possible. We show that such a greedy solution stops in  $O(n)$  phases (where each phase takes  $O(kn^2)$  time) and produces the optimum solution.

In Sect. 2, we consider the class of graphs having all edges of equal weight. Here, we first show that the problem remains  $NP$ -complete even on these (unweighted) graphs. We then give a linear time algorithm for this version of the problem on trees by reducing to the vertex weighted version of the problem on trees.

A brute force exponential algorithm in general graphs takes  $O(k^n)$  time by trying all possible partitions of the vertex set into  $k$  parts. We improve this to  $O^*(3^n)$  using a Dynamic Programming approach in Sect. 4 and to  $O^*(2^n)$  using the well known problem of finding a  $k$ -partition of a set system  $(U, S)$ , where  $U$  is a universe of elements and  $S$  is a collection of non-empty subsets of  $U$ . Finally we conclude in Sect. 5 with some open problems.

## 2 Minimum Spanning $k$ -forest of a Graph Having Equal Weighted Edges

In this section we first show that the problem of finding the minimum spanning  $k$ -forest of a graph having all edge weights equal is NP-complete. The decision version of the problem of finding a minimum spanning  $k$ -forest of a graph for  $k = 2$  is: Given a weighted graph  $G$  and a positive number  $W$ , decide whether  $G$  has a minimum spanning 2-forest of weight  $W$ , this was proved to be NP-complete on general graphs in [1]. We extend their proof by modifying the structure of the graph constructed in their reduction to prove the following theorem.

**Theorem 1.** *The problem of deciding whether a graph  $G$  having all edges of equal weight has a minimum spanning 2-forest of weight  $W$  is NP-complete.*

*Proof.* The reduction is from the NP-complete partition problem [12] where we are given an  $m$ -element multiset of total weight  $2W$  and the goal is to determine whether the set can be partitioned into two sets of equal weight.

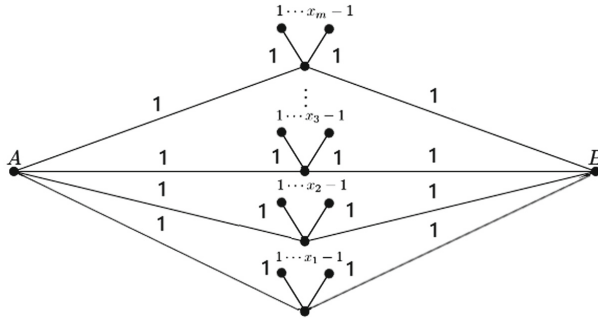


Fig. 1. Graph  $G$  used in the proof of Theorem 1

Let  $M = \{x_1, \dots, x_m\}$  be an  $m$ -element multiset of weight  $2W$ , without loss of generality we assume that no  $x_i \in M$  is of weight greater than  $W$ . We construct a graph  $G$  having all edges of weight one as shown in Fig. 1.  $G$  consists of  $m$  stars where the  $i^{th}$  star  $S_i$  has a centre vertex and  $x_i - 1$  other vertices adjacent to the centre vertex. In addition  $G$  has two other vertices  $A$  and  $B$  adjacent to all centre vertices.

We need to prove that  $M$  can be partitioned into two sets of equal weight  $W$  if and only if  $G$  has a minimum spanning 2-forest  $F$  of weight  $W$ .

For proving the forward direction, let us suppose that there exists a partition of  $M$ , say  $(M_1, M_2)$  such that  $M_1$  and  $M_2$  have equal weight. Consider the partition of  $V(G)$  into  $V_1$  and  $V_2$  where  $V_1 = \{A\} \cup \{V(S_i) \mid x_i \in M_1\}$  and  $V_2 = \{B\} \cup \{V(S_i) \mid x_i \in M_2\}$ . It is clear from our construction of  $G$  that  $G[V_1]$  forms a tree and since the sum of the weights of all the edges incident on the centre vertex of each  $S_i \in G[V_1]$  is  $x_i$  and  $M_1$  has weight  $W$ , the weight of  $G[V_1]$  is  $W$ . Similarly  $G[V_2]$  is a tree of weight  $W$ . Therefore  $F = G[V_1] \cup G[V_2]$  is a minimum spanning 2-forest of  $G$  of weight  $W$ .

Conversely, suppose that  $G$  has a minimum spanning 2-forest  $F$  of weight  $W$ , we need to prove that  $M$  can be partitioned into two sets of equal weight  $W$ . We first note that no star in  $G$  is broken in  $F$ , if some star is broken then  $F$  would contain an isolated vertex that is a tree of weight 0 which contradicts our assumption that  $F$  is a minimum spanning 2-forest of  $G$ . Also we claim that  $A$  and  $B$  occur in two different trees in  $F$ . We prove this by assuming that  $A$  and  $B$  are in the same tree in  $F$ , since no star in  $G$  is broken in  $F$  and since  $F$  contains 2 trees in it, one star  $S_i$  in  $G$  should be one of the two trees in  $F$ . It is easy to see that the weight of  $S_i$  is  $<W$ , as having a weight of  $\geq W$  would imply that  $x_i \geq W + 1$  but we assumed that no such  $x_i$  exists in  $M$ . Since the other tree has all the other stars along with  $A$  and  $B$  in it, the number of nodes in it is equal to  $2W + 2 - x_i$  which is greater than or equal to  $W + 2$ , hence the weight of this tree is  $\geq W + 1$ , which contradicts our assumption that the weight of  $F$  is  $W$ . Therefore  $A$  and  $B$  must belong to different trees in  $F$ . Let  $T_A$  and  $T_B$  be the trees rooted at  $A$  and  $B$  respectively. Each star  $S_i$  in  $G$  must either belong to  $T_A$  or  $T_B$ . Let  $M_1 = \{x_i \mid S_i \in T_A\}$  and  $M_2 = \{x_i \mid S_i \in T_B\}$ . Since,  $x_i - 1$  is the weight of each star  $S_i \in T_A$ , each star along with the edge from its centre vertex to  $A$  contributes a weight of  $x_i$ . As we know that the weight of  $T_A$  is  $W$ , the weight of  $M_1$  is also equal to  $W$ . Using a similar argument, the weight of  $M_2$  is also equal to  $W$ . Therefore  $(M_1, M_2)$  partitions  $M$  into two equal parts.

Therefore the problem of deciding whether a graph  $G$  having all edges of equal weight has a minimum spanning 2-forest of weight  $W$  is NP-hard. Since this problem is clearly also NP, it is NP-complete.  $\square$

In the remainder of this section, we give a  $O(n)$  time algorithm for trees having all edges of equal weight by reducing it to the vertex weighted version of the problem [4]. Firstly we formally define this version of the problem in terms of a spanning forest.

A *spanning  $k$ -forest* of a vertex-weighted, undirected graph  $G$  is a collection of  $k$  trees,  $T_1, \dots, T_k$ , each a subgraph of  $G$ , such that each vertex of  $G$  is contained in exactly one  $T_i$  for some  $i \in \{1, \dots, k\}$ ; A *minimum spanning  $k$ -forest* of  $G$  is a spanning  $k$ -forest such that the quantity  $\max \{w(T_1), \dots, w(T_k)\}$  is minimum among all the spanning  $k$ -forests of  $G$ . Here  $w(T_i)$  is the total weight of the vertices in  $T_i$ . If  $F$  is a spanning  $k$ -forest of  $G$ , then the quantity  $w(F) = \max \{w(T_1), \dots, w(T_k)\}$  is the weight of  $F$ .

The vertex weighted version of the problem of finding a minimum spanning  $k$ -forest is formally defined below:

**Input:** A vertex-weighted, undirected graph  $G$  on  $n$  vertices, integer  $k > 1$   
**Output:** Minimum spanning  $k$ -forest of  $G$ .

**Theorem 2 [2].** *The minimum spanning  $k$ -forest of a vertex weighted tree can be found in  $O(n)$  time.*

The problem of finding a minimum spanning  $k$ -forest in an edge-weighted tree having all edges of weight  $w$  can be reduced to the problem of finding a minimum spanning  $k$ -forest of a tree having all vertices of weight  $w$ . Since the difference between the weight of any subtree in the vertex weighted tree and edge weighted tree is exactly one, an optimum solution for the vertex weighted version is an optimum solution for the edge weighted version. Combining this observation with Theorem 2, we obtain the following result.

**Theorem 3.** *The minimum spanning  $k$ -forest of an edge weighted tree having all edges of equal weight can be found in  $O(n)$  time.*

This type of reduction to the vertex weighted case is infeasible for the case of trees having edges of unequal weights. We observe that there is no easy way to transfer the weight of the edges to the vertices since in a spanning  $k$ -forest we may delete an edge of very large weight but we have to retain such a vertex, thus obtaining no direction relation between the weight of the forests in the two versions. In the subsequent section we propose a polynomial time algorithm for general trees.

### 3 Minimum Spanning $k$ -forest of a Tree

We propose a polynomial time algorithm for computing a minimum spanning  $k$ -forest of an edge-weighted tree  $T$ . The algorithm is as follows:

**Input:** Tree  $T(V, E)$ ,  $1 < k \leq |V|$

**Output:** Minimum spanning  $k$ -forest of  $T$

**Step 1:** Sort the edges of  $T$  according to non increasing order of weights and let  $e_1, \dots, e_{|E|}$  be the edges in this order. Initialize  $C = \{e_1, \dots, e_{k-1}\}$ .

Let  $S$  be the set of edges that have ever been in  $C$  in the course of the algorithm, initialize  $S = C$ . Initialize  $F$  with the spanning  $k$ -forest of  $T$  that results from removing all the edges in  $C$  from  $T$ .

**Step 2:** Construct a  $(k - 1)$  by  $(n - k)$  table  $U$ , such that

$$U[a, d] = w(F \cup a \setminus d), \quad \forall a \in C, d \in E \setminus C$$

That is  $U[a, d]$  is the weight of the maximum component formed by adding  $a$  to  $F$  and deleting  $d$  from  $F$ .

**Step 3:** Let  $\text{minw} = \min\{U[a, d] \mid a \in C, d \in E \setminus C\}$ .

If ( $\text{minw} > w(F)$ ) or ( $\text{minw} = w(F)$ ) but  $\forall U[a, d]$  such that  $U[a, d] = \text{minw}$ ,  $d \in S$ )

Return  $C$  and  $F$ .

Else if ( $\text{minw} = w(F)$ ) and  $\exists U[a, d]$  such that  $U[a, d] = \text{minw}$ ,  $d \notin S$ )

Arbitrarily choose an entry  $U[a, d]$  such that  $U[a, d] = \text{minw}$  and  $d \notin S$ .

Else if ( $\text{minw} < w(F)$ )

Arbitrarily choose an entry  $U[a, d]$  such that  $U[a, d] = \text{minw}$  giving priority to  $U[a, d]$  for which  $d \notin S$  if such a  $d$  exists.

$F = F \cup a \setminus d$ ,  $C = C \cup d \setminus a$  and if  $d \notin S$ ,  $S = S \cup d$  (here  $a$  and  $d$  are the values corresponding to the entry  $U[a, d]$  chosen in this step).

Go to Step 2.

A pseudocode of the algorithm is given in Algorithm 1 (Page 10). In the rest of this section, we argue about the correctness of the proposed algorithm followed by an analysis of its running time. We will be using the notation  $C_i$  to denote the set  $C$  at the beginning of iteration  $i$  of the do while loop in Algorithm 1 (Step 2 in the above description.) in further parts of this section.

**Lemma 1.** *An edge will occur at most twice in  $C$  during the entire execution of Algorithm 1.*

*Proof.* Assume that the claim is not true, and let  $x$  be the first edge that is deleted for the third time from  $F$ . Let  $(j, x)$ ,  $(x, k)$ , and  $(l, x)$  be the (add, delete) pairs chosen in iterations  $t_1, t_2$  and  $t_3$  respectively, where  $t_3$  is the iteration in which  $x$  is deleted from  $F$  for the third time and  $t_1$  and  $t_2$  are the previous two iterations involving  $x$ . Firstly, we prove a claim which will be helpful for the proof.

*Claim.*  $k$  is deleted from  $F$  for the second time in iteration  $t_2$ .

*Proof.* First, we need to prove that  $k$  has been deleted from  $F$  in some iteration before  $t_2$ . If  $k \in C_{t_1}$ , then it is trivially true. Otherwise if  $k \notin C_{t_1}$ , then we prove that  $k$  has been deleted from  $F$  in some iteration before  $t_1$ . Consider the  $U$  table in iteration  $t_1$ . First  $U[j, x] \leq U[j, k]$  is true as  $(j, x)$  was chosen in iteration  $t_1$ . Since  $(x, k)$  was the (add, delete) edge pair chosen in iteration  $t_2$  and  $x$  was not involved in any iteration between  $t_1$  and  $t_2$ , it is equivalent to saying that  $(j, k)$  was chosen in iteration  $t_1$ . This combined with that fact that  $w(F)$  progressively decreases or remains the same during the run of the algorithm, it cannot be the case that  $U[j, x] < U[j, k]$  in iteration  $t_1$ . Therefore it must be the case that  $U[j, x] = U[j, k]$  in iteration  $t_1$ .

As  $x$  was in  $C$  before iteration  $t_1$  and  $U[j, x] = U[j, k]$  in the  $U$  table in iteration  $t_1$ , and since  $(j, k)$  was not chosen in iteration  $t_1$ , we can clearly see that  $k$  must have been in  $C$  before iteration  $t_1$  as per Step 3 of the algorithm and hence must have been in  $C$  before iteration  $t_2$  as well.

We note that  $k$  has been in  $C$  before iteration  $t_2$  and was deleted from  $F$  in iteration  $t_2$ , which implies that it was deleted from  $F$  for the second time in

iteration  $t_2$ . If not, then  $x$  would have not been the first edge to have been deleted from  $F$  for the third time. Hence we have proved that  $k$  is deleted from  $F$  for the second time in iteration  $t_2$ .  $\square$

The  $U$  table mentioned henceforth in the proof is the one in iteration  $t_2$ . Since by assumption  $x$  is the first edge that was deleted for the third time and by the above claim  $k$  is deleted from  $F$  for the second time in  $t_2$ ,  $k$  will not be deleted from  $F$  between iterations  $t_2$  and  $t_3$  if replaced. Therefore, the iterations  $t_2$  and  $t_3$  combined is equivalent to choosing the pair  $(l, k)$  in iteration  $t_2$  if  $l \in C_{t_2}$ , which implies  $U[l, k] < U[x, k]$  since the weight of  $F$  decreases in iteration  $t_3$  as  $x$  has been cut before. If  $l \notin C_{t_2}$ , let the sequence  $\{k', \dots, l\}$ ,  $k' \in C_{t_2}$ , denote the order of addition of cut edges in  $C$  to  $F$  leading to  $l$  after iteration  $t_2$ , where  $k'$  is the first edge in  $C_{t_2}$  in the sequence. In this case  $U[k', k] < U[x, k]$ , since the weight of  $F$  decreases in iteration  $t_3$ . This is a contradiction to our algorithm selecting the (add, delete) edge pair  $(x, k)$  in iteration  $t_2$ , it must have selected  $(l, k)$  if  $l \in C_{t_2}$  or  $(k', k)$  if  $l \notin C_{t_2}$ . Hence we have proved by contradiction that an edge will occur at most twice in  $C$  during the entire execution of the Algorithm.  $\square$

Following corollary is immediate from Lemma 1.

**Corollary 1.** *The do while loop (Steps 2 and 3) in Algorithm 1 is repeated at most  $2n - k$  times.*

**Lemma 2.** *Algorithm 1 returns an optimum solution, a minimum spanning  $k$ -forest of  $T$ .*

*Proof.* Let  $A$  be the spanning  $k$ -forest of  $T$  returned by Algorithm 1 and let  $O$  be a minimum spanning  $k$ -forest of  $T$ . Let  $C(A)$  and  $C(O)$  be the set of edges of  $T$  not in  $A$  and  $O$  respectively, i.e. the set of cut edges.

Let us assume that  $A$  is not a minimum spanning  $k$ -forest of  $T$ , then it follows that  $w(O) < w(A)$ . We also assume that  $\forall e \in C(A) \setminus C(O)$  and  $f \in C(O) \setminus C(A)$ ,  $w(E \setminus C(O) \setminus f \cup e) > w(E \setminus C(O))$ . This is achieved by executing the following step until no longer possible.

If  $\exists e \in C(A) \setminus C(O)$  and  $f \in C(O) \setminus C(A)$  such that  $w(E \setminus C(O) \setminus f \cup e) = w(E \setminus C(O))$ , then change  $C(O)$  as follows,  $C(O) = C(O) \setminus f \cup e$ .

If  $A_{max}$  is a maximum weight tree in  $A$ , then it cannot be a subtree in  $O$  as  $w(O) < w(A)$ . Therefore each  $A_{max}$  has to be broken in  $O$ , and thus at least one edge in each  $A_{max}$  must be in  $C(O) \setminus C(A)$ . Thus we have,

*Claim 1:*  $\forall A_{max} \in A$ , where  $A_{max}$  is a maximum weight tree in  $A$ ,  $\exists x \in A_{max}$  such that  $x \in C(O) \setminus C(A)$ .

We say a tree  $A_l \in A$  is a subtree in  $O$  if  $\forall x \in A_l$ ,  $x \notin C(O)$ . Since the total number of trees in  $A$  is  $k$  and  $|C(O)| = |C(A)| = k - 1$ , there must exist a tree in  $A$  which does not have any edge from  $C(O)$ . Thus we also have,

*Claim 2:*  $\exists$  a tree  $A_m \in A$  that is a subtree in  $O$ . Formally  $\exists A_m \in A$  such that  $\forall x \in A_m$ ,  $x \notin C(O) \setminus C(A)$ .



We define a tree  $A_l \in A$  to be adjacent to a tree  $A_m \in A$ , if  $\exists y \in C(A)$  connecting  $A_l$  and  $A_m$ , that is  $y$  is the cut edge between  $A_l$  and  $A_m$  in  $A$ . For proving our lemma, we now consider the two cases given below; these cases cover all possible configurations of  $A$ . In the proof,  $F$  refers to the spanning  $k$ -forest modified at the beginning of each iteration of the do while loop in Algorithm 1 (Step 2). On termination of the algorithm,  $F$  equals  $A$ . Let  $A_m \in A$  be a subtree in  $O$ , by Claim 2 such a tree exists.

*Case 1:* There exists a tree  $A_l \in A$  adjacent to  $A_m$  such that  $A_l$  is a subtree in  $O$ .

*Case 2:* Every adjacent tree of  $A_m$  is not a subtree in  $O$ .

*Claim.* If  $A$  satisfies Case 1 or Case 2 then it is an optimum solution.

*Proof.* We prove Case 1 and Case 2 separately.

*Case 1:* Let  $A_{max}$  be some maximum weight tree in  $A$ . By Claim 1,  $\exists x \in A_{max}$ , such that  $x \in C(O) \setminus C(A)$ . Choose  $y$  to be the cut edge between  $A_l$  and  $A_m$  in  $A$ .

(i)  $A$  has exactly one maximum weight tree: Consider the  $U$  table at the end of Algorithm 1. Since  $A_{max}$  is the only maximum weight tree and  $A_l \cup y \cup A_m$  is a subtree in  $O$ , we can clearly see that  $U[y, x] < w(A)$ . This contradicts the termination of Algorithm 1.

(ii)  $A$  has more than one maximum weight tree: Consider the  $U$  table in the final iteration, it is easy to see that  $U[y, x] = w(A)$  as there are more than one maximum weight trees in  $A$  and  $U[y, x]$  cannot be greater than  $w(A)$  as  $A_l \cup y \cup A_m$  is a subtree in  $O$ . In this case, since the algorithm has terminated,  $x$  should have been in  $S$  (the set of edges that have ever been in  $C$  during the course of the algorithm) during the last iteration.

By our assumption,  $w(E \setminus C(O) \setminus x \cup y) > w(O)$ ; therefore on removal from  $A$ ,  $x$  contributes to a better decrease in weight of  $A$  than  $y$ . We use this to prove that the algorithm made a wrong choice in some iteration thereby contradicting its working to prove the claim.

Let  $p$  be the iteration in which  $y$  was deleted from  $F$  for the last time, that is  $y$  was not added back to  $F$  in any iteration after  $p$  as  $y \in C(A)$ . If  $x \notin C_p$  it must have been better to delete  $x$  from  $F$  in iteration  $p$ . Let  $k$  be the edge added back to  $F$  in iteration  $p$ , here  $U[k, x] < U[k, y]$  in the  $U$  table in iteration  $p$  since  $x$  being cut contributes to a better decrease in the weight of  $F$  than  $y$  being cut.

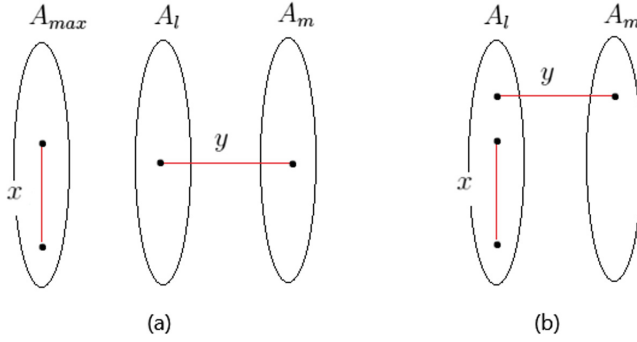
Let  $q$  be the iteration in which  $x$  was added back to  $F$  for the final time, that is  $x$  was not deleted from  $F$  in any iteration after  $q$  as  $x \notin C(A)$ . If  $x \in C_p$ , then iteration  $q$  must have occurred after iteration  $p$ . This implies that  $y \in C_q$  as  $y$  is not added back to  $F$  in any iteration after  $p$ . In this case, it must have been better to add  $y$  to  $F$  in iteration  $q$ . Let  $k'$  be the edge deleted from  $F$  in

iteration  $q$ , here  $U[y, k'] < U[x, k']$  in the  $U$  table in iteration  $q$  since  $x$  being in  $C$  contributes to a better decrease in the weight of  $F$  than  $y$  being in  $C$ .

This is a contradiction to our algorithm selecting the (add, delete) edge pair  $(k, y)$  rather than the optimal pair  $(k, x)$  in iteration  $p$  if  $x \notin C_p$  and the pair  $(x, k')$  rather than the optimal pair  $(y, k')$  in iteration  $q$  if  $x \in C_p$ . Therefore the claim is true.

*Case 2:* Choose the tree  $A_l \in A$  adjacent to  $A_m$  for which the cut edge  $y$  between  $A_l$  and  $A_m$  was first added to  $C$  for the last time (not deleted from  $C$  again). The argument here is similar to (ii) of Case 1 with the chosen  $x$  and  $y$ .  $\square$

Figure 2 depicts edges  $x$  and  $y$  selected in both cases. Hence we have proved that Algorithm 1 produces the optimum solution.  $\square$



**Fig. 2.** Proof of optimality (a) Case 1 (b) Case 2

**Lemma 3.** *Algorithm 1 runs in  $O(kn^3)$  time.*

*Proof.* The time taken to compute the  $k - 1$  largest edges in a tree is  $O(n \log n)$ . Step 2 involves computing the  $U$  table, the table has  $(k - 1)(n - k) = O(kn)$  entries. Computing each entry  $U[a, d]$  takes  $O(n)$  time,  $w(F \cup a \setminus d)$  is computed by performing DFS multiple times until all nodes in  $F$  are visited and by keeping track of the weight of each component while traversing. Hence the time taken for Step 2 is  $O(kn^2)$ . Choosing an edge pair  $(a, d)$  in  $U$  according to the cases in Step 3 takes  $O(n)$  time. Returning the minimum spanning  $k$ -forest in Step 4 takes  $O(1)$  time. The number of times Steps 2 and 3 are executed is at most  $2n - k$  from Corollary 1. Therefore, the total time taken by the algorithm is  $O(n \log n + n(kn^2 + n) + 1) = O(kn^3)$ .  $\square$

Combining Lemmas 2 and 3, we obtain Theorem 4.

**Theorem 4.** *For an undirected weighted tree  $T$  on  $n$  vertices and a positive integer  $k > 1$ , there exists an algorithm running in time  $O(kn^3)$  which computes the minimum spanning  $k$ -forest of  $T$ .*

---

**Algorithm 1.** Computes a Minimum Spanning  $k$ -Forest of  $T$ 


---

```

1: procedure MSKF_TREES( $T(V, E), k$ )
    $T$  - Input tree with  $n$  vertices
    $k$  - Required number of components
    $C$  - Set of  $k - 1$  cut edges
    $S$  - Set of edges that have ever been in  $C$ 
    $F$  - Spanning  $k$ -forest  $E \setminus C$ 
    $w(F)$  - weight of  $F$ 
    $U$  -  $(k - 1) * (n - k)$  table, whose rows correspond to the edges in  $C$  and
   columns correspond to the edges in  $E \setminus C$ .
2:   if  $k > n$  then
3:     return null
4:   end if
5:    $e_1, \dots, e_{|E|} \leftarrow$  Non increasing sorted order of edges in  $E$ .
6:    $C \leftarrow \{e_1, \dots, e_{k-1}\}$ 
7:    $S \leftarrow C$ 
8:   do
9:     for  $a \in C$  do
10:      for  $d \in E(T) \setminus C$  do
11:         $U[a, d] \leftarrow w(F \cup a \setminus d)$ 
12:      end for
13:    end for
14:     $minw \leftarrow$  minimum value in  $U$ 
15:    if ( $minw > w(F)$ ) or
16:    ( $minw = w(F)$  but  $\forall U[a, d]$  such that  $U[a, d] = minw, d \in S$ ) then
17:      return  $C$  and  $F$ 
18:    else if  $minw = w(F)$  and  $\exists U[a, d] = minw$  such that  $d \notin S$  then
19:      Arbitrarily choose any  $U[a, d]$  such that  $U[a, d] = minw$  and  $d \notin S$ .
20:    else if  $minw < w(F)$  then
21:      if  $\exists U[a, d] = minw$  such that  $d \notin S$  then
22:        Arbitrarily choose any  $U[a, d]$  such that  $U[a, d] = minw$  and  $d \notin S$ .
23:      else
24:        Arbitrarily choose any  $U[a, d]$  such that  $U[a, d] = minw$ .
25:      end if
26:    end if
27:     $C \leftarrow C \cup d \setminus a$ 
28:     $F \leftarrow F \cup a \setminus d$ 
29:    if  $d \notin S$  then
30:       $S \leftarrow S \cup d$ 
31:    end if
32:  while ( $true$ )
33: end procedure

```

---

## 4 Exact Exponential Algorithm for Finding a Minimum Spanning $k$ -forest of a Graph

In this section, we first propose a simple  $O^*(3^n)$  for computing the minimum spanning  $k$ -forest of a graph. Following this, we obtain a  $O^*(2^n)$  algorithm for the problem of counting the number of minimum spanning  $k$ -forests of a graph by reducing it to an instance of the well known problem of finding a  $k$ -partition of a set system. We use this algorithm as a black box to compute a minimum spanning  $k$ -forest in  $O^*(2^n)$  time.

A brute force exponential algorithm to compute a minimum spanning  $k$ -forest in general graphs takes  $O^*(k^n)$  time by trying all possible partitions of the vertex set into  $k$  parts. We improve the brute force time to  $O^*(3^n)$  using a Dynamic Programming approach.

Let  $G(V, E)$  be an undirected weighted graph whose total weight is  $W$  and let  $w$  and  $i$  be positive integers.

An optimum  $i$ -partition amongst all the  $i$ -partitions of  $G[S]$ ,  $S \subseteq V$  having the weight of each part in the partition  $\leq w$  is a partition having the least weighing maximum component.

Let  $T(S, i, w)$ ,  $S \subseteq V$  be the weight of a maximum component in an optimum  $i$ -partition of  $G[S]$  having the weight of each part in the partition  $\leq w$ , if such a partition exists, else it is  $\infty$ . Let  $w(G)$  represent the weight of a minimum spanning tree of  $G$ .

We define  $P$ , the set of possible or candidate values of  $w$ , the weight of a minimum spanning  $k$ -forest of  $G$  as follows,

$$P = \{w(G[S]) \mid S \subseteq V, G[S] \text{ is connected}\} \quad (1)$$

The value of  $T(S, i, w)$ ,  $\forall S \subseteq V, 1 \leq i \leq k, w \in P$  can be computed recursively as follows.

$$\begin{aligned} T(S, i, w) &= \min_{i>1} \begin{cases} \max \left\{ w(G[S']), T(S - S', i - 1, w) \right\} & w(G[S']) \leq w \\ \infty & G[S'] \text{ is connected} \\ & \text{otherwise} \end{cases} \\ T(S, 1, w) &= \begin{cases} w(G[S]) & w(G[S]) \leq w \\ \infty & G[S] \text{ is connected} \\ & \text{otherwise} \end{cases} \\ T(\phi, i, w) &= \infty, \end{aligned} \quad (2)$$

**Lemma 4.** *If  $F$  is a minimum spanning  $k$ -forest of  $G(V, E)$  having  $|V| = n$  then  $w(F) = \min_{w \in P} \{T(V, k, w)\}$  and  $w(F)$  can be found in  $O(nt(V, k, w))$  time, where  $t(v, k, w)$  is the time required to compute  $T(V, k, w)$  for a fixed  $w$ .*

*Proof.* If  $O$  is an optimum  $k$ -partition of  $G$  having the weight of each part in the partition  $\leq w$ , then  $O - S'$  must be an optimal  $(k - 1)$ -partition of  $G - S'$  having the weight of each part in it  $\leq w$ , where  $S'$  is one component in  $O$ .

Therefore, we can infer that the problem has the optimal substructure property. The smallest  $w \in P$  for which  $T(V, k, w) \neq \infty$ , gives the weight of a minimum spanning  $k$ -forest of  $G$ . Note that  $T(V, k, w) = T(V, k, w'), \forall w > w', w \in P$ .

Since  $P$  has  $O^*(2^n)$  possible values of  $w$ , we can do a binary search to find the smallest  $w \in P$  for which  $T(V, k, w) \neq \infty$  in time  $O(nt(V, k, w))$  time.  $\square$

**Lemma 5.** *Given an undirected weighted graph  $G(V, E)$  on  $n$  vertices and a positive integer  $1 \leq k \leq n$ ,  $w(F)$  the weight of a minimum spanning  $k$ -forest  $F$  can be computed in  $O^*(3^n)$  time.*

*Proof.* Since the number of subsets of a set  $S$  of cardinality  $|S|$  is  $2^{|S|}$ , it follows that the time taken for computing an entry  $T(S, i, w) = 2^{|S|}$ , where  $S \subseteq V$ ,  $w \in P$ , and  $0 \leq i \leq k$ .

Also since the number of subsets of size  $j$  in  $V = \binom{n}{j}$ , the time taken for computing all the entries  $T(S, i, w)$  for a fixed  $i$  and  $w = \sum_{j=0}^{j=n} \binom{n}{j} 2^j = 3^n$ . From Lemma 4, it follows that  $w(F)$  can be computed in  $O^*(3^n)$  time.  $\square$

The following theorem follows from Lemmas 4 and 5.

**Theorem 5.** *For an undirected weighted graph  $G$  on  $n$  vertices and a positive integer  $1 < k \leq n$ , there exists an algorithm running in time  $O^*(3^n)$  which computes the minimum spanning  $k$ -forest of  $G$ .*

It is easy to see that keeping track of a subset  $S' \subseteq S$  that gives the optimum value for each  $T(S, i, w)$  will help us to backtrack and compute a minimum spanning  $k$ -forest  $F$  of  $G$ .

We can obtain a faster exact exponential algorithm by reducing the problem of finding a minimum spanning  $k$ -forest of a graph to an instance of the well known problem of finding a  $k$ -partition of a set system defined formally below:

**Input:** Integer  $k > 0$  and a Set system  $(U, S)$ , where  $U$  is a universe of elements and  $S$  is a collection of non-empty subsets of  $U$ .  
**Output:** A  $k$ -partition of  $(U, S)$   
 $S_1, S_2, \dots, S_k$  is a  $k$ -partition of  $(U, S)$  if  $S_i \in S, 1 \leq i \leq k, S_1 \cup S_2 \cup \dots \cup S_k = U$ , and  $S_i \cap S_j = \phi, \forall i \neq j$ .

The assumption is that (all the elements of)  $S$  can be enumerated in time  $O^*(2^n)$ . This additional assumption is needed to guarantee that the overall running time of the inclusion-exclusion algorithm is  $O^*(2^n)$ .

Let  $p_k(S)$  be the number of unordered  $k$ -partitions of  $(U, S)$ , few existing results to compute  $p_k(S)$  are given below.

**Theorem 6** [11]. *The number of (unordered)  $k$ -partitions  $p_k$  of a set system  $(U, S)$  can be computed in*

- (i)  $O^*(2^n)$  time and exponential space
- (ii)  $O^*(3^n)$  time and polynomial space, assuming membership in  $S$  (“ $S' \in S$ ?”), can be decided in polynomial time.

The set system  $(U, S)$  for the problem of finding a spanning  $k$ -forest of weight at most  $w$  of a graph  $G$  is defined as follows.

$$\boxed{\begin{array}{l} U = V \\ S = \{G[S'] \mid S' \subseteq V, G[S'] \text{ is connected and } w(G[S']) \leq w\} \end{array}} \quad (3)$$

Here,  $p_k(S)$  denotes the number of spanning  $k$ -forests of weight at most  $w$  of a graph  $G$ . If  $p_k(S) > 0$ , then there exists a spanning  $k$ -forest of weight at most  $w$ . Clearly, membership in  $S$  is decidable in polynomial time as it involves checking the connectivity and determining the weight of a minimum spanning tree. Combining this with Theorem 6, the following theorem follows.

**Theorem 7.** *For an undirected weighted graph  $G$  on  $n$  vertices and a positive integer  $k > 1$ , the existence of a spanning  $k$ -forest of weight at most  $w$  of a graph  $G$  can be determined in*

- (i)  $O^*(2^n)$  time and exponential space.
- (ii)  $O^*(3^n)$  time and polynomial space.

The above procedure determines the existence of a spanning  $k$ -forest of at most weight  $w$ , to find it, we do the following. For each edge  $e$  in the graph, check whether  $G \setminus e$  has a spanning  $k$ -forest of weight at most  $w$ . If yes, then remove  $e$  from  $G$  else do nothing. At the end, the  $G$  remaining is a spanning  $k$ -forest of weight at most  $w$ . Combining this method along with Theorem 7 and Lemma 4 yields the following theorem.

**Theorem 8.** *For an undirected weighted graph  $G$  on  $n$  vertices and a positive integer  $k > 1$ , the minimum spanning  $k$ -forest of  $G$  can be computed in*

- (i)  $O^*(2^n)$  time and exponential space.
- (ii)  $O^*(3^n)$  time and polynomial space.

## 5 Conclusion

We have given an  $O(kn^3)$  time algorithm for finding a minimum spanning  $k$ -forest of a tree answering a question in an earlier paper. Also, for the special case of a graph having equal weighted edges, we have shown the  $NP$ -Completeness and proposed a  $O(n)$  time algorithm for such trees. A natural open problem would be to find an algorithm for trees that can be extended to bounded tree width graphs.

Future work includes designing an algorithm for graphs having equal weighted edges and improving the exact exponential algorithm with better runtime than  $O^*(2^n)$  or showing a lower bound of  $\Omega(2^n)$  under some possible hypothesis. Another open problem is to design approximation algorithms with a guaranteed quality of approximation. Analysing the quality of approximation for the approximation algorithm in [1] is also an interesting problem.

**Acknowledgements.** We thank Prof. Venkatesh Raman for introducing this problem and for sharing his pearls of wisdom with us during the course of this research. This feat could not have been possible without his supervision and support. We are also immensely grateful to the Institute of Mathematical Sciences, Chennai for facilitating our research.

## References

1. Madkour, A.-R., Nadolny, P., Wright, M.: Finding Minimal Spanning Forests in a Graph. [arXiv:1705.00774v1](#) (2017)
2. Frederickson, G.N., Zhou, S.: Optimal Parametric Search for Path and Tree Partitioning. [arXiv:1711.00599v1](#) (2017)
3. Lukes, J.A.: Efficient algorithm for the partitioning of trees. *IBM J. Res. Dev.* **18**, 217–224 (1974)
4. Perl, Y., Schach, S.R.: Max-min tree partitioning. *J. ACM* **28**, 5–15 (1981)
5. Frederickson, G.N.: Optimal algorithms for partitioning trees and locating p-centers in trees. Technical report CSD-TR-1029, Purdue University (1990)
6. Frederickson, G.N.: Optimal algorithms for tree partitioning. In: Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms, San Francisco, pp. 168–177, January 1991
7. An, Z., Feng, Q., Kanj, I., Xia, G.: The Complexity of Tree Partitioning. [arXiv:1704.05896v1](#) (2017)
8. Caraballo, L.E., Diaz-Báñez, J.-M., Kroher, N.: A Polynomial Algorithm for Balanced Clustering via Graph Partitioning. [arXiv:1801.03347v2](#) (2018)
9. Lesnick, M., Wright, M.: Interactive Visualization of 2-D Persistence Modules. [arXiv:1512.00180](#) (2015)
10. Furer, M., Raghvachari, B.: Approximating the minimum-degree steiner tree to within one of optimal. *J. Algorithms* **17**(3), 409–423 (1994)
11. Formin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer, Heidelberg (2010)
12. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) *Complexity of Computer Computations*. The IBM Research Symposia Series, pp. 85–103. Springer, New York (1972). [https://doi.org/10.1007/978-1-4684-2001-2\\_9](https://doi.org/10.1007/978-1-4684-2001-2_9)



# On the Minmax Regret Path Center Problem on Trees

Biing-Feng Wang<sup>(✉)</sup>, Jhih-Hong Ye, and Chih-Yu Li

Department of Computer Science, National Tsing Hua University,  
Hsinchu 30013, Taiwan, Republic of China  
{bfwang, jhong, cyuli}@cs.nthu.edu.tw

**Abstract.** This paper studies the problem of finding the path center on a tree in which vertex weights are uncertain and the uncertainty is described by given intervals. It is required to find a minmax regret solution, which minimizes the worst-case loss in the objective function. An  $O(n \log n)$ -time algorithm is presented, improving the previous upper bound of  $O(n^2)$ .

## 1 Introduction

The objective of a location problem is to decide the location of facilities in a network so as to minimize the communication or transportation costs [14, 15, 22, 24]. A network usually involves two types of parameters: weights of nodes and lengths of edges. Traditionally, the node weights and edge lengths of a network are assumed to be known precisely. In real transportation systems, the weights and lengths of a network may fluctuate or be inaccurate due to poor measurements. Thus, location models involving uncertainty have attracted significant research efforts [11, 12, 17, 23]. One of the most important ways for modeling network uncertainty is the *minmax regret approach*, introduced by Kouvelis and Yu [16]. In the model, uncertainty of network parameters is characterized by given intervals, and it is required to minimize the worst-case loss in the objective function that may occur because of the uncertain parameters.

Minmax regret location problems have received considerable attention in the past two decades. In network location theory, the shapes of facilities can be points, paths, or trees. Path- and tree-shaped facilities are called extensive facilities [18]. For point-shaped facility problems, most important ones have been studied comprehensively on the minmax regret model [3–6, 8, 9, 16, 31]. However, for extensive facility problems, there are only a few results on the minmax regret model, although there are considerable results on the classical model [7, 18, 20, 26–28]. In a breakthrough paper by Puerto et al. [21], polynomial algorithms were presented for the following three important path-shaped problems: the minmax regret path center, path median, and path centroid problems. Since these problems are NP-hard on general networks, their work was confined to trees. The time complexities of their algorithms are, respectively,  $O(n^2)$ ,  $O(n^4)$ , and  $O(n^5 \log n)$ . In [29, 30] the upper bounds of the minmax regret path median and path centroid problems were improved to  $O(n^2)$  and  $O(n^4)$ , respectively.

**Contribution:** The focus of this paper is the minmax regret path center problem on trees. For this problem, Puerto *et al.*'s algorithm requires  $O(n^2)$  time. This paper



presents an  $O(n \log n)$ -time algorithm. The bottleneck of Puerto *et al.*'s algorithm is to compute the classical path centers of the given tree under  $n$  different settings of node weights. For each setting, their algorithm finds the classical path center in  $O(n)$  time. Our improvement is established on the following simple observation: the  $n$  settings of node weights are almost the same. Based on this observation, we preprocess the given tree in  $O(n \log n)$  time to compute some useful auxiliary data structures; and then use the computed data structures to find the path center under each setting in  $O(\log n)$  time.

Section 2 gives notation and definitions. Section 3 describes Puerto *et al.* algorithm [21] for finding a minmax regret path center of a tree. Section 4 presents efficient algorithms for a problem, called the entry vertex problem, and its extension. Then, using the algorithms in Sect. 4, Sect. 5 gives an improved  $O(n \log n)$ -time algorithm.

## 2 Notation and Definitions

Let  $T = (V, E)$  be a tree, where  $V$  is the vertex set and  $E$  is the edge set. Let  $n = |V|$ . In this paper,  $T$  also denotes the set of all points of the tree. Thus, the notation  $x \in T$  means that  $x$  is a point along any edge of  $T$ , which may or may not be a vertex of  $T$ . Each edge  $e$  has a nonnegative length. For any two points  $p, q \in T$ , let  $P(p, q)$  be the unique path from  $p$  to  $q$  and  $d(p, q)$  be its length. Throughout this paper, we assume that  $T$  has been preprocessed so that  $d(p, q)$  can be answered in  $O(1)$  time for any  $p, q \in V$ . This preprocessing requires  $O(n)$  time [10]. For a subgraph  $X$  of  $T$ , the vertex set and edge set of  $X$  are, respectively,  $V(X)$  and  $E(X)$ . For each vertex  $v \in V$ , the subgraph having a vertex set  $\{v\}$  is simply denoted by  $v$ . For any vertex  $v \in V$  and subgraph  $X$  of  $T$ , the *distance* from  $v$  to  $X$ , denoted by  $d(v, X)$ , is the shortest distance from  $v$  to any point of  $X$  (i.e.,  $d(v, X) = \min_{x \in X} d(v, x)$ ) and *close*( $v, X$ ) is the vertex or point in  $X$  nearest to  $v$ . A path in  $T$  is called a *v-path*, where  $v \in V$ , if  $v$  is one of its endpoints.

Each vertex  $v \in V$  is associated with an interval  $[w_v^-, w_v^+]$ , where  $0 \leq w_v^- \leq w_v^+$ . The *weight* of each vertex  $v \in V$  can be any value in the interval  $[w_v^-, w_v^+]$ . Let  $\Sigma$  be the Cartesian product of intervals  $[w_v^-, w_v^+]$ , where  $v \in V$ . Any element  $S \in \Sigma$  is called a *scenario* and represents a feasible assignment of weights to the vertices of  $T$ . For any scenario  $S \in \Sigma$  and any vertex  $v \in V$ , let  $w_v^S$  be the weight of  $v$  under the scenario  $S$ .

Let  $S \in \Sigma$  be a scenario. For any two subgraphs  $X$  and  $Y$  of  $T$ , the *eccentricity* from  $X$  to  $Y$  under the scenario  $S$  is  $C^S(X, Y) = \max_{v \in V(X)} w_v^S d(v, Y)$ , which is the maximum weighted distance from any vertex in  $X$  to  $Y$  according to the scenario  $S$ . A path  $H$  that minimizes  $C^S(T, H)$  is called a *path center* of  $T$  under the scenario  $S$ . The finding of a path center of  $T$  under a fixed scenario  $S$  is called the *classical path center problem*. We use  $\pi(S)$  to denote a path center of  $T$  under a scenario  $S$ .

For any path  $H$  in  $T$ , the *regret* of  $H$  with respect to a scenario  $S \in \Sigma$  is  $R^S(H) = C^S(T, H) - C^S(T, \pi(S))$  and the *maximum regret* of  $H$  is  $R^*(H) = \max_{S \in \Sigma} R^S(H)$ . The *minmax regret path center problem* is to determine a path  $H$  in  $T$  that minimizes  $R^*(H)$ . The determined path is called a *minmax regret path center*.

For ease of discussion, throughout this paper, we assume that each internal vertex of  $T$  has exactly three neighbors. In case this is not true, the given tree is transformed into an equivalent tree in linear time [13, 19]. Consider an internal vertex  $v \in V$ . There

are three subtrees of  $T$  attached to  $v$  through the edges incident on  $v$ . For each  $(u, v) \in E$ , we denote by  $T_u^v$  the subtree of  $T$  attached to  $v$  through the edge  $(u, v)$ , excluding this edge and the vertex  $v$ . We define the *subtrees* of a path  $H$  to be the subtrees  $T_j^i$  such that  $i$  is an internal node of  $H$  and  $j$  is the neighbor of  $i$  that is not on  $H$ . For any  $p, q \in V$ , define *subtree*( $p, q$ ) to be the union of the subtrees of  $P(p, q)$  (see Fig. 1). For ease of description, sometimes we will orient  $T$  into a rooted tree. In such a case, for each node  $v \in V$ , we use  $p(v)$  and  $sib(v)$  to denote, respectively, its parent and sibling, and use  $T_v$  to denote the subtree of  $T$  rooted at  $v$ .

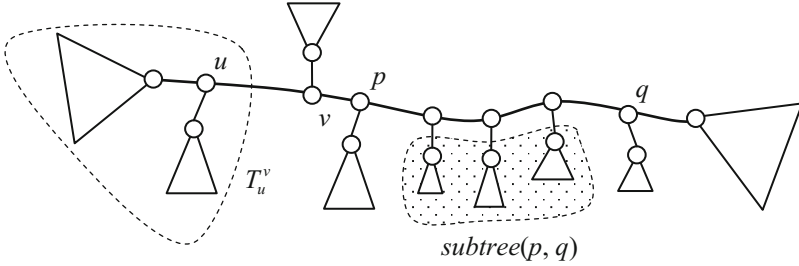


Fig. 1. Subtree  $T_u^v$  and *subtree*( $p, q$ ).

### 3 Puerto, Ricca, and Scozzari’s Algorithm

Puerto et al. [21] had an  $O(n^2)$ -time algorithm for finding a minmax regret path center of a tree. This section reviews their algorithm.

Recall that  $\pi(S)$  denotes a path center of  $T$  under a scenario  $S$ . For any scenario  $S \in \Sigma$ , let  $\alpha(S) = C^S(T, \pi(S))$ . For each  $i \in V$ , let  $S_i$  be the scenario in which the weight of vertex  $i$  is  $w_i^+$  and the weight of any other vertex  $v$  is  $w_v^-$ . Based on an augmented tree approach introduced by Averbakh and Berman [3], Puerto, Ricca, and Scozzari solved the minmax regret path center problem by an elegant transformation to the classical path center problem. Define an auxiliary tree  $T'$  as follows. Let  $M$  be a number that is larger than  $\alpha(S_i)$  for any  $i \in V$ . The tree  $T'$  is obtained from  $T$  by appending to each vertex  $i \in V$  a vertex  $i'$  and an edge  $(i, i')$  with length  $(M - \alpha(S_i))/w_i^+$ . Specific weights are assigned to the vertices of  $T'$ . For each  $i \in V$ , the weight of  $i$  is zero and the weight of  $i'$  is  $w_i^+$ . Let  $P$  be a path in the auxiliary tree  $T'$ . The *restriction* of  $P$  to  $T$  is the path obtained from  $P$  by deleting the edges of  $P$  that are not in  $T$ . Puerto, Ricca, and Scozzari gave the following nice property for solving the minmax regret path center problem.

**Lemma 1 [21].** Let  $P$  be a path center of  $T'$ . Then, the restriction of  $P$  to  $T$  is a minmax regret path center of  $T$ .

Based upon Lemma 1, Puerto, Ricca, and Scozzari solved the minmax regret path center problem in  $O(n^2)$  time as follows. First,  $\alpha(S_i)$  is computed for each  $i \in V$ . By using the linear-time algorithm in [7] for the classical path center problem on a tree, this step is done in  $O(n^2)$  time. Next, the auxiliary tree  $T'$  is constructed, which requires  $O(n)$  time. Finally, a solution is obtained by applying the algorithm in [7] again to  $T'$ .

## 4 The Entry Vertex Problem

A *rooted path center* of a rooted tree with root  $r$  under a fixed scenario is an  $r$ -path  $H$  that minimizes the eccentricity from the tree to  $H$ . For a rooted path center, the endpoint other than the root is called its *terminal*, which may be a vertex or an interior point of an edge. A rooted tree may have more than one rooted path center. However, it is easy to see that the shortest one is unique and can be obtained as follows: initially set the terminal at the root and then continuously extend it toward a farthest vertex below it, until the extension does not decrease the eccentricity.

The entry vertex problem is defined as follows. Let  $T = (V, E)$  be a tree with a fixed scenario  $S$ . For any  $(u, v) \in E$ , let  $Q^S(T_u^v)$  be the *shortest* rooted path center of  $T_u^v$  under the scenario  $S$ , where  $T_u^v$  is considered as a rooted tree with root  $u$ . The *entry vertex* of a vertex  $x$  to a path  $H$  is the vertex on  $H$  nearest to  $x$ . For any  $(u, v) \in E$  and  $x \in V(T_u^v)$ , define  $\text{ENTRY}(x, T_u^v)$  to be a query that returns the entry vertex of  $x$  to the shortest rooted path center,  $Q^S(T_u^v)$ , of  $T_u^v$ . (See Fig. 2) Note that  $\text{ENTRY}(x, T_u^v)$  may not be the same as  $\text{close}(x, Q^S(T_u^v))$ , since only vertices can be entry vertices. The *entry vertex problem* is to preprocess the tree  $T$  such that each  $\text{ENTRY}$  query can be answered efficiently.

This section shows that with an  $O(n \log n)$ -time preprocessing, each  $\text{ENTRY}$  query can be answered in  $O(\log n)$  time.

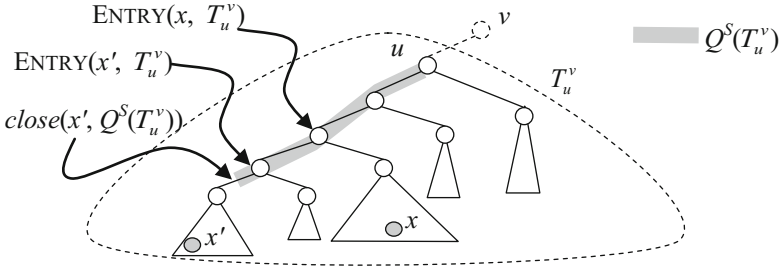


Fig. 2. Entry vertices to the shortest rooted path center of  $T_u^v$ .

### 4.1 Preprocessing

A query  $\text{NODE}(p, q, k)$ , where  $p, q \in V$  and  $k$  is an integer, requests the  $k$ -th vertex on the path from  $p$  to  $q$ . For any two vertices  $p, q \in V$  and scenario  $S \in \Sigma$ , let  $M^S(p, q) = C^S(\text{subtree}(p, q), P(p, q))$ , which is the eccentricity of  $P(p, q)$  from its subtrees. We need the following two lemmas.

**Lemma 2.** With an  $O(n)$ -time preprocessing, a query  $\text{NODE}(p, q, k)$  can be answered in  $O(1)$  time for any  $p, q \in V$  and integer  $k$ .

**Lemma 3.** Suppose that  $C^S(T_u^v, v)$  of all  $(u, v) \in E$  are given. Then, with an  $O(n)$ -time preprocessing,  $M^S(p, q)$  can be computed in  $O(1)$  time for any  $p, q \in V$ .

Given a rooted tree in which each node is associated with a *cost*, a query  $lca(a, b)$  requests the least common ancestor of two vertices  $a, b$ ; a query  $la(v, l)$  requests the level  $l$  ancestor of a vertex  $v$ , where the level of a vertex is the number of edges from it to the root; and a query  $\text{MAX}(a, b)$  requests the largest cost of the vertices on a path  $P(a, b)$ . It was shown in [1, 13] that after an  $O(n)$  time preprocessing, each  $lca$ ,  $la$ , and  $\text{MAX}$  query can be answered in  $O(1)$  time. Using these results, it is not difficult to prove the above two lemmas.

Using the divide-and-conquer approach, Tamir [25] gave an  $O(n \log n)$ -time algorithm to compute  $C^S(T, v)$  for all  $v \in V$ . Based on the same idea and the top tree data structure in [2], we can show the following.

**Lemma 4.** The computation of  $C^S(T_u^v, v)$  for all  $(u, v) \in E$  can be done in  $O(n \log n)$  time.

We proceed to describe the preprocessing algorithm. First, we compute  $C^S(T_u^v, v)$  for every  $(u, v) \in E$ . By Lemma 4, this step requires  $O(n \log n)$  time. Next, by Lemmas 2 and 3, we preprocess  $T$  so that  $\text{NODE}(p, q, k)$  and  $M^S(p, q)$  can be obtained in  $O(1)$  time for any  $p, q \in V$  and integer  $k$ .

## 4.2 Algorithm for Queries

Consider a query  $\text{ENTRY}(x, T_u^v)$ . For notational simplicity, in this section, we assume that  $T_u^v$  is rooted at  $u$ . In addition, since the scenario  $S$  is fixed, we write  $C(\cdot, \cdot)$ ,  $M(\cdot, \cdot)$ , and  $Q(\cdot)$ , respectively, for  $C^S(\cdot, \cdot)$ ,  $M^S(\cdot, \cdot)$ , and  $Q^S(\cdot)$ . Our query algorithm finds the answer in a binary search manner, mainly based upon the following.

**Lemma 5.** A vertex  $i$  of  $T_u^v$  is on the path  $Q(T_u^v)$  if and only if  $C(T_i, i) \geq M(v, i)$ .

**Proof.** Assume first that  $C(T_i, i) \geq M(v, i)$ . Consider an  $u$ -path  $H$  in  $T_u^v$  not containing  $i$ . Since  $C(T_u^v, H) \geq C(T_i, H) > C(T_i, i)$  and  $C(T_u^v, P(u, i)) = \max\{C(T_i, i), M(v, i)\} = C(T_i, i)$ , we have  $C(T_u^v, H) > C(T_u^v, P(u, i))$ . Thus, any  $u$ -path in  $T_u^v$  not containing  $i$  is not a rooted path center. Therefore, the if-part holds.

Next, assume that  $i$  is a vertex on  $Q(T_u^v)$ . By contradiction, suppose that  $C(T_i, i) < M(v, i)$ . Since  $Q(T_u^v)$  passes through  $i$ , we have  $C(T_i, Q(T_u^v)) \leq C(T_i, i) < M(v, i)$ . Therefore,  $C(T_u^v, Q(T_u^v)) = \max\{C(T_i, Q(T_u^v)), M(v, i)\} = M(v, i)$ . Let  $t$  be any point of edge  $(i, p(i))$  such that  $0 < d(i, t) \leq (M(v, i) - C(T_i, i))/w^*$ , where  $w^*$  is the largest weight in  $T_i$ . Consider the path  $P(u, t)$ , which is shorter than  $Q(T_u^v)$ . Clearly,  $C(T_i, t) \leq C(T_i, i) + d(i, t) \times w^* \leq M(v, i)$ . Since  $C(T_i, t) \leq M(v, i)$ , we have  $C(T_u^v, P(u, t)) = \max\{C(T_i, t), M(v, i)\} = M(v, i) = C(T_u^v, Q(T_u^v))$ , which contradicts that  $Q(T_u^v)$  is the shortest rooted path center of  $T_u^v$ . Therefore,  $C(T_i, i) \geq M(v, i)$ . Consequently, the lemma holds.  $\square$

The entry vertex  $m = \text{ENTRY}(x, T_u^v)$  is found as follows. By definition,  $m$  is the first vertex on  $P(x, u)$  that is contained in  $Q(T_u^v)$ . All successors of  $m$  on  $P(x, u)$  are contained in  $Q(T_u^v)$ ; and all predecessors of  $m$  on  $P(x, u)$  are not contained in  $Q(T_u^v)$ . Therefore,  $m$  can be identified by performing binary search on  $P(x, u)$ . With the help of  $\text{NODE}$  queries, any node on  $P(x, u)$  can be accessed in  $O(1)$  time. By Lemma 5, whether a vertex  $i$  is on  $Q(T_u^v)$  can be checked in  $O(1)$  time by using the values of  $C(T_i, i)$  and  $M(v, i)$ . After the preprocessing in Sect. 4.1,  $C(T_i, i) = \max\{C(T_a^i, i), C(T_b^i, i)\}$  and  $M(v, i)$

$i$ ) can be computed in  $O(1)$  time for any vertex  $i$  in  $T_u^v$ , where  $a$  and  $b$  are the two children of  $i$ . Therefore, we have the following.

**Theorem 1.** With an  $O(n \log n)$ -time preprocessing, each ENTRY query can be answered in  $O(\log n)$  time.

### 4.3 An Extended Problem

Our improvement on the minmax regret path center problem is based on solving an extended version of the entry vertex problem, in which it is allowed to temporarily increase the weight of a vertex during a query. For any  $i \in V$  and  $w \geq w_i^S$ , we use  $S|(i, w)$  to denote the scenario obtained from  $S$  by increasing the weight of  $i$  to  $w$ . A query EXTENDEENTRY( $x, T_u^v, i, w$ ) reports the entry vertex of  $x$  to the rooted path center of  $T_u^v$  under the scenario  $S|(i, w)$ , where  $(u, v) \in E$ ,  $x \in V(T_u^v)$ ,  $i \in V$ , and  $w \geq w_i^S$ . That is, EXTENDEENTRY( $x, T_u^v, i, w$ ) reports the entry vertex of  $x$  to the path  $Q^{S|(i, w)}(T_u^v)$ . In the following, we show that after an  $O(n \log n)$ -time preprocessing, each EXTENDEENTRY query can also be answered in  $O(\log n)$  time. Using the *lca* algorithm in [13], it is not difficult to prove the following lemma.

**Lemma 6.** With an  $O(n)$ -time preprocessing,  $close(x, P(p, q))$  can be computed in  $O(1)$  time for any three vertices  $p, q, x \in V$ .

**Lemma 7.** With an  $O(n \log n)$ -time preprocessing,  $C^{S|(i, w)}(T_u^v, v)$  and  $M^{S|(i, w)}(p, q)$  can be computed in  $O(1)$  time for any  $i, p, q \in V$ ,  $(u, v) \in E$ , and  $w \geq w_i^S$ .

**Proof.** As in Sect. 4.1, we preprocess  $T$  so that  $C^S(T_u^v, v)$  for any  $(u, v) \in E$  and  $M^S(p, q)$  for any vertices  $p, q \in V$  can be computed in  $O(1)$  time. In addition, we preprocess  $T$  so that  $close(x, P(p, q))$  can be accessed in  $O(1)$  time for any  $x, p, q \in V$ .

For any  $i \in V$ ,  $(u, v) \in E$ , and  $w \geq w_i^S$ , since  $S|(i, w)$  differs from  $S$  only in the weight of  $i$ ,  $C^{S|(i, w)}(T_u^v, v)$  is computed in  $O(1)$  time as follows. First, determine whether  $i \in T_u^v$  by checking whether  $close(i, P(u, v)) = u$ . Next, if  $i \in T_u^v$ , we set  $C^{S|(i, w)}(T_u^v, v) = \max\{C^S(T_u^v, v), w \times d(i, v)\}$ ; otherwise, we set  $C^{S|(i, w)}(T_u^v, v) = C^S(T_u^v, v)$ . For any  $i, p, q \in V$ ,  $M^{S|(i, w)}(p, q)$  is computed in  $O(1)$  time as follows. First, determine whether  $i$  is a vertex in  $subtree(p, q)$  or an internal node of  $P(p, q)$  by checking whether  $close(i, P(p, q)) \notin \{p, q\}$ . Next, if  $i$  is a vertex in  $subtree(p, q)$  or an internal node of  $P(p, q)$ , we set  $M^{S|(i, w)}(p, q) = \max\{M^S(p, q), w \times d(i, close(i, P(p, q)))\}$ ; otherwise, we set  $M^{S|(i, w)}(p, q) = M^S(p, q)$ . Consequently, the lemma holds.  $\square$

Consider a query EXTENDEENTRY( $x, T_u^v, i, w$ ). According to the query algorithm in Sect. 4.2, to show that this query can be answered in  $O(\log n)$  time, it suffices to show that  $C^{S|(i, w)}(T_u^v, v)$ ,  $M^{S|(i, w)}(p, q)$ , and NODE( $p, q, k$ ) can be obtained in  $O(1)$  time for any  $i, p, q \in V$ ,  $(u, v) \in E$ , and integer  $k$ . As a result, by combining Lemmas 2 and 7, we obtain the following.

**Theorem 2.** Let  $T$  be a tree with a fixed scenario  $S$ . With an  $O(n \log n)$ -time preprocessing, each EXTENDEENTRY query can be answered in  $O(\log n)$  time.

## 5 An Improved Algorithm for the Path Center Problem

The bottleneck of the algorithm in [21] is to compute  $\alpha(S_i)$  for every  $i \in V$ . Recall that for any scenario  $S \in \Sigma$ ,  $\alpha(S)$  denotes  $C^S(T, \pi(S))$  and for each  $i \in V$ ,  $S_i$  denotes the scenario in which the weight of vertex  $i$  is  $w_i^+$  and the weight of any other vertex  $v$  is  $w_v^-$ . In this section, we improve the upper bound of the minmax regret path center problem on a tree by showing that the computation of all  $\alpha(S_i)$  can be done in  $O(n \log n)$  time.

Let  $S^-$  be the scenario in which the weight of every vertex  $v$  is  $w_v^-$ . The scenario  $S^-$  differs from each  $S_i$  only in the weight of vertex  $i$ . Our idea is to preprocess  $T$  under the scenario  $S^-$ , so that each  $\alpha(S_i)$  can be determined efficiently. Let  $M^S(p, q)$  and  $Q^S(T_u^v)$  be defined the same as in Sect. 4. For any  $(u, v) \in E$  and scenario  $S$ , let  $\lambda^S(T_u^v) = C^S(T_u^v, Q^S(T_u^v))$ , which is the eccentricity from  $T_u^v$  to the rooted path center  $Q^S(T_u^v)$ . For notational simplicity, in this section,  $C^{S^-}(\cdot, \cdot)$ ,  $C^{S_i}(\cdot, \cdot)$ ,  $M^{S^-}(\cdot, \cdot)$ ,  $M^{S_i}(\cdot, \cdot)$ ,  $l^{S^-}(\cdot)$ , and  $\lambda^{S_i}(\cdot)$  are simply denoted, respectively, by  $C^-(\cdot, \cdot)$ ,  $C^i(\cdot, \cdot)$ ,  $M^-(\cdot, \cdot)$ ,  $M^i(\cdot, \cdot)$ ,  $\lambda^-(\cdot)$ , and  $\lambda^i(\cdot)$ .

**Lemma 8.** Suppose that  $C^-(T_u^v, v)$  for all  $(u, v) \in E$  are given. In  $O(n)$  time, we can compute  $\lambda^-(T_u^v)$  for all edges  $(u, v) \in E$ .

**Proof.** In this proof, we assume that  $T$  is under the scenario  $S^-$ . We orient  $T$  into a rooted tree with an arbitrary root  $r$ . Since there always exists a rooted path center whose terminal is a leaf, using the dynamic programming approach, all  $\lambda^-(T_u^v)$  are computed in two phases.

- Phase 1. This phase computes  $\lambda^-(T_x)$  for all  $x \in V$  in a bottom-up manner as follows. If  $x$  is a leaf, we have  $\lambda^-(T_x) = 0$ . Assume that  $x$  is an internal vertex and let  $x_1, x_2$  be its two children. Let  $H$  be a rooted path center of  $T_x$ . If  $H$  passes through  $x_1$ , since  $\lambda^-(T_x) = C^-(T_x, H) = \max\{C^-(T_{x_1}, H), C^-(T_{x_2}, x)\}$  and a rooted path center of  $T_{x_1}$  has the minimum eccentricity from  $T_{x_1}$  among all  $x_1$ -paths, it can be concluded that  $\lambda^-(T_x) = \max\{\lambda^-(T_{x_1}), C^-(T_{x_2}, x)\}$ . Similarly, if  $H$  passes through  $x_2$ , it can be concluded that  $\lambda^-(T_x) = \max\{C^-(T_{x_1}, x), \lambda^-(T_{x_2})\}$ . Therefore, we compute  $\lambda^-(T_x)$  as  $\min\{\max\{\lambda^-(T_{x_1}), C^-(T_{x_2}, x)\}, \max\{C^-(T_{x_1}, x), \lambda^-(T_{x_2})\}\}$ .
- Phase 2. This phase computes  $\lambda^-(T_{p(x)}^x)$  for all  $x \in V$  in a top-down manner as follows. If  $x$  is the root  $r$ , we have  $\lambda^-(T_{p(x)}^x) = \lambda^-(\emptyset) = 0$ . Assume that  $x \neq r$ . A rooted path center of  $T_{p(x)}^x$  passes through either  $sib(x)$  or  $p(p(x))$ . If it passes through  $sib(x)$ , we have  $\lambda^-(T_{p(x)}^x) = \max\{\lambda^-(T_{sib(x)}), C^-(T_{p(p(x))}^{p(x)}), p(x)\}$ ; otherwise, we have  $\lambda^-(T_{p(x)}^x) = \max\{C^-(T_{sib(x)}, p(x)), \lambda^-(T_{p(p(x))}^{p(x)}), p(x)\}$ . Therefore,  $\lambda^-(T_{p(x)}^x)$  can be computed in  $O(1)$  time.

The above computation requires  $O(n)$  time. Thus, the lemma holds.  $\square$

**Lemma 9.** Suppose that the following can be accessed in  $O(1)$  time:  $\lambda^-(T_u^v)$  for any  $(u, v) \in E$ ,  $C^i(T_u^v, v)$  for any  $(u, v) \in E$  and  $i \in V$ , and  $M^i(p, q)$  for any  $i, p, q \in V$ ; and suppose that the entry vertex of  $i$  to  $Q^{S_i}(T_u^v)$  can be accessed in  $O(\log n)$  time for any  $(u, v) \in E$  and  $i \in V$ . Then,  $\lambda^i(T_u^v)$  can be computed in  $O(\log n)$  time for any  $(u, v) \in E$  and  $i \in V$ .

**Proof.** We prove this lemma by presenting an algorithm. For ease of description, assume that  $T_u^v$  is rooted at  $u$  and is under the scenario  $S_i$ . First, compute  $m$  as the entry vertex of  $i$  to  $Q^{S_i}(T_u^v)$  in  $O(\log n)$  time. Let  $m_1, m_2$  be the two children of  $m$ . Next, in  $O(1)$  time, we find the values of  $C^i(T_{m_1}, m)$  and  $C^i(T_{m_2}, m)$ . By symmetry, assume that  $C^i(T_{m_1}, m) \geq C^i(T_{m_2}, m)$ . We first establish the following claim.

**Claim.** There is a rooted path center of  $T_u^v$  (under  $S_i$ ) that passes through  $m_1$ .

**Proof of the Claim.** Let  $P(u, t) = Q^{S_i}(T_u^v)$ . Clearly, any  $u$ -path containing  $Q^{S_i}(T_u^v)$  is a rooted path center. Thus, to prove this claim, we only need to show that the terminal  $t$  is a point of edge  $(m, m_1)$  or is in  $T_{m_1}$ . Since  $m$  is a vertex on  $Q^{S_i}(T_u^v)$ ,  $Q^{S_i}(T_u^v)$  can be obtained by initially setting the terminal  $t$  at  $m$  and then continuously extending it toward a farthest vertex below it, until the extension does not decrease the eccentricity. Since  $C^i(T_{m_1}, m) \geq C^i(T_{m_2}, m)$ , it is easy to conclude that at  $t = m$  an extension can decrease the eccentricity only if it is toward the vertex  $m_1$ . Therefore,  $t$  is a point of edge  $(m, m_1)$  or is in  $T_{m_1}$ . Consequently, the claim follows.

We now complete the proof of the lemma. Let  $H$  be a rooted path center of  $T_u^v$  that passes through  $m_1$ . Two cases are discussed.

Case 1:  $i \in V(T_{m_1})$ .

In this case,  $m_1$  is not on the shortest path center  $Q^{S_i}(T_u^v)$ . Otherwise, since  $m_1$  is closer to  $x$  than  $m$ ,  $m$  is not the entry vertex of  $i$  to  $Q^{S_i}(T_u^v)$ . By Lemma 4,  $C^i(T_{m_1}, m_1) < M^i(v, m_1)$ . Since  $H$  passes through  $m_1$ , we have  $C^i(T_{m_1}, H) \leq C^i(T_{m_1}, m_1) < M^i(v, m_1)$ . Therefore,  $\lambda^i(T_u^v) = C^i(T_u^v, H) = \max\{C^i(T_{m_1}, H), C^i(\text{subtree}(v, m_1), H)\} = \max\{C^i(T_{m_1}, H), M^i(v, m_1)\} = M^i(v, m_1)$ . Consequently, in this case, we compute  $\lambda^i(T_u^v) = M^i(v, m_1)$  in  $O(1)$  time.

Case 2:  $i \notin V(T_{m_1})$ .

Since  $i \notin V(T_{m_1})$ , we have  $C^i(T_{m_1}, H) = C^-(T_{m_1}, H)$  and thus  $C^i(T_u^v, H) = \max\{C^-(T_{m_1}, H), M^i(v, m_1)\}$ . Let  $H^*$  be the union of  $P(u, m_1)$  and  $Q^{S^-}(T_{m_1})$ . Under  $S^-$ , the path  $Q^{S^-}(T_{m_1})$  has the minimum eccentricity from  $T_{m_1}$  among all  $m_1$ -paths in  $T_{m_1}$ . Consequently, it can be concluded that  $C^i(T_u^v, H^*) = \max\{C^-(T_{m_1}, H^*), M^i(v, m_1)\} \leq C^i(T_u^v, H)$ . Therefore,  $H^*$  is also a rooted path center of  $T$  under  $S_i$  and thus  $\lambda^i(T_u^v) = \max\{C^-(T_{m_1}, H^*), M^i(v, m_1)\} = \max\{\lambda^-(T_{m_1}), M^i(v, m_1)\}$ . Consequently, in this case, we compute  $\lambda^i(T_u^v) = \max\{\lambda^-(T_{m_1}), M^i(v, m_1)\}$  in  $O(1)$  time.

The above computation of  $\lambda^i(T_u^v)$  requires  $O(\log n)$  time. Thus, the lemma holds.  $\square$

A *discrete 1-center* of  $T$  under a scenario  $S$  is a vertex  $v \in V$  that minimizes  $C^S(T, v)$ . Tamir *et al.* [26] gave the following.



**Lemma 10 [26].** Let  $T$  be a tree with a fixed scenario and  $c$  be its discrete 1-center. Then,  $T$  has a path center that contains  $c$ .

We proceed to present an algorithm for computing  $\alpha(S_i)$  of all  $i \in V$ . Consider the computation for a fixed  $i \in V$ . Assume that  $T$  is under the scenario  $S_i$ . Let  $c$  be a discrete 1-center of  $T$ . By Lemma 10, there is a path center containing  $c$ . Let  $x, y, z$  be the three children of  $c$ . Without losing any generality, assume that  $C^i(T_x^c, c) \geq C^i(T_y^c, c) \geq C^i(T_z^c, c)$ . Then, there exists a path center that passes through  $x$  and  $y$  [26]. For any path  $H$  passing through  $x$  and  $y$ , we have  $C^i(T, H) = \max\{C^i(T_x^c, H), C^i(T_y^c, H), C^i(T_z^c, c)\}$ . Let  $H^*$  be the union of  $Q^{S_i}(T_x^c)$ ,  $P(x, y)$ , and  $Q^{S_i}(T_y^c)$ . The path  $Q^{S_i}(T_x^c)$  has the minimum eccentricity from  $T_x^c$  among all  $x$ -paths in  $T_x^c$ ; and the path  $Q^{S_i}(T_y^c)$  has the minimum eccentricity from  $T_y^c$  among all  $y$ -paths in  $T_y^c$ . Consequently, it can be concluded that  $H^*$  has the minimum eccentricity among all paths that pass through  $x$  and  $y$ . That is,  $H^*$  is a path center of  $T$ . Therefore,  $\alpha(S_i)$  can be computed as  $C^i(T, H^*) = \max\{C^i(T_x^c, Q^{S_i}(T_x^c)), C^i(T_y^c, Q^{S_i}(T_y^c)), C^i(T_z^c, c)\} = \max\{\lambda^i(T_x^c), \lambda^i(T_y^c), C^i(T_z^c, c)\}$ .

Based upon the above discussion, an algorithm for computing all  $\alpha(S_i)$  is described as follows.

**Algorithm 1.** ALLPATHCENTERS

**Input:** a tree  $T = (V, E)$  and  $[w_i^-, w_i^+]$  for each vertex  $i \in V$

**Output:**  $\alpha(S_i)$  for each vertex  $i \in V$

**begin**

1. preprocess  $T$  so that  $C^-(T_u^v, v)$ ,  $C^i(T_u^v, v)$ , and  $M^i(p, q)$  can be obtained in  $O(1)$  time for any  $(u, v) \in E$  and  $i, p, q \in V$
2. preprocess  $T$  so that the entry vertex of  $x$  to  $Q^{S_i}(T_u^v)$  can be determined in  $O(\log n)$  time for any  $x, i \in V$  and  $(u, v) \in E$
3. compute  $\lambda^-(T_u^v)$  for every  $(u, v) \in E$
4. compute the discrete 1-center of  $T$  under  $S_i$  for each  $i \in V$
5. **for** each vertex  $i \in V$  **do**       /\* compute  $\alpha(S_i)$
6.   **begin**
7.     $c \leftarrow$  the discrete 1-center computed for  $S_i$  in Line 4
8.     $(x, y, z) \leftarrow$  the three neighbors of  $c$ , where  $C^i(T_x^c, c) \geq C^i(T_y^c, c) \geq C^i(T_z^c, c)$
9.    compute  $\lambda^i(T_x^c)$  and  $\lambda^i(T_y^c)$
10.    $\alpha(S_i) \leftarrow \max\{\lambda^i(T_x^c), \lambda^i(T_y^c), C^i(T_z^c, c)\}$
11.   **end**
12. **return**  $(\{\alpha(S_i) \mid i \in V\})$

**end**

Since  $S_i = S^- \lfloor (i, w_i^+)$  for each  $i \in V$ , by using Lemmas 4 and 7 with  $S = S^-$ , Line 1 requires  $O(n \log n)$  time. By definition, when  $T$  is under  $S^-$ , the entry vertex of  $x$  to  $Q^{S_i}(T_u^v)$  is  $\text{EXTENDENTRY}(x, T_u^v, i, w_i^+)$ . Therefore, by using Theorem 2 with  $S = S^-$ , Line 2 requires  $O(n \log n)$  time. By Lemma 8, Line 3 takes  $O(n)$  time. Yu et al. [31] showed that a discrete 1-center of  $T$  under  $S_i$  can be computed in  $O(n \log n)$  time for every  $i \in V$ . Thus, Line 4 requires  $O(n \log n)$  time. Consider the for-loop in Lines 5–11. Lines 7, 8, 10 take  $O(1)$  time. By Lemma 9, after the preprocessing in Lines 1, 2, and 3, the computation of  $\lambda^i(T_x^c)$  and  $\lambda^i(T_y^c)$  in Line 9 can be done in  $O(\log n)$  time. Therefore, each iteration of the for-loop requires  $O(\log n)$  time. As a result, we obtain the following.



**Lemma 11.** We can compute  $\alpha(S_i)$  for all  $i \in V$  in  $O(n \log n)$  time.

**Theorem 3.** The minmax regret path center problem on a tree can be solved in  $O(n \log n)$  time.

## References

1. Alstrup, S., Holm, J.: Improved algorithms for finding level ancestors in dynamic trees. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 73–84. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-45022-X\\_8](https://doi.org/10.1007/3-540-45022-X_8)
2. Alstrup, S., Lauridsen, Peter W., Sommerlund, P., Thorup, M.: Finding cores of limited length. In: Dehne, F., Rau-Chaplin, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1997. LNCS, vol. 1272, pp. 45–54. Springer, Heidelberg (1997). [https://doi.org/10.1007/3-540-63307-3\\_47](https://doi.org/10.1007/3-540-63307-3_47)
3. Averbakh, I., Berman, O.: Minimax regret p-center location on a network with demand uncertainty. *Locat. Sci.* **5**(4), 247–254 (1997)
4. Averbakh, I., Berman, O.: Minmax regret median location on a network under uncertainty. *INFORMS J. Comput.* **12**(2), 104–110 (2000)
5. Bhattacharya, B., Kameda, T., Song, Z.: A linear time algorithm for computing minmax regret 1-median on a tree network. *Algorithmica* **70**(1), 2–21 (2014)
6. Bhattacharya, B., Kameda, T., Song, Z.: Minmax regret 1-center algorithms for path/tree/unicycle/cactus networks. *Discrete Appl. Math.* **195**, 18–30 (2015)
7. Bhattacharya, B., Shi, Q., Tamir, A.: Optimal algorithms for the path/tree-shaped facility location problems in trees. *Algorithmica* **55**(4), 601–618 (2009)
8. Conde, E.: Minmax regret location–allocation problem on a network under uncertainty. *Eur. J. Oper. Res.* **179**(3), 1025–1039 (2007)
9. Conde, E.: A note on the minmax regret centroid location on trees. *Oper. Res. Lett.* **36**(2), 271–275 (2008)
10. Djidjev, H.N., Pantziou, G.E., Zaroliagis, C.D.: Computing shortest paths and distances in planar graphs. In: Albert, J.L., Monien, B., Artalejo, M.R. (eds.) ICALP 1991. LNCS, vol. 510, pp. 327–338. Springer, Heidelberg (1991). [https://doi.org/10.1007/3-540-54233-7\\_145](https://doi.org/10.1007/3-540-54233-7_145)
11. Drezner, Z.: Sensitivity analysis of the optimal location of a facility. *Naval Res. Logist. Q.* **32**(2), 209–224 (1985)
12. Frank, H.: Optimum locations on a graph with probabilistic demands. *Oper. Res.* **14**(3), 409–421 (1966)
13. Harel, D., Tarjan, R.: Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* **13**(2), 338–355 (1984)
14. Kariv, O., Hakimi, S.L.: An algorithmic approach to network location problems. I: the p-centers. *SIAM J. Appl. Math.* **37**(3), 513–538 (1979)
15. Kariv, O., Hakimi, S.L.: An algorithmic approach to network location problems. II: the p-medians. *SIAM J. Appl. Math.* **37**(3), 539–560 (1979)
16. Kouvelis, P., Yu, G.: *Robust Discrete Optimization and Its Applications*. Springer, New York (2013). <https://doi.org/10.1007/978-1-4757-2620-6>
17. Labbé, M., Thisse, J.-F., Wendell, R.E.: Sensitivity analysis in minisum facility location problems. *Oper. Res.* **39**(6), 961–969 (1991)
18. Minieka, E.: The optimal location of a path or tree in a tree network. *Networks* **15**(3), 309–321 (1985)
19. Mirchandani, P.B., Odoni, A.R.: Locations of medians on stochastic networks. *Transp. Sci.* **13**(2), 85–97 (1979)

20. Morgan, C.A., Slater, P.J.: A linear algorithm for a core of a tree. *J. Algorithms* **1**(3), 247–258 (1980)
21. Puerto, J., Ricca, F., Scozzari, A.: Minimax regret path location on trees. *Networks* **58**(2), 147–158 (2011)
22. Puerto, J., Rodríguez-Chía, A.M., Tamir, A., Pérez-Brito, D.: The bi-criteria doubly weighted center-median path problem on a tree. *Networks* **47**(4), 237–247 (2006)
23. Synder, L.: Facility location under uncertainty: a review. *IIE Trans.* **38**(7), 547–564 (2006)
24. Tamir, A.: An  $O(pn^2)$  algorithm for the p-median and related problems on tree graphs. *Oper. Res. Lett.* **19**(2), 59–64 (1996)
25. Tamir, A.: Sorting weighted distances with applications to objective function evaluations in single facility location problems. *Oper. Res. Lett.* **32**(3), 249–257 (2004)
26. Tamir, A., Puerto, J., Mesa, J.A., Rodríguez-Chía, A.M.: Conditional location of path and tree shaped facilities on trees. *J. Algorithms* **56**(1), 50–75 (2005)
27. Tamir, A., Puerto, J., Pérez-Brito, D.: The centdian subtree on tree networks. *Discrete Appl. Math.* **118**(3), 263–278 (2002)
28. Wang, B.-F.: Efficient parallel algorithms for optimally locating a path and a tree of a specified length in a weighted tree network. *J. Algorithms* **34**(1), 90–108 (2000)
29. Ye, J.-H., Li, C.-Y., Wang, B.-F.: An improved algorithm for the minmax regret path centdian problem on trees. Submitted to journal for publication, under revision
30. Ye, J.-H., Wang, B.-F.: On the minmax regret path median problem on trees. *J. Comput. Syst. Sci.* **81**(7), 1159–1170 (2015)
31. Yu, H.-I., Lin, T.-C., Wang, B.-F.: Improved algorithms for the minmax-regret 1-center and 1-median problems. *ACM Trans. Algorithms* **4**(3), 1–27 (2008)



# A Strongly Polynomial Time Algorithm for the Maximum Supply Rate Problem on Trees

Koki Takayama<sup>(✉)</sup> and Yusuke Kobayashi

University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan  
s1720509@s.tsukuba.ac.jp

**Abstract.** Suppose that we are given a graph whose each vertex is either a supply vertex or a demand vertex and is assigned a nonnegative integer supply or demand value. We consider partitioning  $G$  into connected components by removing edges from  $G$  so that each connected component has exactly one supply vertex and there exists a flow in each connected component satisfying the supply/demand constraints. The problem that determines the existence of such a partition is called the partition problem. Ito et al. (2005) showed that the partition problem is  $\mathcal{NP}$ -complete in general and it can be solved in linear time if the given graph is a tree. When the graph does not have such a partition, we scale the demand values uniformly by scale factor  $r$  so that the obtained graph has a desired partition. The maximum supply rate problem is the problem that finds the maximum value of such  $r$ . Whereas the maximum supply rate problem is  $\mathcal{NP}$ -hard in general in the same way as the partition problem, Morishita and Nishizeki (2015) gave a weakly polynomial-time algorithm for the problem on trees.

In this paper, we give a first strongly polynomial-time algorithm for the maximum supply rate problem on trees. Our algorithm is based on the dynamic programming technique, in which we compute “surplus” and “deficit” of the supply in subproblems from leaves to the root. We use piecewise linear functions of  $r$  to represent them, and one of our important contributions is to bound the size of the representation of each function.

## 1 Introduction

Let  $G$  be an undirected graph whose each vertex is either a supply vertex or a demand vertex. Each supply (resp. demand) vertex is assigned a nonnegative integer *supply value* (resp. *demand value*), which represents the amount of flow that the vertex can send (resp. has to receive). We consider partitioning  $G$  into connected components by removing edges from  $G$  so that each connected component has exactly one supply vertex and there exists a flow in each connected

---

This work is partially supported by ACT-I, JST and by JSPS KAKENHI Grant Numbers JP16K16010 and JP16H03118.

component satisfying the supply/demand constraints. Ito et al. [1] introduce the decision problem that asks whether  $G$  has such a partition, which they call the *partition problem*. It is mentioned in [1, 2] that we can also consider the problem with edge capacity constraints. The partition problem is a model of power delivery networks and VLSI circuits [3, 4].

It is obvious that not all graphs have such a partition. If  $G$  does not have such a partition, we scale the demand values uniformly by scale factor  $r$  so that the obtained graph has a partition. Morishita and Nishizeki [5] introduce the *maximum supply rate problem* as the problem that finds the maximum value of such  $r$ . The maximum supply rate problem is a special case of the partition problem for parametric networks that are a model of power delivery networks, in which demand values are changed by time, temperature, oil price, etc. [5, 6].

We now give a formal definition of the partition problem and show some known results. A *power delivery network* is expressed by an undirected graph  $G = (V, E)$  with vertex set  $V$  and edge set  $E$ . Each vertex of  $G$  is either a supply vertex or a demand vertex. That is,  $V = S \cup D$  and  $S \cap D = \emptyset$ , where  $S$  is the set of all supply vertices and  $D$  is the set of all demand vertices. Each supply vertex  $v \in S$  is assigned a nonnegative integer supply value  $s(v) \in \mathbb{Z}_+$ , and each demand vertex  $v \in D$  is assigned a nonnegative integer demand value  $d(v) \in \mathbb{Z}_+$ . Here,  $\mathbb{Z}_+$  denotes the set of all nonnegative integers. Each edge  $e \in E$  is assigned a nonnegative integer capacity  $c(e) \in \mathbb{Z}_+$ .

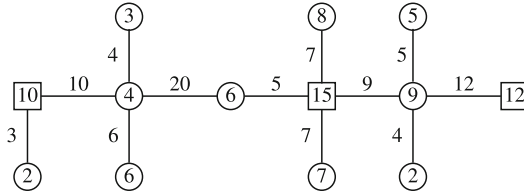
We partition  $G$  into some connected components by removing some edges from  $G$  so that each connected component has exactly one supply vertex and there exists a flow in each connected component satisfying the supply/demand/capacity constraints. Such a partition is called a *feasible partition* [5]. More precisely, a feasible partition consists of  $|S|$  subsets  $V_1, V_2, \dots, V_{|S|}$  of  $V$  such that  $V_i \cap V_j = \emptyset$  for each  $i, j$  ( $i \neq j$ ),  $V_1 \cup \dots \cup V_{|S|} = V$ , and the following conditions hold.

- For each  $i$ , the subgraph  $G[V_i]$  of  $G$  induced by  $V_i$  is connected.
- For each  $i$ ,  $V_i$  has exactly one supply vertex. That is, there exists a vertex  $u_i \in V$  such that  $S \cap V_i = \{u_i\}$ .
- There exists a flow satisfying the following.
  - The inflow of each demand vertex  $v \in D$  is exactly  $d(v)$ .
  - The outflow of each supply vertex  $v \in S$  is at most  $s(v)$ .
  - The flow does not pass through edges between  $V_i$  and  $V_j$  for distinct  $i, j$ .
  - The amount of flow passing through each edge  $e \in E$  is at most the edge capacity  $c(e)$ .

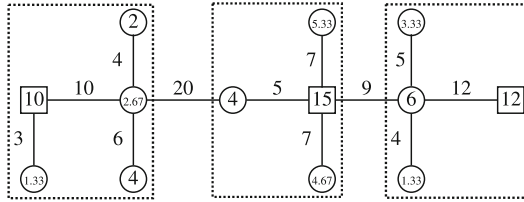
The *partition problem* is the decision problem that asks whether a given graph  $G$  has a feasible partition or not, and is  $\mathcal{NP}$ -complete even for series-parallel graphs [1]. On the other hand, the partition problem on trees can be solved in linear time by dynamic programming [1, 2].

When  $G$  has no feasible partition, we try to modify the instance so that the obtained instance has a feasible partition. A natural modification is to scale every demand value uniformly by scale factor  $r$ , i.e., replacing the demand  $d(v)$  of  $v$  with  $r \cdot d(v)$ . The *maximum supply rate* is defined as the maximum value

of  $r$  such that  $G$  has a feasible partition with respect to  $r \cdot d$ , where  $r$  might be greater than 1. The *maximum supply rate problem* is the problem of finding the maximum supply rate  $r^*$ , which is introduced by Morishita and Nishizeki [5]. An instance of the maximum supply rate problem is shown in Fig. 1. The maximum supply rate is  $\frac{2}{3}$  in this instance.



(a) A tree  $T$  having no partition.



(b) A new tree  $T'$  constructed from  $T$  and a feasible partition for the maximum supply rate  $r^* = 2/3$ .

**Fig. 1.** An example of the partition problem and the maximum supply rate problem, where a supply vertex is represented as a square with a supply value, a demand vertex is represented as a circle with a demand value, and an edge is represented as a line with an edge capacity.

Since the partition problem is  $\mathcal{NP}$ -complete, the maximum supply rate problem is also  $\mathcal{NP}$ -hard even for series-parallel graphs [1]. However, if the graph is a tree, we can solve the maximum supply rate problem in weakly polynomial-time by guessing the maximum supply rate with a binary search and by applying the linear time algorithm for the partition problem [5, 6].

The main result of this paper is to show the following theorem.

**Theorem 1.** *The maximum supply rate problem on trees can be solved in time  $\mathcal{O}(n^3)$ , where  $n$  is the number of vertices of the tree.*

This theorem gives a first strongly polynomial-time algorithm for the maximum supply rate problem on trees. In order to make the running time strongly polynomial, we do not guess the maximum supply rate  $r^*$  by binary search. We adopt the dynamic programming approach instead. In our algorithm, we regard  $T$  as a rooted tree by choosing an arbitrary vertex as the root. We pick up each vertex  $v \in V$  in the order from leaves to the root and compute “surplus” of the subtree rooted at  $v$  (see Sect. 3 for details).

We now point out two difficulties in designing the algorithm. The first difficulty is that information on the surplus cannot be represented as a dynamic programming table of finite size. This is because the surplus depends on the supply rate  $r$  that can take any nonnegative real number. To overcome this difficulty, we represent the surplus of each subtree as a piecewise linear function of  $r$ . Note that a similar technique was also used in [5, 6]. The second difficulty is in obtaining a polynomial bound on the size of the function representation of the surplus. If we adopt a naive piecewise linear function to represent the surplus, then it is hard to obtain a polynomial bound on its size (see Sect. 3.1 for details). Our key idea is to represent the surplus by using multiple piecewise linear functions. This makes the update formula quite simple and enables us to show that the representation of the surplus has a polynomial size.

Theorem 1 can be extended to a generalized problem in which the supply (resp. demand) values are represented by monotonically decreasing (resp. increasing) piecewise linear functions. By using the same argument as Theorem 1, we show that this problem can also be solved in polynomial time. See Sect. 5 for the formal statement of our result.

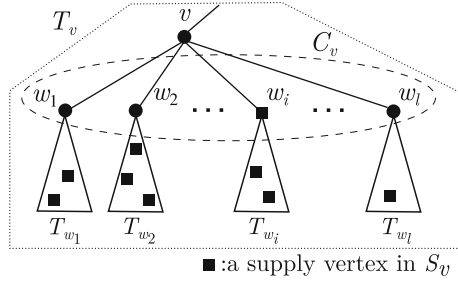
We here describe some related results. The *maximum partition problem* introduced by Ito et al. [1] is another optimization version of the partition problem. If a graph does not have a feasible partition, we partition the graph into some connected components so that each component has at most one supply vertex and the sum of the demand values is at most the supply value in each component with a supply vertex. That is, there might exist components without a supply vertex. The maximum partition problem is the problem of finding such a partition that maximizes the sum of the demand values in all the components with supply vertices. This problem is also  $\mathcal{NP}$ -hard even for a star with exactly one supply vertex. For the maximum partition problem on restricted classes of graphs, pseudo-polynomial-time algorithms and fully polynomial-time approximation schemes are proposed in [1, 2, 7, 8]. Recently, there has been studied a heuristic method for solving the problem [9].

The remaining of this paper is organized as follows. In Sect. 2, we give preliminary definitions. In Sect. 3, we give an outline of our algorithm and describe a strongly polynomial-time algorithm for the maximum supply rate problem on trees. In Sect. 4, we analyze the running time of our algorithm. Finally, in Sect. 5, we extend our result to a problem on parametric networks.

## 2 Preliminaries

One may assume without loss of generality that a given tree  $T = (V, E)$  is rooted at an arbitrarily chosen vertex  $v_{\text{root}}$ . For a vertex  $v \in V$ , let  $T_v$  be the subtree of  $T$  rooted at  $v$ ,  $S_v$  be the set of supply vertices in  $T_v$ , and  $C_v$  be the set of the children of  $v$  (see Fig. 2). Note that  $C_v = \emptyset$  if  $v$  is a leaf.

For a vertex  $v \in V$  and for a supply rate  $r \geq 0$ , we consider the instance of the partition problem in which the graph is  $T_v$ , the supply vertex set is  $S_v = S \cap V(T_v)$ , the demand vertex set is  $D \cap V(T_v)$ , the supply value of  $x \in S_v$  is



**Fig. 2.** A subtree  $T_v$  rooted at  $v$ .

$s(x)$ , and the demand value of  $x \in D \cap V(T_v)$  is  $r \cdot d(x)$ . For a supply vertex  $u \in S_v$ , we define  $f_v^u(r)$  as the maximum of  $s(u) - \sum_{x \in D \cap V_1} r \cdot d(x)$  among all feasible partitions  $(V_1, V_2, \dots, V_{|S_v|})$  of  $V(T_v)$  such that  $u, v \in V_1$ , that is,

$$f_v^u(r) := \max \left\{ s(u) - \sum_{x \in D \cap V_1} r \cdot d(x) \right. \\ \left. \mid (V_1, V_2, \dots, V_{|S_v|}) \text{ is a feasible partition of } V(T_v), u, v \in V_1 \right\}.$$

Define  $f_v^u(r) = -\infty$  if  $T_v$  has no feasible partition with  $u, v \in V_1$ . Intuitively,  $f_v^u(r)$  is the maximum amount of flow that can be delivered from  $u$  to the outside of  $T_v$  through  $v$ . We regard  $f_v^u(r)$  as a function of  $r$ , and call it the *surplus function*.

For  $v \in V$ , define  $r_v^{\max}$  as the maximum value of the supply rate such that  $T_v$  has a feasible partition, that is,

$$r_v^{\max} := \max \{ r \mid f_v^u(r) \geq 0 (\exists u \in S_v) \}.$$

For convenience, define  $r_v^{\max} = 0$  if  $T_v$  does not have a supply vertex. Note that  $r_{v_{\text{root}}}^{\max}$  is the maximum supply rate of  $T$ .

For a vertex  $v \in V$  and for a supply rate  $r \geq 0$ , we define the *deficit function*  $g_v(r)$  as follows. We add a dummy supply vertex  $u_{\text{dum}}$  with  $s(u_{\text{dum}}) = +\infty$  to  $T_v$  as a child of  $v$ , and connect  $v$  and  $u_{\text{dum}}$  by a dummy edge with  $c(v, u_{\text{dum}}) = +\infty$ . Let  $T'_v$  be the obtained tree with  $|S_v| + 1$  supply vertices, and consider the instance of the partition problem in  $T'_v$  with respect to  $r \cdot d$ . Note that if  $T'_v$  has a feasible partition, then it consists of  $|S_v| + 1$  vertex subsets. We define  $g_v(r)$  as the maximum of  $-\sum_{x \in D \cap V_0} r \cdot d(x)$  among all feasible partitions  $(V_0, V_1, \dots, V_{|S_v|})$  of  $V(T'_v)$  such that  $u_{\text{dum}} \in V_0$ , that is,

$$g_v(r) := \max \left\{ - \sum_{x \in D \cap V_0} r \cdot d(x) \right. \\ \left. \mid (V_0, V_1, \dots, V_{|S_v|}) \text{ is a feasible partition of } V(T'_v), u_{\text{dum}} \in V_0 \right\}.$$

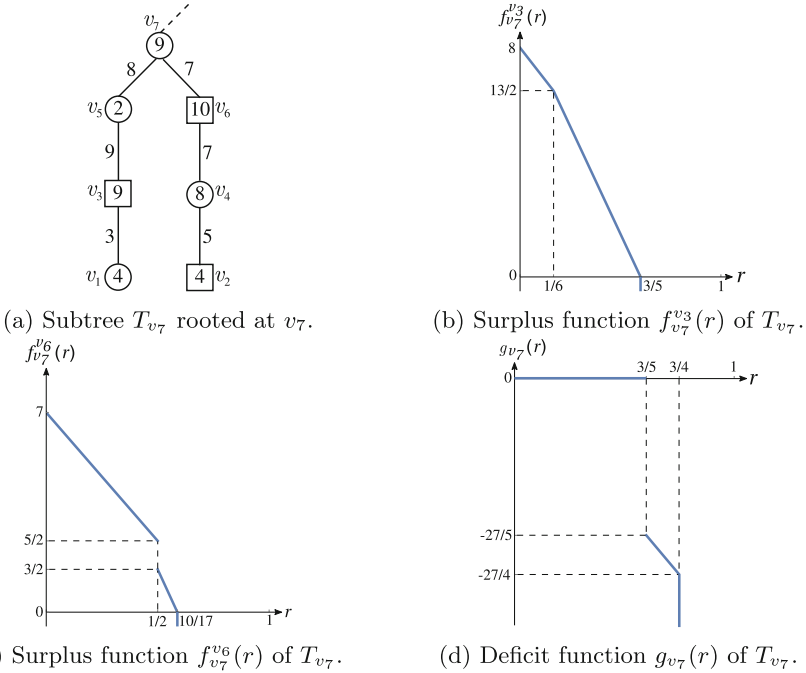
Define  $g_v(r) = -\infty$  if  $T'_v$  has no feasible partition. Note that  $D \cap V_0 = V_0 \setminus \{u_{\text{dum}}\}$ . Intuitively,  $-g_v(r)$  is the minimum amount of flow that has to enter  $T_v$  from the outside through  $v$  to fulfill the demands in  $T_v$ .

We now observe some properties of the surplus function  $f_v^u(r)$  and the deficit function  $g_v(r)$ . By the definitions, they are monotonically decreasing piecewise linear functions of  $r$  and can take the value  $-\infty$ . One can see that  $g_v(r) \leq 0$ , and the equality holds if and only if  $r \leq r_v^{\text{max}}$ . One can also see that  $f_v^u(r)$  can take nonnegative values or  $-\infty$ . In order to deal with this property of  $f_v^u(r)$ , for  $x \in \mathbb{R}$ , we use the notation  $(x)^+$  defined by

$$(x)^+ := \begin{cases} x & \text{if } x \geq 0, \\ -\infty & \text{otherwise.} \end{cases}$$

Similarly, for a function  $f : \mathbb{R}_+ \rightarrow \mathbb{R} \cup \{-\infty\}$ , we define  $f^+ : \mathbb{R}_+ \rightarrow \mathbb{R} \cup \{-\infty\}$  by  $f^+(x) = (f(x))^+$  for  $x \in \mathbb{R}_+$ . Here,  $\mathbb{R}_+$  is the set of all nonnegative real numbers.

*Example 1.* Figure 3a shows a subtree  $T_{v_7}$  rooted at  $v_7$ , where a supply vertex is represented as a square with a supply value, a demand vertex is represented as a circle with a demand value, and an edge is represented as a line with an edge capacity. Subtree  $T_{v_7}$  does not have a feasible partition when the supply



**Fig. 3.** An example of the surplus functions and the deficit function of  $T_{v_7}$ .



rate is  $r > \frac{3}{5}$ , but it has a feasible partition that consists of  $V_1 = \{v_1, v_3, v_5, v_7\}$ ,  $V_2 = \{v_4, v_6\}$ , and  $V_3 = \{v_2\}$  when the supply rate is  $0 \leq r \leq \frac{3}{5}$ . Furthermore, one can see that  $V_1 = \{v_4, v_6, v_7\}$ ,  $V_2 = \{v_1, v_3, v_5\}$ , and  $V_3 = \{v_2\}$  form a feasible partition if  $0 \leq r \leq \frac{10}{17}$ , and  $V_1 = \{v_6, v_7\}$ ,  $V_2 = \{v_1, v_3, v_5\}$ , and  $V_3 = \{v_2, v_4\}$  form a feasible partition if  $0 \leq r \leq \frac{1}{2}$ . This shows that  $T_{v_7}$  has multiple feasible partitions for some supply rate  $r$ . The surplus functions  $f_{v_7}^{v_3}(r)$  and  $f_{v_7}^{v_6}(r)$  of  $T_{v_7}$  are expressed as functions of  $r$  as shown in Figs. 3b and c, respectively. Note that  $f_{v_7}^{v_3}(r) \leq f_{v_7}^{v_6}(r)$  if  $\frac{1}{3} \leq r \leq \frac{1}{2}$  and  $f_{v_7}^{v_3}(r) \geq f_{v_7}^{v_6}(r)$  otherwise. One can see that  $r_{v_7}^{\max} = \frac{3}{5}$ . The deficit function  $g_{v_7}(r)$  of  $T_{v_7}$  is shown in Fig. 3d. Note that  $g_{v_7}(r) = 0$  for  $r \leq r_{v_7}^{\max} = \frac{3}{5}$ .

### 3 Algorithm for the Maximum Supply Rate Problem

In this section, we give a strongly polynomial time algorithm for the maximum supply rate problem on trees. We first give an outline of the algorithm in Sect. 3.1, then describe the algorithm in detail in Sect. 3.2.

#### 3.1 Outline

In a similar way to the algorithm for the partition problem on parametric power supply networks [5, 6], our algorithm is based on dynamic programming. Our algorithm computes the surplus function  $f_v^u(r)$  ( $u \in S_v$ ), the deficit function  $g_v(r)$ , and the maximum value of the supply rate  $r_v^{\max}$  for each vertex  $v$  of  $T$  from leaves to the root. After finishing the computation, the algorithm outputs  $r_{v_{\text{root}}}^{\max}$  as the solution  $r^*$  of the maximum supply rate problem.

It will be natural to consider the algorithm that computes  $f_v(r) := \max\{f_v^u(r) \mid u \in S_v\}$  instead of  $f_v^u(r)$  ( $u \in S_v$ ). Indeed, the function  $f_v(r)$  for each  $v \in V$  can be computed from leaves to the root by dynamic programming as well. However, since the updating formula of  $f_v(r)$  is complicated, it is hard to bound the size of the representation of  $f_v(r)$ , which makes the analysis of the running time complicated. A key idea in our algorithm is to consider multiple surplus functions  $f_v^u(r)$  instead of  $f_v(r)$ . Although it looks inefficient to increase the number of functions, an advantage of our algorithm is that the updating formula of the surplus functions is quite simple. In particular, we do not have to compute the maximum of two or more functions in our algorithm. This enables us to show that our algorithm runs in strongly polynomial time.

To analyze the sizes of the representations of the surplus functions and the deficit functions, we define the *number of intervals* of a piecewise linear function as follows. For a monotonically decreasing piecewise linear function  $f : \mathbb{R}_+ \rightarrow \mathbb{R} \cup \{-\infty\}$ , define the number of intervals of  $f$  as the minimum integer  $k$  such that  $\mathbb{R}_+$  can be partitioned into intervals  $I_1, \dots, I_{k+1}$ , where  $f$  is linear on each  $I_j$  ( $j = 1, \dots, k$ ) and  $f(r) = -\infty$  for  $r \in I_{k+1}$ . Note that  $I_{k+1}$  might be the empty set. The following observations play an important role in the analysis of our algorithm.

**Observation 2.** *If  $f : \mathbb{R}_+ \rightarrow \mathbb{R} \cup \{-\infty\}$  is a monotonically decreasing piecewise linear function whose number of intervals is at most  $k$ , then the number of intervals of  $f^+$  is at most  $k$ .*

**Observation 3.** *If  $f_1, f_2 : \mathbb{R}_+ \rightarrow \mathbb{R} \cup \{-\infty\}$  are monotonically decreasing piecewise linear functions whose numbers of intervals are at most  $k_1$  and  $k_2$ , respectively, then the number of intervals of  $f_1 + f_2$  is at most  $k_1 + k_2$ .*

We note that, for monotonically decreasing piecewise linear functions  $f_1$  and  $f_2$  whose numbers of intervals are at most  $k_1$  and  $k_2$ , the number of intervals of  $\max\{f_1, f_2\}$  is not necessarily bounded by  $k_1 + k_2$ . Since the updating formula of  $f_v(r)$  requires the max operation, it is hard to bound the number of intervals of  $f_v(r)$ . We emphasize again that the updating formula of  $f_v^u(r)$  does not require the max operation. We only use the operations as in Observations 2 and 3, which is essential to design a strongly polynomial-time algorithm.

### 3.2 Algorithm Description

As mentioned above, our algorithm computes the surplus functions, the deficit function, and the maximum value of the supply rate for each vertex from leaves to the root. The full description of our algorithm is as follows.

**Step 1.** Compute the surplus function and the deficit function for every leaf  $v$  of  $T$ . If  $v$  is a supply vertex, then we have  $f_v^v(r) = s(v)$  and  $g_v(r) = 0$  for every  $r$ , and hence  $r_v^{\max} = \infty$ . If  $v$  is a demand vertex, then we have  $g_v(r) = -r \cdot d(v)$  for every  $r$ , and hence  $r_v^{\max}(r) = 0$ .

**Step 2.** Compute the surplus functions, the deficit function, and  $r_v^{\max}$  of each internal vertex  $v$  from the leaves to the root by using those of its children as follows.

**Step 2-1.** For each  $w \in C_v$ , define the new deficit function  $g'_w(r)$  as

$$g'_w(r) = \begin{cases} g_w(r) & \text{if } -g_w(r) \leq c(v, w), \\ -\infty & \text{otherwise.} \end{cases}$$

Recall that  $g_w(r) \leq 0$ . This modification means that if the flow value that has to enter  $T_w$  from the outside is larger than the edge capacity of  $(v, w)$ , i.e.,  $-g_w(r) > c(v, w)$ , then there is no feasible partition in  $T_v$ , which is represented by  $g'_w(r) = -\infty$ .

**Step 2-2.** If  $v$  is a supply vertex, then execute the following (a)–(c).

(a) Compute  $f_v^v(r)$  by

$$f_v^v(r) = \left( s(v) + \sum_{w \in C_v} g'_w(r) \right)^+.$$

This equation means that  $s(v)$  is reduced by  $-g'_w(r)$  for each  $w \in C_v$ . If  $f_v^v(r) < 0$ , then  $f_v^v(r) = -\infty$  since there is no feasible partition in  $T_v$ . Since  $v$  cannot receive a flow from any other supply node, we have  $f_v^u(r) = -\infty$  for each  $u \in S_v \setminus \{v\}$ .

(b) Compute  $r_v^{\max}$  by

$$r_v^{\max} = \max\{r \mid f_v^v(r) \geq 0\}.$$

(c) Compute  $g_v(r)$  by

$$g_v(r) = \begin{cases} 0 & \text{if } r \leq r_v^{\max}, \\ -\infty & \text{otherwise.} \end{cases}$$

If  $r \leq r_v^{\max}$ , then we have  $g_v(r) = 0$  since there is at least one feasible partition in  $T_v$ . Otherwise, we have  $g_v(r) = -\infty$  since there is no feasible partition even if we add a dummy node as in the definition of  $g_v(r)$ .

**Step 2-3.** If  $v$  is a demand vertex, then execute the following (a)–(c).

(a) For each supply vertex  $u \in S_v$ , we compute  $f_v^u(r)$  by executing the following steps.

- Choose  $x \in C_v$  such that the supply vertex  $u$  is contained in the subtree  $T_x$ , i.e.,  $u \in S_x$ .
- Define  $f_x^{f'u}(r)$  as

$$f_x^{f'u}(r) = \min\{f_x^u(r), c(v, x)\}.$$

This modification means that if the edge capacity of  $(v, x)$  is less than the surplus function, i.e.,  $f_x^u(r) > c(v, x)$ , then  $u$  can deliver a flow whose value is at most  $c(v, x)$  to  $v$  through  $x$ , which is represented by  $f_x^{f'u}(r) = c(v, x)$ .

- Compute  $f_v^u(r)$  by

$$f_v^u(r) = \left( f_x^{f'u}(r) - r \cdot d(v) + \sum_{w \in C_v \setminus \{x\}} g'_w(r) \right)^+.$$

This equation means that the flow value  $f_x^{f'u}(r)$  delivered by  $u$  is reduced by  $r \cdot d(v)$  and by  $-g'_w(r)$  for each  $w \in C_v \setminus \{x\}$ . If  $f_v^u(r) < 0$ , then  $f_v^u(r) = -\infty$  since there is no feasible partition in  $T_v$ .

(b) Compute  $r_v^{\max}$  by

$$r_v^{\max} = \max\{r \mid f_v^u(r) \geq 0 \ (\exists u \in S_v)\}.$$

(c) Compute the deficit  $g_v(r)$  by

$$g_v(r) = \begin{cases} 0 & \text{if } r \leq r_v^{\max}, \\ -r \cdot d(v) + \sum_{w \in C_v} g'_w(r) & \text{otherwise.} \end{cases}$$

If  $r \leq r_v^{\max}$ , then we have  $g_v(r) = 0$  since there is at least one feasible partition in  $T_v$ . Otherwise, the flow value that  $v$  has to receive increases of  $r \cdot d(v)$ .

**Step 3.** Output  $r^* := r_{v_{\text{root}}}^{\max}$  as the optimal solution.

The correctness of the algorithm is trivial by the definitions of  $f_v^u(r)$ ,  $g_v(r)$ , and  $r_v^{\max}$ .

## 4 Running Time

In this section, we analyze the running time of our algorithm in Sect. 3.2 and give a proof of Theorem 1. Obviously, the surplus functions and the deficit functions for all leaves can be computed in time  $\mathcal{O}(n)$  as in Step 1. In order to analyze the running time of Step 2, it is important to estimate the sizes of the representations of the surplus functions and the deficit functions. Since the surplus functions and the deficit functions are piecewise linear functions, it suffices to bound the number of intervals of these functions. Let  $|T_v|$  be the number of vertices in the subtree  $T_v$ . We now show the following lemma.

**Lemma 1.** *For  $v \in V$ , the number of intervals of  $f_v^u(r)$  is at most  $|T_v|$  for each  $u \in S_v$ , and that of  $g_v(r)$  is also at most  $|T_v|$ .*

*Proof.* We show the lemma by induction on  $|T_v|$ .

- Suppose that  $|T_v| = 1$ , i.e.,  $v$  is a leaf.

As in Step 1, if  $v$  is a supply vertex, then we have  $f_v^v(r) = s(v)$  and  $g_v(r) = 0$ . If  $v$  is a demand vertex, then we have  $g_v(r) = -r \cdot d(v)$ . Therefore, the number of intervals of each function is exactly one, which is equal to  $|T_v|$ .

- Suppose that  $|T_v| \geq 2$ , i.e.,  $v$  is not a leaf.

For each  $w \in C_v$ , the induction hypothesis shows that the numbers of intervals of  $f_w^u(r)$  ( $u \in S_w$ ) and that of  $g_w(r)$  are at most  $|T_w|$ . Then, we consider the numbers of intervals of  $f_v^u(r)$  and that of  $g_v(r)$ . Note that  $|T_v| = 1 + \sum_{w \in C_v} |T_w|$ . Note also that the number of intervals of  $g'_w(r)$  is at most  $|T_w|$  since Step 2-1 of our algorithm defines  $g'_w(r)$  from  $g_w(r)$  by just shifting the minimum value of  $r$  with  $g_w(r) = -\infty$ . We consider the following two cases separately.

- Suppose that  $v$  is a supply vertex.

In this case, we compute  $f_v^u(r)$  for each  $u \in S_v$  and  $g_v(r)$  in Step 2-2. Since adding the constant value  $s(v)$  does not affect the intervals of the piecewise linear function, by Observations 2 and 3, the number of intervals of  $f_v^v(r)$  is at most

$$\sum_{w \in C_v} |T_w| \leq |T_v|.$$

For  $u \in S_v \setminus \{v\}$ , the number of intervals of  $f_v^u(r)$  is zero since  $f_v^u(r) = -\infty$ . The number of intervals of  $g_v(r)$  is at most one since  $g_v(r)$  can take either 0 or  $-\infty$ .

- Suppose that  $v$  is a demand vertex.

In this case, we compute  $f_v^u(r)$  for each  $u \in S_v$  and  $g_v(r)$  in Step 2-3. Recall that  $x \in C_v$  satisfies that  $u \in S_x$ . In (a), the number of intervals of  $f_x^u(r)$  is at most that of  $f_x^u(r)$  plus one since we take the minimum of  $f_x^u(r)$  and the edge capacity  $c(v, x)$ . This implies that the number of intervals of  $f_x^u(r)$  is at most  $|T_x| + 1$ . We are now ready to consider  $f_v^u(r)$ . Since adding the linear function  $-r \cdot d(v)$  does not affect the intervals of

the piecewise linear function, by Observations 2 and 3, the number of intervals of  $f_v^u(r)$  is at most

$$1 + \sum_{w \in C_v} |T_w| = |T_v|.$$

Similarly, since adding the linear function  $-r \cdot d(v)$  does not affect the intervals of the piecewise linear function, by Observation 3, the number of intervals of  $g_v(r)$  is at most

$$1 + \sum_{w \in C_v} |T_w| = |T_v|.$$

These complete the proof of Lemma 1.  $\square$

By Lemma 1, since each of  $f_v^u(r)$  ( $u \in S_v$ ) and  $g_v(r)$  can be represented by using  $\mathcal{O}(n)$  linear functions, it can be computed in time  $\mathcal{O}(n)$  in Step 2. Since we compute  $|S_v| + 1 = \mathcal{O}(n)$  functions for each  $v \in V$ , one iteration of Step 2 is executed in time  $\mathcal{O}(n^2)$ . Since the number of internal vertices is  $\mathcal{O}(n)$ , the total running time is at most  $\mathcal{O}(n^3)$ . This completes the proof of Theorem 1.

We note that one can easily obtain a feasible partition of a tree with respect to the maximum supply rate  $r^*$  by modifying our algorithm slightly.

## 5 Extension to the Maximum Supply Parameter Problem

We have already seen that there exists a strongly polynomial-time algorithm for the maximum supply rate problem on trees. In this section, we extend our algorithm to the problem on parametric networks (see e.g. [5, 6, 10]). A *parametric tree* is a tree  $T = (V, E)$ , in which each vertex is either a supply vertex or a demand vertex, such that each of the supply values and the demand values is represented as a piecewise linear function of a parameter. More precisely, suppose that

- each supply vertex  $v \in S$  is assigned a nonnegative monotonically decreasing piecewise linear function  $s_v : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ , and
- each demand vertex  $v \in D$  is assigned a nonnegative monotonically increasing piecewise linear function  $d_v : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ .

The supply value  $s_v(r)$  and the demand value  $d_v(r)$  are determined if we fix a parameter  $r$ . In the *maximum supply parameter problem*, the input is a parametric tree and the objective is to find the maximum value of the parameter  $r$  such that the tree has a feasible partition with respect to  $s_v(r)$  and  $d_v(r)$ . Note that the monotonicity of the functions implies that if there exists a feasible partition for some parameter  $r_0$ , then there exists a feasible partition for any  $r \leq r_0$ . Note also that the maximum supply rate problem is a special case of the maximum supply parameter problem in which the supply values are represented by constant functions and the demand values are represented by linear functions.

In order to solve the maximum supply parameter problem on parametric trees, we apply the same algorithm as in Sect. 3.2 in which  $s(v)$  and  $r \cdot d(v)$  are replaced with  $s_v(r)$  and  $d_v(r)$ , respectively. In what follows, we show that this algorithm solves the maximum supply parameter problem in polynomial time. We basically use the same notation as in Sect. 3.2, and also use the following notation. For a monotonically decreasing piecewise linear function  $f : \mathbb{R}_+ \rightarrow \mathbb{R}_+ \cup \{-\infty\}$ , let  $I(f)$  denote the number of intervals of  $f$ . In order to make it clear that  $f$  is a function of  $r$ ,  $I(f)$  is also denoted by  $I(f(r))$  in this paper. For each vertex  $v \in V$ , define  $K_v$  as the sum of the numbers of intervals of the supply/demand functions in  $T_v$ , that is,

$$K_v := \sum_{v \in S_v} I(s_v(r)) + \sum_{v \in D \cap V(T_v)} I(-d_v(r)).$$

In the same way as Lemma 1, we give an upper bound on the numbers of intervals of the surplus functions and the deficit functions as follows.

**Lemma 2.** *For  $v \in V$ , we have  $I(f_v^u(r)) \leq 2K_v$  for each  $u \in S_v$  and  $I(g_v(r)) \leq 2K_v$ .*

*Proof.* We show the lemma by induction on  $K_v$ .

– Suppose that  $v$  is a leaf.

As in Step 1, if  $v$  is a supply vertex, then we have  $f_v^u(r) = s_v(r)$  and  $g_v(r) = 0$ . If  $v$  is a demand vertex, then we have  $g_v(r) = -d_v(r)$ . Therefore, the number of intervals of each function is at most  $K_v$ , which is no more than  $2K_v$ .

– Suppose that  $v$  is not a leaf.

For each  $w \in C_v$ , the induction hypothesis shows that  $I(f_w^u(r)) \leq 2K_w$  ( $u \in S_w$ ) and  $I(g_w(r)) \leq 2K_w$ . Then, we consider  $I(f_v^u(r))$  and  $I(g_v(r))$ . Note that  $I(g'_w(r)) \leq 2K_w$  holds since Step 2-1 of our algorithm defines  $g'_w(r)$  from  $g_w(r)$  by just shifting the minimum value of  $r$  with  $g_w(r) = -\infty$ . We consider the following two cases separately.

• Suppose that  $v$  is a supply vertex.

In this case, we compute  $f_v^u(r)$  for each  $u \in S_v$  and  $g_v(r)$  in Step 2-2. Since  $f_v^u(r)$  is defined by

$$f_v^u(r) = \left( s_v(r) + \sum_{w \in C_v} g'_w(r) \right)^+,$$

by Observations 2 and 3, we have

$$I(f_v^u(r)) \leq I(s_v(r)) + \sum_{w \in C_v} I(g'_w(r)) \leq I(s_v(r)) + \sum_{w \in C_v} 2K_w \leq 2K_v.$$

For  $u \in S_v \setminus \{v\}$ , we have  $I(f_v^u(r)) = 0$  since  $f_v^u(r) = -\infty$ . We can also see that  $I(g_v(r)) \leq 1$  since  $g_v(r)$  can take either 0 or  $-\infty$ .

- Suppose that  $v$  is a demand vertex.

In this case, we compute  $f_v^u(r)$  for each  $u \in S_v$  and  $g_v(r)$  in Step 2-3. Recall that  $x \in C_v$  satisfies that  $u \in S_x$ . Since  $f_x^u(r)$  is defined as the minimum of  $f_x^u(r)$  and the edge capacity  $c(v, x)$ , we have  $I(f_x^u(r)) \leq 1 + I(f_x^u(r)) \leq 1 + 2K_x$ . Since  $f_v^u(r)$  is defined by

$$f_v^u(r) = \left( f_x^u(r) - d_v(r) + \sum_{w \in C_v \setminus \{x\}} g'_w(r) \right)^+,$$

by Observations 2 and 3, we have

$$\begin{aligned} I(f_v^u(r)) &\leq I(f_x^u(r)) + I(-d_v(r)) + \sum_{w \in C_v \setminus \{x\}} I(g'_w(r)) \\ &\leq 1 + 2K_x + I(-d_v(r)) + \sum_{w \in C_v \setminus \{x\}} 2K_w \leq 2K_v. \end{aligned}$$

Note that we use  $I(-d_v(r)) \geq 1$  in the last inequality. Similarly, since  $g_v(r)$  is defined by

$$g_v(r) = \begin{cases} 0 & \text{if } r \leq r_v^{\max}, \\ -d_v(r) + \sum_{w \in C_v} g'_w(r) & \text{otherwise,} \end{cases}$$

by Observation 3, we have

$$I(g_v(r)) \leq 1 + I(-d_v(r)) + \sum_{w \in C_v} 2K_w \leq 2K_v.$$

These complete the proof of Lemma 2.  $\square$

By using Lemma 2, we show that our algorithm can solve the maximum supply parameter problem in polynomial time.

**Theorem 4.** *The maximum supply parameter problem on parametric trees can be solved in time  $\mathcal{O}(n^2K)$ , where  $n$  is the number of vertices of the parametric tree and  $K := K_{v_{\text{root}}}$  is the sum of the numbers of intervals of all the functions representing the demand values and the supply values.*

*Proof.* We apply the same algorithm as in Sect. 3.2 in which  $s(v)$  and  $r \cdot d(v)$  are replaced with  $s_v(r)$  and  $d_v(r)$ , respectively. By Lemma 2, the number of intervals of each surplus/deficit function is  $\mathcal{O}(K)$ , and hence each surplus/deficit function can be computed in time  $\mathcal{O}(K)$ . Since we compute  $|S_v| + 1 = \mathcal{O}(n)$  functions for each  $v \in V$ , one iteration of Step 2 is executed in time  $\mathcal{O}(nK)$ . Since the number of internal vertices is  $\mathcal{O}(n)$ , the total running time is at most  $\mathcal{O}(n^2K)$ .  $\square$

Note that the input size of the maximum supply parameter problem is  $\Theta(n + K)$ , and hence the above running time is bounded by a polynomial in the input size.

As a corollary of Theorem 4, we give a polynomial-time algorithm for another variant of the maximum supply rate problem. When we consider practical applications, it is also natural to increase the supply values instead of decreasing the demand values. We consider scaling the supply values uniformly by scale factor  $q$  so that the obtained graph has a feasible partition. The *minimum increment rate* is defined as the minimum value of  $q$  such that the obtained graph has a feasible partition with respect to  $q \cdot s$ , and the *minimum increment rate problem* is the problem that finds the minimum increment rate  $q^*$ . We now show that the minimum increment rate problem on trees can be solved in polynomial time.

**Corollary 1.** *The minimum increment rate problem on trees can be solved in time  $\mathcal{O}(n^3)$ , where  $n$  is the number of vertices of the tree.*

*Proof.* Let  $M$  be a sufficiently large number, e.g.  $M := (\sum_{v \in D} d(v)) / \min\{s(v) \mid v \in S, s(v) > 0\}$ . If there is no feasible partition for increment ratio  $q = M$ , then we can immediately conclude that the minimum increment rate problem has no feasible solution. Otherwise, consider the maximum supply parameter problem, in which the demand value  $d'(v)$  is equal to  $d(v)$ , and the supply value  $s'(v)$  is the piecewise linear function  $\max\{M \cdot s(v) - r \cdot s(v), 0\}$  of the parameter  $r$ . Since  $K = \mathcal{O}(n)$ , we can find the optimal solution  $r^*$  of the maximum supply parameter problem in time  $\mathcal{O}(n^3)$  by Theorem 4. Then, we have that  $M - r^*$  is the minimum increment rate, which shows that the minimum increment rate problem on trees can be solved in time  $\mathcal{O}(n^3)$ .  $\square$

## References

1. Ito, T., Zhou, X., Nishizeki, T.: Partitioning trees of supply and demand. *Int. J. Found. Comput. Sci.* **16**(4), 803–827 (2005)
2. Kawabata, M., Nishizeki, T.: Partitioning trees with supply, demand and edge-capacity. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **96**(6), 1036–1043 (2013)
3. Dunlop, A.E., Kernighan, B.W.: A procedure for placement of standard cell VLSI circuits. *IEEE Trans. Comput.-Aided Des.* **4**(1), 92–98 (1985)
4. Hatzigargyriou, N., Asano, H., Iravani, R., Marnay, C.: Microgrids. *IEEE Power Energy Mag.* **5**(4), 78–94 (2007)
5. Morishita, S., Nishizeki, T.: Parametric power supply networks. *J. Comb. Optim.* **29**(1), 1–15 (2015)
6. Maruta, S., Nishizeki, T.: Parametric power supply networks with edge-capacities. *IPSJ SIG Tech. Rep.* **2014-AL-147**(3), 1–8 (2014)
7. Ito, T., Demaine, E.D., Zhou, X., Nishizeki, T.: Approximability of partitioning graphs with supply and demand. *J. Discrete Algorithms* **6**(4), 627–650 (2008)
8. Ito, T., Zhou, X., Nishizeki, T.: Partitioning graphs of supply and demand. *Discrete Appl. Math.* **157**(12), 2620–2633 (2009)
9. Jovanovic, R., Bousseth, A., Vo, S.: A heuristic method for solving the problem of partitioning graphs with supply and demand. *Ann. Oper. Res.* **235**(1), 371–393 (2015)
10. Minieka, E.: Parametric network flows. *Oper. Res.* **20**(6), 1162–1170 (1972)





# The Maximum Distance- $d$ Independent Set Problem on Unit Disk Graphs

Sangram K. Jena<sup>1</sup>, Ramesh K. Jallu<sup>2</sup>, Gautam K. Das<sup>1</sup>(✉),  
and Subhas C. Nandy<sup>2</sup>

<sup>1</sup> Indian Institute of Technology, Guwahati, India  
{sangram,gkd}@iitg.ernet.in

<sup>2</sup> Indian Statistical Institute, Kolkata, India  
jalluramesh@gmail.com, nandysc@isical.ac.in

**Abstract.** In this article, we study the maximum distance- $d$  independent set problem, a variant of the maximum independent set problem, on unit disk graphs. We first show that the problem is NP-hard. Next, we propose a polynomial-time constant-factor approximation algorithm and a PTAS for the problem.

**Keywords:** Independent set · Distance- $d$  independent set · PTAS  
Unit disk graph

## 1 Introduction

The *independent set problem* is one of the best known classical combinatorial optimization problems in graph theory due to its many important applications, including but not limited to networks, map labeling, computer vision, coding theory, scheduling, clustering. Given an unweighted graph  $G = (V, E)$ , a non-empty subset of pairwise non-adjacent vertices of  $G$  is known as an *independent set* of  $G$ . Any singleton set is a trivial independent set of  $G$ . The maximum independent set problem asks to find an independent set of maximum possible size in a given unweighted graph  $G$ , and such a set is called as *maximum independent set* (MIS) of  $G$ . For an integer  $d \geq 2$ , a *distance- $d$  independent set* (DdIS) of an unweighted graph  $G = (V, E)$  is an independent set  $I$  of  $V$  such that the shortest path distance (i.e., the number edges on a shortest path) between any pair of vertices in  $I$  is at least  $d$ . For a given unweighted graph  $G$ , the objective of the maximum distance- $d$  independent set problem is to find a DdIS of maximum cardinality in  $G$ . A DdIS of maximum possible size is called as *maximum distance- $d$  independent set* (MDdIS). Observe that the MDdIS problem is a generalization of the MIS problem and in fact for  $d = 2$ , the MDdIS problem and MIS problem are the same.

Given a set  $P = \{p_1, p_2, \dots, p_n\}$  of points in the plane, a *unit disk graph* (UDG) corresponding to the point set  $P$  is a simple graph  $G = (V, E)$  satisfying  $V = P$ , and  $E = \{(p_i, p_j) \mid d(p_i, p_j) \leq 1\}$ , where  $d(p_i, p_j)$  denotes the Euclidean

distance between  $p_i$  and  $p_j$ . In other words, a unit disk graph is an intersection graph of disks of unit diameter centered at the points in  $P$ .

An algorithm for a minimization (resp. maximization) problem is said to be a  $\rho$ -factor approximation algorithm if for every instance of the problem the algorithm produces a feasible solution whose value is within a factor of  $\rho$  (resp.  $\frac{1}{\rho}$ ) of the optimal solution value and runs in polynomial-time of the input size. Here,  $\rho$  is called the *approximation factor* or *approximation ratio* of the algorithm and the optimization problem is said to have a  $\rho$ -factor approximation algorithm. A *polynomial-time approximation scheme* (PTAS) for an optimization problem is a collection of algorithms  $\{\mathcal{A}_\epsilon\}$  such that for a given  $\epsilon > 0$ ,  $\mathcal{A}_\epsilon$  is a  $(1 + \epsilon)$ -factor approximation algorithm in case of minimization problem ( $(1 - \epsilon)$  in case of maximization). The running time of  $\mathcal{A}_\epsilon$  is required to be polynomial in the size of the problem depending on  $\epsilon$ .

## 2 Related Work

The MIS problem is known to be NP-hard for general graphs [12] and also many subclasses of planar graphs, namely planar graphs of maximum degree 3 [11], planar graphs of large girth [23], cubic planar graphs [13], triangle-free graphs [26],  $K_{1,4}$ -free graphs [21], etc. In general, the MIS problem cannot be approximated within a constant factor unless  $P = NP$  [2]. However, the problem is polynomially solvable for bipartite graphs, outer-planar graphs, perfect graphs, claw-free graphs, chordal graphs, etc. [14, 17]. The MIS problem is well studied on UDGs too and is shown to be NP-hard [5]. Unlike in general graphs, the problem admits approximation algorithms [6, 15, 18–20, 24] and approximation schemes [6, 7, 18, 25].

The MD $d$ IS problem, for any fixed  $d \geq 3$ , is known to be NP-hard for bipartite graphs [4] and planar bipartite graphs of maximum degree 3 [8]. It is also known that getting an  $n^{\frac{1}{2}-\epsilon}$ -factor approximation result, for any  $\epsilon > 0$ , on bipartite graphs is NP-hard (this result also holds for chordal graphs when  $d \geq 3$  is an odd number) [8]. The problem is polynomially solvable for some intersection graphs, such as interval graphs, trapezoid graphs, and circular arc graphs [1]. If the input graph is restricted to be a chordal graph, then the problem is solvable in polynomial time for any even  $d \geq 2$ ; on the other hand, the problem is NP-hard for any odd  $d \geq 3$  [8]. Eto et al. [9] studied the problem on  $r$ -regular graphs and planar graphs. The authors showed that for  $d \geq 3$  and  $r \geq 3$ , the MD $d$ IS problem on  $r$ -regular graphs is APX-hard, and proposed  $O(r^{d-1})$  and  $O(\frac{r^{d-2}}{d})$ -factor approximation algorithms. When  $d = r = 3$ , they enhanced their  $O(\frac{r^{d-2}}{d})$ -factor result to a 2-factor approximation result (recently, the approximation factor is improved to 1.875 [10]). Finally, they proposed a PTAS in case of planar graphs. Montealegre and Todinca studied the problem in graphs with few minimal separators [22].

## 2.1 Our Work

In this paper, we study the MDdIS problem on unit disk graphs and we call it *the geometric maximum distance- $d$  independent set* (GMDdIS) problem. We show that the decision version of the GMDdIS problem (for  $d \geq 3$ ) is NP-complete on unit disk graphs. We also propose a 4-factor approximation algorithm, and a PTAS for this problem.

## 2.2 Organization

The hardness of the GMDdIS problem is discussed in Sect. 3. The proposed 4-factor approximation algorithm and PTAS are explained in Sects. 4 and 5, respectively. Finally, Sect. 6 concludes the paper.

## 3 The GMDdIS Problem on Unit Disk Graphs

For an integer  $d \geq 3$ , we define the GMDdIS problem formally as follows:

*Given an unweighted unit disk graph  $G = (V, E)$  corresponding to a point set  $P = \{p_1, p_2, \dots, p_n\}$  in the plane, find a maximum cardinality subset  $I \subseteq V$ , such that for every pair of vertices  $p_i, p_j \in I$ , the length (number of edges) of the shortest path between  $p_i$  and  $p_j$  is at least  $d$ .*

For a fixed constant  $d \geq 3$ , the decision version of the GMDdIS problem is defined as follows:

### GEOMETRIC DISTANCE- $d$ INDEPENDENT SET (GDdIS) PROBLEM

**Input.** An unweighted unit disk graph  $G = (V, E)$  defined on a point set  $P$  and a positive integer  $k \leq |V|$ .

**Question.** Does there exist a distance- $d$  independent set of size at least  $k$  in  $G$ ?

We show that the GDdIS ( $d \geq 3$ ) problem belongs to the class of NP-complete problems by reducing the NP-complete problem *distance- $d$  independent set problem* ( $d \geq 3$ ) on planar bipartite graphs with girth<sup>1</sup> at least  $d$  and maximum degree 3, which is defined as follows:

### DISTANCE- $d$ INDEPENDENT SET ON PLANAR BIPARTITE GRAPHS

**Input.** An unweighted planar bipartite graph  $G = (V, E)$  with girth at least  $d$  and maximum vertex degree 3, and a positive integer  $k \leq |V|$ .

**Question.** Does there exist a distance- $d$  independent set of size at least  $k$  in  $G$ ?

In [8], it has been shown that the distance- $d$  independent set problem on planar bipartite graphs with maximum degree 3 is NP-complete by reducing the distance-2 independent set problem on planar cubic graphs. In fact, the reduced graph in their reduction has girth at least  $d$  and hence the distance- $d$  independent set problem on planar bipartite graphs with maximum degree 3 and girth at least  $d$  is NP-complete.

Our reduction is based on the concept of planar embedding of planar graphs. The following lemma is very useful in our reduction.

<sup>1</sup> The length of a smallest cycle.

**Lemma 1** [5]. *A planar graph  $G = (V, E)$  with maximum degree 4 can be embedded in the plane using  $O(|V|^2)$  area in such a way that its vertices are at integer coordinates and its edges are drawn using axis-parallel line segments at integer coordinates (i.e., edges lie on the lines  $x = i_1, i_2, \dots$  and/or  $y = j_1, j_2, \dots$ , where  $i_1, i_2, \dots, j_1, j_2, \dots$  are integers).*

This kind of embedding is known as *orthogonal drawing* of a graph. Biedl and Kant [3] gave a linear time algorithm that produces an orthogonal drawing of a given graph with the property that the number of bends along each edge is at most 2.

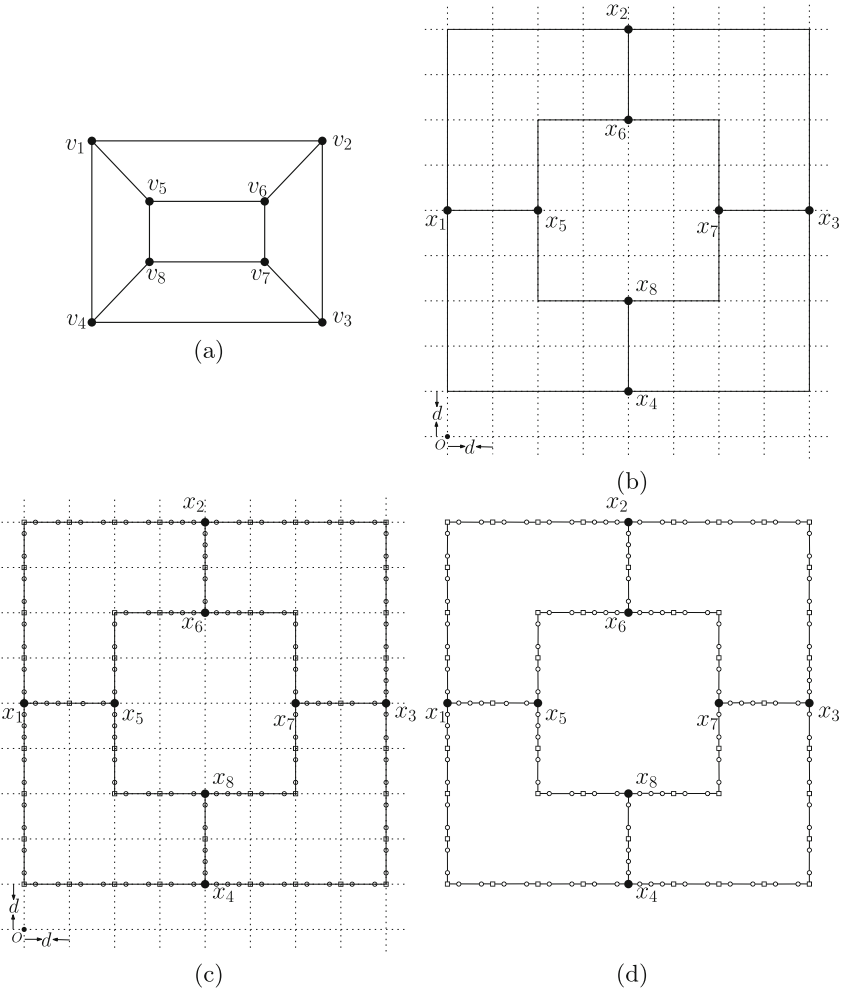
**Corollary 1.** *Let  $G = (V, E)$  be a planar bipartite graph with maximum degree 3 and girth at least  $d (d \geq 3)$ . The graph  $G$  can be embedded on a grid in the plane, whose each grid cell is of size  $d \times d$ , so that its vertices lie at points of the form  $(i \cdot d, j \cdot d)$  and its edges are drawn using a sequence of consecutive line segments drawn on the vertical lines of the form  $x = i \cdot d$  and/or horizontal lines of the form  $y = j \cdot d$ , for some integers  $i$  and  $j$  (see Fig. 1).*

Lemma 1 suggests that, any planar graph  $G$  of maximum degree 4 can be embedded on a grid in the plane so that the following holds.

1. Each vertex  $v_i$  of  $G$  is associated with a point with integer coordinates in the plane.
2. An edge of  $G$  is represented as a sequence of alternating horizontal and/or vertical line segments drawn on the grid lines. For example, see edges  $(v_1, v_4)$  or  $(v_2, v_6)$  in Fig. 1(a). The edge  $(v_1, v_4)$  is drawn as a sequence of four vertical line segments and four horizontal line segments in the embedding (see  $(x_1, x_4)$  in Fig. 1(b)). Similarly, the edge  $(v_2, v_6)$  is drawn as a sequence of two vertical line segments in the embedding.
3. No two sets of consecutive line segments corresponding to two distinct edges of  $G$  have a common point unless the edges are incident at a vertex in  $G$ .

Let  $G = (V, E)$  be an arbitrary instance of DdIS for planar bipartite graph having maximum degree three and girth at least  $d$ . Let  $V = \{v_1, v_2, \dots, v_n\}$  and  $E = \{e_1, e_2, \dots, e_m\}$ . We denote the shortest path distance between two vertices  $v_i$  and  $v_j$  in  $G$  by  $d_G(v_i, v_j)$ , and  $v_i, v_j$  are said to be distance- $d$  independent in  $G$  if  $d_G(v_i, v_j) \geq d$ .

We construct a graph  $G' = (V', E')$  by embedding  $G$  on a grid in which each cell is of size  $d \times d$  as described in Corollary 1. Let  $V' = \{x_1, x_2, \dots, x_n\}$  be the vertices in  $G'$  corresponding to  $v_1, v_2, \dots, v_n$  in  $G$ . The coordinate of each member in  $V'$  is of the form  $(d \cdot i, d \cdot j)$ , where  $i, j$  are integers, and shown using big dots in Fig. 1(c). Let  $\ell$  be the number of line segments used for drawing all the edges in  $G'$ . To make  $G'$  a UDG we introduce a set  $Y$  of extra points on the segments used to draw the edges of  $G'$ . Thus, the set of points in  $V'$  (hereafter denoted by  $X$ ) together with  $Y$  form a UDG  $G''$ . Let  $(x_i, x_j)$  be an edge in  $G'$  corresponding to the edge  $(v_i, v_j)$  in  $G$  and has  $\ell'$  segments. We introduce  $\ell' d$  points on the polyline denoting the edge  $(x_i, x_j)$  in such a way that (i) after



**Fig. 1.** (a) A planar bipartite graph  $G$  of maximum degree 3, (b) its embedding  $G'$  on a grid of cell size  $3 \times 3$ , (c) adding of extra points to  $G'$ , (d) the obtained UDG  $G''$ .

adding the extra points, the length of the path from  $x_i$  to  $x_j$  thus obtained is exactly  $\ell'd + 1$ , (ii) a point is placed at each of the co-ordinates of the form  $(d \cdot i, d \cdot j)$ , where  $i$  and  $j$  are integers (shown using small squares in Fig. 1(c)), (iii) the segment adjacent to the point  $x_i$  contains exactly  $d$  newly added points and other segments on the path from  $x_i$  to  $x_j$  have  $d - 1$  points (shown using small circles in Fig. 1(c)), and (iv) only consecutive points on the path  $x_i \rightsquigarrow x_j$  are within unit distance apart.

Now, we construct a UDG  $G'' = (V'', E'')$ , where  $V'' = X \cup Y$ , and  $E'' = \{(p_i, p_j) \mid p_i, p_j \in V'' \text{ and } d(p_i, p_j) \leq 1\}$ . Here  $|V''| = |X| + |Y| = n + ld$ , and  $|E''| = ld + m$ , where  $m$  is the number of edges in  $G$ . Thus,  $G''$  can be

constructed in polynomial time. We will use the term  $d$ -grid for a grid whose each cell is of size  $d \times d$ .

The notion of points and vertices of  $G''$  are used interchangeably in the rest of the paper.

**Lemma 2.** *Any DdIS of  $G''$  contains at most  $\ell$  points from  $Y$ .*

*Proof.* For each segment in the  $d$ -grid used to draw  $G'$ , the number of points of  $Y$  appearing on it is  $d$  or  $d - 1$ . Thus, each segment may contain at most one point from  $Y$  in the DdIS of  $G''$ . In particular, if two end-points of a segment  $\eta$  of the  $d$ -grid (that are vertices of  $G''$ ) are chosen in DdIS, then no point of  $Y$  lying on  $\eta$  will be chosen. Now, the result follows from the fact that  $\ell$  many segments of the  $d$ -grid are used to draw  $G'$ .  $\square$

**Lemma 3.**  *$G$  has a DdIS of cardinality at least  $k$  if and only if  $G''$  has a DdIS of cardinality at least  $k + \ell$ .*

*Proof (Necessity).* Let  $G$  have a DdIS  $D$  of size at least  $k$ . Let  $X' = \{x_i \in X \mid v_i \in D\}$ . Let  $G_{i,\alpha}$  denote a spanning tree of  $G$  with the set of vertices  $V_{i,\alpha} = \{v_j \in V(G) \mid d_G(v_i, v_j) \leq \alpha\}$ . For each  $v_i \in D$  start traversing from  $x_i$  in  $G''$ . Let  $Y_i = \{y_\theta \in Y \mid d_{G''}(x_i, y_\theta) = d.\theta, \forall \theta = 1, 2, \dots, \ell'\}$ , where  $\ell'$  is the number of segments between  $x_i$  and  $x_j$ , where  $x_j$  corresponds to  $v_j \in V_{i, \lfloor \frac{d}{2} \rfloor}$ . Let  $Y' = \bigcup_{x_i \in X'} Y_i$ . The set  $X' \cup Y'$  is a DdIS in  $G''$ . Observe that there are some segments (corresponding to the edges which are not part of any  $G_{i, \lfloor \frac{d}{2} \rfloor}$ ) that have not been traversed in the above process. Now, we consider every such segment and choose the  $\lceil \frac{d}{2} \rceil$ -th point on it. Let  $Y''$  be the set of chosen points. Needless to say,  $Y''$  is also a DdIS of  $G''$ .

Also, observe that there exists no pair of points  $y_\alpha \in Y'$  and  $y_\beta \in Y''$  such that  $d_{G''}(y_\alpha, y_\beta) < d$ . On the contrary, suppose  $d_{G''}(y_\alpha, y_\beta) < d$ . Implies,  $y_\alpha$  and  $y_\beta$  are from two segments, each having one, incident at some  $x_j \in X \setminus X'$ , where  $x_j$  corresponds to a leaf  $v_j \in G_{i, \lfloor \frac{d}{2} \rfloor}$ . Note that  $d_{G''}(y_\alpha, x_j) \geq \lfloor \frac{d}{2} \rfloor$  and  $d_{G''}(x_j, y_\beta) \geq \lceil \frac{d}{2} \rceil$ . Implies,  $d_{G''}(y_\alpha, y_\beta) \geq d$ , arrived at a contradiction. Let  $D' = X' \cup Y' \cup Y''$ . As per our selection method each segment contributes one point in  $Y' \cup Y''$ . Thus,  $|D'| \geq k + \ell$  since  $|X'| \geq k$  and  $|Y' \cup Y''| = \ell$ .

**(Sufficiency).** Let  $G''$  have a DdIS  $D'$  of cardinality at least  $k + \ell$  and  $D = \{v_i \in V \mid x_i \in D' \cap X\}$ . Observe that  $|D' \cap Y| \leq \ell$  (due to Lemma 2); so  $|D| \geq k$ . We shall show that, by suitably modifying  $D$  (i.e., by removing some of the vertices in  $D$ ), we get at least  $k$  points from  $X$  such that the set of corresponding vertices in  $G$  is a DdIS of  $G$ . Consider a pair of vertices  $v_i, v_j \in D$  such that  $d_G(v_i, v_j) < d$  in  $G$  (if there is no such pair, then  $D$  is a DdIS of  $G$  with  $|D| \geq k$ ). Let  $x_i, x_j \in D' \cap X$  be the vertices in  $G''$  corresponding to  $v_i, v_j \in D$ , respectively. Also, let  $\hat{\ell}$  be the number of segments on the path  $x_i \rightsquigarrow x_j$  corresponding to the shortest path  $v_i \rightsquigarrow v_j$ . As each segment can contribute at most one point (from  $Y$ ) in any solution,  $D'$  can contain at most  $\hat{\ell} + 1$  points (including  $x_i$  and  $x_j$ ) from the path  $x_i \rightsquigarrow x_j$ .

We update  $D$  by deleting one of the conflicting vertices  $v_i, v_j$  from  $D$ . The assumption that  $d_G(v_i, v_j) < d$  implies that one of the segments on the path

$x_i \rightsquigarrow x_j$  has no point in  $D' \cap Y$ , so  $|D' \cap Y| < \ell$  and we still have  $|D| \geq k$  after removal.

Repeat the same for all pair of points in  $D$  for which the shortest path distance is less than  $d$  in  $G$ . Therefore,  $|D| \geq k$  and  $D$  is a distance- $d$  independent set in  $G$ .  $\square$

**Theorem 1.** *GDDIS problem is NP-complete for unit disk graphs.*

## 4 Approximation Algorithm

In this section, we discuss a simple 4-factor approximation algorithm for the GMDdIS problem, for a fixed constant  $d \geq 3$ . Let  $\mathcal{R}$  be the rectangular region containing the point set  $P$  (disk centers). Let  $G$  be the UDG defined on  $P$ . We partition  $\mathcal{R}$  into disjoint horizontal strips  $H_1, H_2, \dots, H_\nu$ , each of width  $d$  ( $H_\nu$  may be of width less than  $d$ ). The basic idea behind our algorithm is as follows:

- (i) Compute a feasible solution for each non-empty strip  $H_i$  ( $1 \leq i \leq \nu$ ) independently as stated below:

We split the horizontal strip into squares of size  $d \times d$ . In each square, we compute an optimum solution of the GMDdIS problem defined on that square. We consider all odd numbered squares and compute the union  $S_{odd}^i$  of optimum solutions of these squares. Similarly, the union  $S_{even}^i$  of optimum solutions of all even numbered squares are also computed. Each of these is a feasible solution of GMDdIS problem in  $H_i$  as the minimum distance between each pair of considered squares is at least  $d$ . We choose  $S^i = S_{even}^i$  or  $S_{odd}^i$  such that  $|S^i| = \max(|S_{even}^i|, |S_{odd}^i|)$  as the desired feasible solution for the strip  $H_i$ .

- (ii) Compute  $S_{even}$  and  $S_{odd}$ , which are the union of the solutions of even and odd strips respectively, and
- (iii) Report  $S^* = S_{even}$  or  $S_{odd}$  such that  $|S^*| = \max(|S_{even}|, |S_{odd}|)$  as a solution to the GMDdIS problem.

Note that, thus, the solution obtained in the above process is a feasible solution for the entire problem.

**Lemma 4.** *If  $OPT$  is an optimum solution for the GMDdIS problem, then  $\max(|S_{even}|, |S_{odd}|) \geq \frac{1}{4}|OPT|$ .*

*Proof.* Let us denote by  $OPT$  an optimum solution for the given GMDdIS problem, and by  $OPT^i$  an optimum solution of the non-empty strip  $H_i$ . Since any two even (resp. odd) numbered strips, say  $H_i$  and  $H_j$ , are at least  $d$  distance apart, the feasible solutions computed in any method for  $H_i$  and  $H_j$  are independent<sup>2</sup>. Thus, both  $OPT_{even} = \bigcup_{i \text{ is even}} OPT^i$  and  $OPT_{odd} = \bigcup_{i \text{ is odd}} OPT^i$  are feasible solutions for the given GMDdIS problem.

<sup>2</sup> By independent we mean for any  $p_i \in H_i \cap P$  and  $p_j \in H_j \cap P$ ,  $p_i$  and  $p_j$  are distance- $d$  independent and also,  $OPT^i \cap OPT^j = \emptyset$ .

Note that  $|OPT| \leq |OPT_{even}| + |OPT_{odd}| \leq 2|OPT_*|$ , where  $OPT_* = OPT_{even}$  if  $|OPT_{even}| > |OPT_{odd}|$ ; otherwise  $OPT_* = OPT_{odd}$ .

Also, note that we have not computed  $OPT^i$  for the strip  $H_i$ . Instead, we have computed  $S_{even}^i$  and  $S_{odd}^i$  by splitting the strip  $H_i$  into  $d \times d$  squares, and accumulating the optimum solutions of even and odd numbered squares separately. By the same argument as stated above, we have  $|OPT^i| \leq 2|S^{i*}|$ , where  $S^{i*} = S_{even}^i$  if  $|S_{even}^i| \geq |S_{odd}^i|$ ; otherwise  $S^{i*} = S_{odd}^i$ .

Combining both the inequalities, we have  $|OPT| \leq 4 \max(|S_{even}|, |S_{odd}|)$ .

□

### 4.1 Solving a $d \times d$ Square Optimally

Let  $\mathcal{Q} \subseteq P$  be the set of points inside a  $d \times d$  square  $\chi$ , and  $G_\chi$  be the UDG defined on  $\mathcal{Q}$  (i.e.,  $G_\chi$  is an induced subgraph of  $G$  with vertex set  $\mathcal{Q}$ ). Solving  $\chi$  in time  $d^4 n^{O(d^2)}$  is trivial as the number of points of  $\mathcal{Q}$  in any optimal solution is  $O(d^2)$  (by checking all possible subsets of size  $O(d^2)$ ). However, an optimal solution in  $\chi$  can be obtained with a better running time.

Note that  $G_\chi$  need not be a connected graph. Let  $C_1, C_2, \dots, C_l$  be the components of  $G_\chi$  such that any two components in  $G_\chi$  are at least  $d$  distance apart<sup>3</sup> in  $G$ . The following lemmas Lemmas 5 and 6 give loose upper bounds on the total number of components and the cardinality of a DdIS in any component, respectively.

**Lemma 5.** *The worst case number of components in  $G_\chi$  is  $O(d^2)$ .*

In order to have the worst case size of a DdIS in  $G_\chi$ , we need to have an idea about the worst case size of a DdIS in a component in  $G_\chi$ .

**Lemma 6.** *Let  $C$  be any component of  $G_\chi$ . The number of mutually distance- $d$  independent points in  $C$  is bounded by  $O(d)$ .*

*Proof.* Consider the square region  $\chi'$  of size  $3d \times 3d$  whose each side is  $d$  distance away from the corresponding side of  $\chi$ . Let  $\mathcal{Q}' \subseteq P$  be the subset of points in  $\chi'$ . Partition  $\chi'$  into cells of size  $\frac{1}{2\sqrt{2}} \times \frac{1}{2\sqrt{2}}$ . Thus, the number of cells in  $\chi'$  is  $O(d^2)$ , and in each cell the unit disks centered at the points inside that cell are mutually connected. Let a pair of points  $p_i, p_j \in C$  which are distance- $d$  independent. The shortest path  $p_i \rightsquigarrow p_j$  between  $p_i$  and  $p_j$  entirely lies inside  $\chi'$ . If there is another point  $p_k \in C$  which is distance- $d$  independent with both  $p_i$  and  $p_j$ , then  $p_k$  is at least distance  $\frac{d}{2}$  away from each point on the path  $p_i \rightsquigarrow p_j$ . Thus, the path from  $p_k$  to any point on the path  $p_i \rightsquigarrow p_j$  occupies at least  $O(d)$  cells, and none of the points from these cells are distance- $d$  independent to all the points  $p_i, p_j, p_k$ . Thus, the addition of each point in the set of mutually distance- $d$  independent points in  $\chi$  prohibits points in  $O(d)$  cells to belong in that set, and hence the lemma follows. □

<sup>3</sup> If there are two components of  $G_\chi$  having distance less than  $d$  in  $G$ , then we can view them as a single component.



**Lemma 7.** *An optimal DdIS in  $\chi$  can be computed in  $d^2n^{O(d)}$  time.*

*Proof.* Let  $\mathcal{M}$  be a matrix containing distances of the all pair shortest paths in  $G$ . By definition, intersection of distance- $d$  independent sets of any two components is empty. Thus, a DdIS of maximum size in  $G_\chi$  can be computed by considering all components of the UDG  $G_\chi$ , and computing the union of the DdIS of maximum sizes of those components. We consider each component of  $G_\chi$  separately. For each component  $C$ , consider all possible tuples of size at most  $O(d)$  (due to Lemma 6) and for each tuple, check whether they form a DdIS or not by consulting the matrix  $\mathcal{M}$  in  $O(d^2)$  time. Thus, a maximum size DdIS in  $C$  can be computed in  $O(d^2|C|^{O(d)})$  time, and the total time for computing a maximum size DdIS in  $G_\chi$  is  $O(d^2 \sum_{C \in G_\chi} |C|^{O(d)}) = d^2|\mathcal{Q}|^{O(d)}$ .  $\square$

**Theorem 2.** *Given a set  $P$  of  $n$  points in the plane, we can always compute a DdIS of size at least  $\frac{1}{4}|OPT|$  in  $d^2n^{O(d)}$  time, where  $|OPT|$  is the cardinality of a GMDdIS.*

## 5 Approximation Scheme

In this section, using the shifting strategy [16], we propose a polynomial time approximation scheme (PTAS) for the GMDdIS problem, for a given fixed constant  $d \geq 3$ . Let  $\mathcal{R}$  be an axis parallel rectangular region containing the point set  $P$  (i.e., centers of the disks of the given UDG). Compute the all pair shortest paths between every pair of vertices in the UDG  $G$  defined on  $P$  and store them in a matrix  $\mathcal{M}$ .

We use two-level nested shifting strategy. The first level executes  $k$  iterations, where  $k \gg d$ . The  $i$ -th iteration ( $1 \leq i \leq k$ ) of the first level is as follows:

- Assuming  $\mathcal{R}$  is left-open, partition  $\mathcal{R}$  into vertical strips such that (a) first strip is of width  $i$ , (b) every even strip is of width  $d$ , and (c) every odd strip, except the first strip, is of width  $k$ .
- Without loss of generality, assume that the points lying on the left boundary of a strip belong to the adjacent strip to its left (i.e., every strip is left open and right closed).
- Compute some desired feasible solutions for the odd strips (of width  $k$ ). These solutions can be merged to produce a solution of the entire problem since these odd numbered strips are distance- $d$  apart.

The second level of the nested shifting strategy is used to find a solution for an iteration in the first level. We consider each non-empty odd strip separately, and execute  $k$  iterations. In the  $i$ -th iteration, we partition it horizontally as in the first level (mentioned in the first bullet above). We get a solution of a strip by solving each  $k \times k$  square in that strip optimally. The union of the solutions of all the odd numbered squares/rectangles in that strip is the desired solution of that vertical strip of the first level. Finally, we take the union of the solutions of all the odd vertical strips to compute the solution of that iteration of the first

level. Thus, we have the solutions of all the iterations of the first level. We report the one having the maximum cardinality as the solution of the given GMD $d$ IS problem. The method of computing an optimum solution inside a  $k \times k$  square is described below.

### 5.1 Computing an Optimum Solution in a $k \times k$ Square

We apply a divide and conquer strategy to compute an optimum solution for the GMD $d$ IS problem defined on a set of points  $\mathcal{Q} \subseteq P$  inside a square  $\chi$  of size  $k \times k$ . We partition  $\chi$  into four sub-squares, each of size  $\frac{k}{2} \times \frac{k}{2}$ , using a horizontal line  $\ell_h$  and a vertical lines  $\ell_v$  (see Fig. 2(b)). Let  $\mathcal{Q}_1 \subseteq \mathcal{Q}$  be the subset of points in  $\chi$  which are at most  $d$  distance away from  $\ell_h$  and/or  $\ell_v$  (see Fig. 2(c)). Let  $\mathcal{Q}_2$  be a maximum cardinality subset of  $\mathcal{Q}_1$  such that all the points in  $\mathcal{Q}_2$  are pairwise distance- $d$  independent in  $P$ .

**Lemma 8.**  $|\mathcal{Q}_2| \leq O(k)$ .

*Proof.* Consider a vertical (resp. horizontal) strip of size  $(k + 2d) \times 4d$  (resp.  $4d \times (k + 2d)$ ) around  $\ell_v$  (resp.  $\ell_h$ ), and partition it into cells of size  $\frac{1}{2\sqrt{2}} \times \frac{1}{2\sqrt{2}}$  (showed as shaded region in Fig. 2(d)). Using the similar combinatorial argument discussed in the proof of Lemma 6, we can argue that the number of points of  $\mathcal{Q}_1$ , in both strips, contributed to any optimum solution is  $O(k)$ . Therefore  $|\mathcal{Q}_2| \leq O(k)$  as  $\mathcal{Q}_2 \subseteq \mathcal{Q}_1$ .  $\square$

We apply the divide and conquer strategy on  $\chi$  as follows:

**Step 1:** Choose all possible subsets of points of sizes at most  $O(k)$  in  $\mathcal{Q}_1$ .

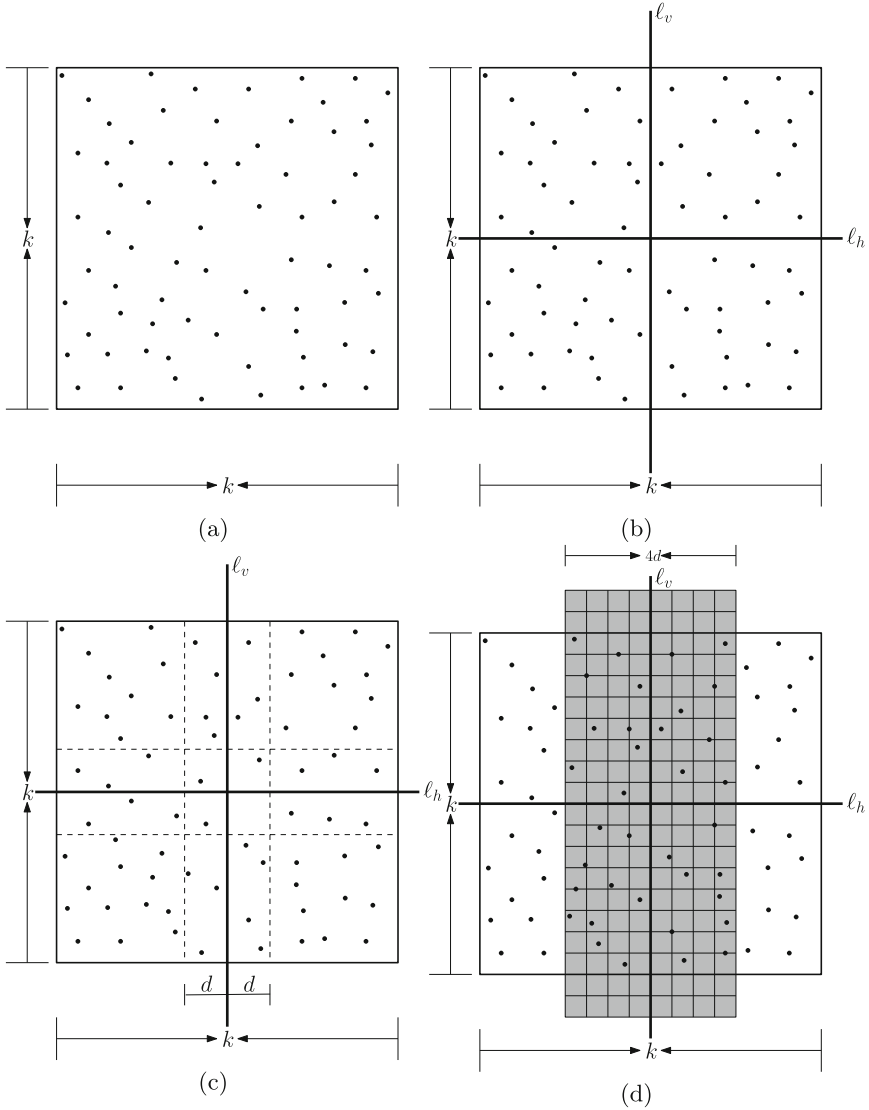
**Step 2:** For each subset, do the following:

- Check whether they are mutually distance- $d$  independent by consulting  $\mathcal{M}$ . If so, then they form  $\mathcal{Q}_2$ .
- Consult the matrix  $\mathcal{M}$  to delete the points in  $\chi$  which are at most distance  $d - 1$  away from each member in  $\mathcal{Q}_2$ .
- Recursively solve the four independent subproblems defined by the points of  $\mathcal{Q} \setminus \mathcal{Q}_1$  in the four quadrants  $\chi_1, \chi_2, \chi_3, \chi_4$  defined by  $\ell_h$  and  $\ell_v$ .
- Return  $\mathcal{Q}_2 = \mathcal{Q}_2 \cup (\bigcup_{i=1}^4 \mathcal{Q}_2^i)$ , where  $\mathcal{Q}_2^i$  is the solution of the subproblem on the points of  $\chi_i$ .
- Retain the solution for the present subset if it is better than the solutions produced by earlier choices of  $\mathcal{Q}_2$ .

**Lemma 9.** *The solution produced for the cell  $\chi$  (of size  $k \times k$ ) in the afore-said process is optimum, and the time complexity of the proposed algorithm is  $k^2 m^{O(k)}$ , where  $m = |\mathcal{Q}|$ .*

*Proof.* Let  $OPT_\chi$  be an optimal solution for the points lying in  $\chi$ . Note that our process checks all combinations of points of size  $|OPT_\chi|$ . Thus, the combination of points in  $OPT_\chi$  must appear at some stage in the process.

If  $T(m, k)$  denote the time complexity of computing the distance- $d$  independent set in  $\chi$ , then  $T(m, k) = 4 \times T(m, \frac{k}{2}) \times m^{O(k)} + O(k^2)$ , which is  $k^2 \times m^{O(k)}$  in the worst case.  $\square$



**Fig. 2.** (a) A square  $\chi$  of size  $k \times k$ , (b) the horizontal and vertical lines  $\ell_h$  and  $\ell_v$  divide  $\chi$  into four squares, each of size  $\frac{k}{2} \times \frac{k}{2}$ , (c) the points lying in the strips of width of  $2d$  belong to  $\mathcal{Q}_1$ , and (d) A strip of width  $4d$  around the vertical line  $\ell_v$ .

Using the analysis of [16], we have the following result.

**Theorem 3.** *Given a set  $P$  of  $n$  points (centers of the unit disks) in the plane and an integer  $k > 1$ , the proposed scheme produces a DdIS of size at least  $\frac{1}{(1+\frac{1}{k})^2} |OPT|$  in  $k^2 n^{O(k)}$  time, where  $|OPT|$  is the cardinality of a GMDdIS.*

## 6 Conclusion

In this article, we studied the maximum distance- $d$  independent set (MD $d$ IS) problem, a variant of the maximum independent set problem, on unit disk graphs. The MD $d$ IS problem asks to find a maximum cardinality subset of vertices such that any two vertices in the subset are at distance at least  $d$  in the graph. We proved that the problem is NP-hard on unit disk graphs, proposed a 4-factor approximation algorithm, and an approximation scheme for the problem. Results analogous to the above for the ordinary independent set problem (i.e., for  $d = 2$ ) are well known for many years.

**Acknowledgements.** The authors would like to thank the anonymous referees for their valuable comments and suggestions.

## References

1. Agnarsson, G., Damaschke, P., Halldórsson, M.M.: Powers of geometric intersection graphs and dispersion algorithms. *Discrete Appl. Math.* **132**(1), 3–16 (2003)
2. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. *J. ACM (JACM)* **45**(3), 501–555 (1998)
3. Biedl, T., Kant, G.: A better heuristic for orthogonal graph drawings. *Comput. Geom.* **9**(3), 159–180 (1998)
4. Chang, G.J., Nemhauser, G.L.: The  $k$ -domination and  $k$ -stability problems on sun-free chordal graphs. *SIAM J. Algebraic Discrete Methods* **5**(3), 332–345 (1984)
5. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. *Discrete Math.* **86**(1–3), 165–177 (1990)
6. Das, G.K., De, M., Kolay, S., Nandy, S.C., Sur-Kolay, S.: Approximation algorithms for maximum independent set of a unit disk graph. *Inf. Process. Lett.* **115**(3), 439–446 (2015)
7. Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.* **34**(6), 1302–1323 (2005)
8. Eto, H., Guo, F., Miyano, E.: Distance- $d$  independent set problems for bipartite and chordal graphs. *J. Comb. Optim.* **27**(1), 88–99 (2014)
9. Eto, H., Ito, T., Liu, Z., Miyano, E.: Approximability of the distance independent set problem on regular graphs and planar graphs. In: Chan, T.-H.H., Li, M., Wang, L. (eds.) COCOA 2016. LNCS, vol. 10043, pp. 270–284. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-48749-6\\_20](https://doi.org/10.1007/978-3-319-48749-6_20)
10. Eto, H., Ito, T., Liu, Z., Miyano, E.: Approximation algorithm for the distance-3 independent set problem on cubic graphs. In: Poon, S.-H., Rahman, M.S., Yen, H.-C. (eds.) WALCOM 2017. LNCS, vol. 10167, pp. 228–240. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-53925-6\\_18](https://doi.org/10.1007/978-3-319-53925-6_18)
11. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.* **32**(4), 826–834 (1977)
12. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, vol. 29. W. H. Freeman, New York (2002)
13. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete problems. In: *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, pp. 47–63. ACM (1974)

14. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.* **1**(2), 180–187 (1972)
15. Halldórsson, M.M.: Approximating discrete collections via local improvements. In: *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 160–169. Society for Industrial and Applied Mathematics (1995)
16. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM (JACM)* **32**(1), 130–136 (1985)
17. Hsu, W.L., Nemhauser, G.L.: Algorithms for minimum covering by cliques and maximum clique in claw-free perfect graphs. *Discrete Math.* **37**(2–3), 181–191 (1981)
18. Jallu, R.K., Das, G.K.: Improved algorithm for maximum independent set on unit disk graph. In: Govindarajan, S., Maheshwari, A. (eds.) *CALDAM 2016*. LNCS, vol. 9602, pp. 212–223. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-29221-2\\_18](https://doi.org/10.1007/978-3-319-29221-2_18)
19. Marathe, M.V., Breu, H., Hunt, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple heuristics for unit disk graphs. *Networks* **25**(2), 59–68 (1995)
20. Matsui, T.: Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In: Akiyama, J., Kano, M., Urabe, M. (eds.) *JCDCG 1998*. LNCS, vol. 1763, pp. 194–200. Springer, Heidelberg (2000). [https://doi.org/10.1007/978-3-540-46515-7\\_16](https://doi.org/10.1007/978-3-540-46515-7_16)
21. Minty, G.J.: On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory Ser. B* **28**(3), 284–304 (1980)
22. Montealegre, P., Todinca, I.: On distance- $d$  independent set and other problems in graphs with “few” minimal separators. In: Heggernes, P. (ed.) *WG 2016*. LNCS, vol. 9941, pp. 183–194. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53536-3\\_16](https://doi.org/10.1007/978-3-662-53536-3_16)
23. Murphy, O.J.: Computing independent sets in graphs with large girth. *Discrete Appl. Math.* **35**(2), 167–170 (1992)
24. Nandy, S.C., Pandit, S., Roy, S.: Faster approximation for maximum independent set on unit disk graph. *Inf. Process. Lett.* **127**, 58–61 (2017)
25. Nieberg, T., Hurink, J., Kern, W.: A robust PTAS for maximum weight independent sets in unit disk graphs. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) *WG 2004*. LNCS, vol. 3353, pp. 214–221. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30559-0\\_18](https://doi.org/10.1007/978-3-540-30559-0_18)
26. Poljak, S.: A note on stable sets and colorings of graphs. *Commentationes Math. Univ. Carol.* **15**(2), 307–309 (1974)



# New Approximation Algorithms for the Minimum Cycle Cover Problem

Wei Yu<sup>1</sup>(✉), Zhaohui Liu<sup>1</sup>, and Xiaoguang Bao<sup>2</sup>

<sup>1</sup> Department of Mathematics, East China University of Science and Technology,  
Shanghai 200237, China

{yuwei,zhliu}@ecust.edu.cn

<sup>2</sup> College of Information Technology, Shanghai Ocean University,  
Shanghai 201306, China

xgbao@shou.edu.cn

**Abstract.** Given an undirected weighted graph  $G = (V, E)$  with nonnegative weight function obeying the triangle inequality, a set  $\{C_1, C_2, \dots, C_k\}$  of cycles is called a *cycle cover* if  $V \subseteq \bigcup_{i=1}^k V(C_i)$  and its cost is given by the maximum weight of the cycles. The Minimum Cycle Cover Problem aims to find a cycle cover of cost at most  $\lambda$  with the minimum number of cycles. An  $O(n^2)$  24/5-approximation algorithm and an  $O(n^5)$  14/3-approximation algorithm are given by Yu and Liu (Theor Comput Sci 654:45–58, 2016). However, the original proofs for approximation ratios are incomplete. In this paper we first present a corrected simplified analysis on the 24/5-approximation algorithm. Then we give a new  $O(n^3)$  approximation algorithm that achieves the same ratio 14/3 and has much simpler proof on the approximation ratio. Moreover, we derive an improved 32/7-approximation algorithm that runs in  $O(n^5)$ .

**Keywords:** Vehicle routing · Cycle cover  
Traveling Salesman Problem · Approximation algorithm

## 1 Introduction

Given an undirected weighted graph  $G = (V, E)$  with nonnegative weight function obeying the triangle inequality, a set  $\{C_1, C_2, \dots, C_k\}$  of cycles is called a *cycle cover* if  $V \subseteq \bigcup_{i=1}^k V(C_i)$  and the cost of a cycle cover is given by the maximum weight of the cycles. The goal of the Minimum Cycle Cover Problem (MCCP) is to find a cycle cover of cost at most  $\lambda$  with the minimum number of cycles. This fundamental vehicle routing problem and its variants (min-max cycle/tree/path cover problem, minimum tree/path cover problems, etc.) have a wide range of application and have received considerable research attention in the past twenty years (see [1–4, 8–11]).

MCCP is NP-hard since it generalizes the well-known Traveling Salesman Problem (TSP). Arkin et al. [1] first give a 6-approximation algorithm for

MCCP. The same result can also be achieved by a combination of the edge-doubling strategy with the 3-approximation algorithm, proposed by the same authors, for a closely related Minimum Tree Cover Problem (MTCP) which is obtained by replacing cycles with trees in MCCP. Khani and Salavatipour [6] derived an improved  $5/2$ -approximation algorithm for MTCP, which implies a 5-approximation algorithm for MCCP. Recently, Yu and Liu [12] showed that a  $\rho$ -approximation algorithm for TSP can be transformed into a  $4\rho$ -approximation algorithm for MCCP. They also proposed a simple  $O(n^2)$   $24/5$ -approximation algorithm and a matching-based  $O(n^5)$   $14/3$ -approximation algorithm for MCCP.

In the rooted version of MCCP (RMCCP), each cycle is required to contain at least one vertex in some prescribed depot set  $D \subseteq V$  and the objective is to cover the vertices in  $V \setminus D$  to minimize the number of cycles. Currently, all the available results for RMCCP consider only the case of one single depot vertex, which is called the Distance Constrained Vehicle Routing Problem by Nagarajan and Ravi [7]. The authors developed an approximation algorithm of ratio  $\min\{\log n, \log \lambda\}$ , which is improved to  $\frac{\log \lambda}{\log \log \lambda}$  by Friggstad and Swamy [5]. Actually, whether single-depot RMCCP admits a constant-factor approximation algorithm is a major open problem in this area, although a 2-approximation algorithm for the problem defined on a tree is given by Nagarajan and Ravi [7].

In this paper, we note that there are some missing cases in the analysis of the  $24/5$ - and  $14/3$ -approximation algorithms in [12]. In particular, the enumeration of the configurations of the bad cycles is incomplete, as shown in Figs. 1 and 2. The original goal of this paper is to give a corrected analysis on these algorithms. In this course we not only correct the analysis but also simplify the proofs significantly. For the  $24/5$ -approximation algorithm in [12] we show a much simpler proof. Based on the simplified approach and some new observations, we present a new  $14/3$ -approximation algorithm that runs in  $O(n^3)$  and give an improved  $32/7$ -approximation algorithm that runs in  $O(n^5)$ .

The rest of the paper is organized as follows. We formally state the problem and give some preliminary results in Sect. 2. In Sect. 3 we analyze the simple  $24/5$ -approximation algorithm while our new  $14/3$ -approximation algorithm is discussed in Sect. 4. And the improved  $32/7$ -approximation algorithm is presented in Sect. 5.

## 2 Preliminaries

Given an undirected weighted graph  $G = (V, E)$  with vertex set  $V$  and edge set  $E$ ,  $w(e)$  denotes the weight or length of edge  $e$ . If  $e = (u, v)$ , we also use  $w(u, v)$  to denote the weight of  $e$ . For  $B > 0$ ,  $G[B]$  denotes the subgraph of  $G$  obtained by removing all the edges in  $E$  with weight greater than  $B$ . For a subgraph  $H$  (e.g. tree, cycle, path) of  $G$ , let  $V(H), E(H)$  be the vertex set and edge set of  $H$ , respectively. The weight of  $H$  is defined as  $w(H) = \sum_{e \in E(H)} w(e)$ . If  $H$  is connected, let  $MST(H)$  be the minimum spanning tree on  $V(H)$  and its weight  $w(MST(H))$  is simplified to  $w_T(H)$ . The distance between  $H$  and another subgraph  $H'$  is given by

$$d(H, H') = \min\{w(e) \mid e = (u, v) \text{ such that } u \in V(H), v \in V(H')\}.$$

A cycle  $C$  is also called a tour on  $V(C)$ . A cycle (path, tree) containing only one vertex and no edges is a trivial cycle (path, tree) and its weight is defined as zero. A set  $\{C_1, C_2, \dots, C_k\}$  of cycles (some of them may be trivial cycles) is called a *cycle cover* if  $V \subseteq \bigcup_{i=1}^k V(C_i)$ . And the cost of this cycle cover is defined as  $\max_{1 \leq i \leq k} w(C_i)$ , i.e., the maximum weight of the cycles. By replacing cycles with trees we can define *tree cover* and its cost similarly.

The Minimum Cycle Cover Problem (MCCP) is defined as follows. Given an undirected weighted complete graph  $G = (V, E)$  with nonnegative and metric (i.e., obeying the triangle inequality) weight function  $w(\cdot)$  on  $E$ , the aim is to find a cycle cover of cost at most  $\lambda$  with the minimum number of cycles.

In the above definition we assume that the graph is complete. This involves no loss of generality since we can take the metric closure of a connected graph if it is not complete. For a disconnected graph, we simply connect distinct connected components by edges with sufficiently large weights. We also suppose w.l.o.g. that the weights of the edges and  $\lambda$  are integers.

Given an instance of MCCP,  $OPT$  indicates its optimal value and each cycle in the optimal solution is called an optimum cycle. By the triangle inequality, we can assume w.l.o.g. that any two optimum cycles are vertex-disjoint. We use  $n$  to denote the number of vertices of  $G$ .

The following result on decomposing a tree of large weight into a series of trees of small weight is very useful to design and analyze the algorithms for MCCP.

**Lemma 1.** [3, 6] *Given  $B > 0$  and a tree  $T$  with  $\max_{e \in E(T)} w(e) \leq B$ , we can decompose  $T$  into  $k \leq \max\{\lfloor \frac{w(T)}{B} \rfloor, 1\}$  edge-disjoint trees  $T_1, T_2, \dots, T_k$  with  $w(T_i) \leq 2B$  for each  $i = 1, 2, \dots, k$  in  $O(|V(T)|)$  time.*

### 3 A Simple $\frac{24}{5}$ -Approximation

The main idea of the  $\frac{24}{5}$ -approximation algorithm in [12] is as follows. First, we obtain the graph  $G[\frac{\lambda}{5}]$  by deleting all the edges with weight greater than  $\frac{\lambda}{5}$ . Then the connected components of  $G[\frac{\lambda}{5}]$  are partitioned into *light components*  $F_1, F_2, \dots, F_l$  with  $w_T(F_i) \leq \frac{\lambda}{2}$  ( $i = 1, 2, \dots, l$ ) and *heavy components*  $F_{l+1}, F_{l+2}, \dots, F_{l+h}$  with  $w_T(F_i) > \frac{\lambda}{2}$  ( $i = l+1, l+2, \dots, l+h$ ). Next we construct a tree cover of cost at most  $\frac{\lambda}{2}$  by taking the minimum spanning trees of all the light components as well as the trees of weight at most  $\frac{\lambda}{2}$  obtained by decomposing the minimum spanning trees of the  $h$  heavy components using Lemma 1. Lastly, by doubling all the edges in the tree cover we derive a cycle cover of cost at most  $\lambda$ . The following is the formal description of the algorithm.



**Algorithm *MCCP1***

Step 1. Delete all the edges with weight greater than  $\frac{\lambda}{5}$  in  $G$  to obtain  $G[\frac{\lambda}{5}]$  with light components  $F_1, F_2, \dots, F_l$  and heavy components  $F_{l+1}, F_{l+2}, \dots, F_{l+h}$ .

Step 2. Set  $T := \emptyset$  and  $S := \emptyset$ .

- (i) For each  $i = 1, 2, \dots, l$ , put  $MST(F_i)$  into  $T$ ;
- (ii) For each  $s = l + 1, l + 2, \dots, l + h$ , decompose  $MST(F_s)$  by Lemma 1 into a set of trees of weight at most  $\frac{\lambda}{2}$  and put them into  $T$ .
- (iii) For each tree in  $T$ , double all the edges to obtain an Eulerian graph and shortcut the repeated vertices of the Eulerian tour of this graph to obtain a cycle. And put this cycle into  $S$ .

Step 3. Return the cycle cover  $S$ .

By construction it is easy to see that  $S$  is a cycle cover of cost no greater than  $\lambda$ . We proceed to show that the number of cycles in  $S$  is not greater than  $\frac{24}{5}OPT$ .

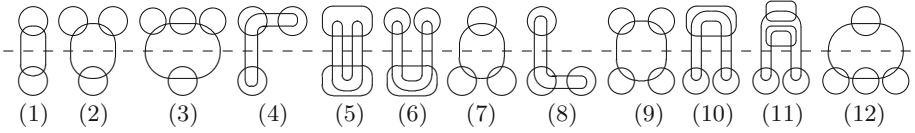
To analyze the algorithm, let  $OPT = k$  and  $C_1^*, C_2^*, \dots, C_k^*$  be the vertex-disjoint optimum cycles. Given a subgraph  $H$  of  $G$ , if  $V(F_i) \cap V(H) \neq \emptyset$  for some  $i$  with  $1 \leq i \leq l + h$ , we say that  $F_i$  is *incident* to  $H$  or  $H$  is *incident* to  $F_i$ . Therefore, all the optimum cycles can be classified into three types. The first type of optimum cycles, called *light cycles*, are incident to only light components. The second type of optimum cycles, i.e., *heavy cycles*, are incident to only heavy components. The last type of optimum cycles, known as *bad cycles*, are incident to at least one light component and at least one heavy component. Let  $k_l, k_h, k_b$  be the number of light, heavy and bad cycles, respectively. Clearly,  $k = k_l + k_h + k_b$ .

An edge  $e$  of an optimum cycle with weight  $w(e) > \frac{\lambda}{5}$  is referred to as a *long edge*. For a long edge  $e$ , if it is incident to only one connected component of  $G[\frac{\lambda}{5}]$  we call it an *internal long edge*. Otherwise, i.e.,  $e$  is incident to two distinct connected components of  $G[\frac{\lambda}{5}]$ ,  $e$  is called an *external long edge*. For any optimum cycle  $C$ , the number of external long edges of  $C$  is denoted by  $Ex(C)$ . Since  $w(C) \leq \lambda$ , we have  $Ex(C) \leq 4$ . Moreover, an optimum cycle cannot include a single external long edge since a cycle is a closed path, which implies  $Ex(C) \geq 2$ . By these facts, all possible configurations of a bad cycle are shown in Fig. 1. For example, the fourth configuration indicates that the bad cycle is incident to two heavy components and one light component. There are two external long edges that are incident to two distinct heavy components and another two external long edges that are incident to one light component and one heavy component.

For  $j = 1, 2, 3$ , denote by  $B_j$  the set of bad cycles incident to exactly  $j$  light components and set  $k_j = |B_j|$ . It can be seen from Fig. 1 that  $B_1, B_2, B_3$  contains exactly the bad cycles with configurations (1)–(6), (7)–(11) and (12), respectively. Therefore,  $k_b = \sum_{j=1}^3 k_j$ .

First we bound the number  $l$  of trees generated in the Step 2(i) of Algorithm *MCCP1*.

heavy components



light components

**Fig. 1.** The configurations of a bad cycle. The circles or ovals not intersecting with the dashed line represent light or heavy components and the closed curves crossing the dashed line indicate the bad cycles.

**Lemma 2.**  $l \leq 4k_l + k_b + k_2 + 2k_3$ .

*Proof.* On the one hand, each light component is incident to at least one light or bad cycle since the optimal solution is a feasible cycle cover. On the other hand, each light cycle is incident to at most four light components and each bad cycle in  $B_j$  ( $j = 1, 2, 3$ ) is incident to  $j$  light components. Thus the number  $l$  of light components is at most  $4k_l + k_1 + 2k_2 + 3k_3 = 4k_l + k_b + k_2 + 2k_3$ .  $\square$

By this lemma, if  $h = 0$  Algorithm *MCCP1* returns a 4-approximate solution since in this case  $k = k_l$  and Step 2(ii) does not generate any trees. So we assume  $h \geq 1$  in the remaining discussion of this section.

Before we bound the number of trees generated in the Step 2(ii) of Algorithm *MCCP1*, we give an upper bound on the total weight of the minimum spanning trees of the heavy components.

**Lemma 3.**  $\sum_{s=l+1}^{l+h} w_T(F_s) \leq \frac{6}{5}k_h\lambda + \frac{4}{5}k_b\lambda - \frac{1}{5}(k_2 + 2k_3)\lambda - \frac{h}{5}\lambda$ .

*Proof.* Let  $H^0$  be the subgraph (of  $G$ ) consisting of  $k_h$  heavy cycles and  $k_b$  bad cycles. Since each optimum cycle is of length at most  $\lambda$ , we have

$$w(H^0) \leq (k_h + k_b)\lambda. \tag{1}$$

Next we obtain a subgraph  $H^1$  from  $H^0$  by the following three operations. First, for each heavy cycle  $C$  with  $Ex(C) = 0$  we delete an arbitrary edge of it to transform it into a path. Second, for each heavy cycle  $C$  with  $Ex(C) \geq 1$  we delete all the external long edges to turn it into  $Ex(C)$  paths. Lastly, we delete all the external long edges of the bad cycles to obtain more paths and throw away the paths incident to light components. By construction the three operations generate some paths in  $H^1$  and each path is incident to only one heavy component.

Let  $p_0$  be the number of heavy cycles  $C$  with  $Ex(C) = 0$ . Apparently, the first operation generates  $p_0$  paths. In the second operation the total number of paths generated equals

$$q_h = \sum_{C \text{ is a heavy cycle in } H^0 \text{ with } Ex(C) \geq 1} Ex(C).$$

The total number of paths generated in the last operation equals  $\sum_{j=1}^3 p_j$ , where  $p_j (j = 1, 2, 3)$  is the number of paths derived by the bad cycles in  $B_j$ .

For  $j = 1, 2, 3$ , let  $q_j = \sum_{C \in B_j} Ex(C)$ , then  $q_b = \sum_{j=1}^3 q_j$  is the total number of external long edges of the bad cycles. For each bad cycle  $C \in B_j (j = 1, 2, 3)$ , we have  $Ex(C) \geq 1$  by definition and by removing all its external long edges we obtain  $Ex(C)$  paths. Since  $C$  is incident to  $j$  light components, at least  $j$  paths among the  $Ex(C)$  paths are incident to light components and hence are thrown away in the last operation of the construction of  $H^1$  (Note that an optimum cycle may enter/leave the same light component several times, thus creating multiple paths within the same component.) As a result,  $C$  derives at most  $Ex(C) - j$  paths in  $H^1$ . Therefore, we have

*Claim.* For  $j = 1, 2, 3$ , any bad cycle  $C \in B_j$  that derives at most  $r$  paths in  $H^1$ , it holds that  $Ex(C) \geq r + j$ .

Using the above claim, the  $p_j$  paths in  $H^1$  derived by the bad cycles in  $B_j$  correspond to at least  $p_j + jk_j$  external long edges, i.e.,  $q_j \geq p_j + jk_j$ . Therefore,

$$q_b = \sum_{j=1}^3 q_j \geq \sum_{j=1}^3 (p_j + jk_j) = \sum_{j=1}^3 p_j + k_b + k_2 + 2k_3. \tag{2}$$

Since we delete at least  $q_h + q_b$  external long edges from  $H^0$  to obtain  $H^1$ , we have

$$w(H^1) \leq w(H^0) - (q_h + q_b) \cdot \frac{\lambda}{5} \leq (k_h + k_b)\lambda - (q_h + q_b) \cdot \frac{\lambda}{5}, \tag{3}$$

where the first inequality follows from the definition of external long edges and the second inequality follows from (1).

In  $H^1$ , for each  $s = l + 1, l + 2, \dots, l + h$  we connect the  $k_s$  paths incident to  $F_s$  by  $k_s - 1$  edges in  $E(F_s)$  to obtain a spanning tree  $\hat{T}_s$  of  $V(F_s)$ . This is feasible since  $F_s$  is a connected component. The resulted subgraph  $H^2$  has  $h$  connected components while  $H^1$  consists of  $q_h + \sum_{j=0}^3 p_j$  connected components (more exactly, paths) as shown before. So the total number of edges added to  $H^2$  is  $\sum_{s=l+1}^{l+h} (k_s - 1) = q_h + \sum_{j=0}^3 p_j - h$ . Since the weight of each edge in  $\bigcup_{s=l+1}^{l+h} E(F_s)$  is at most  $\frac{\lambda}{5}$ , we have

$$\begin{aligned} w(H^2) &\leq w(H^1) + (q_h + \sum_{j=0}^3 p_j - h) \cdot \frac{\lambda}{5} \\ &\leq k_h \lambda + \frac{p_0}{5} \lambda + (5k_b + \sum_{j=1}^3 p_j - q_b - h) \cdot \frac{\lambda}{5} \\ &\leq \frac{6}{5} k_h \lambda + (4k_b - k_2 - 2k_3 - h) \cdot \frac{\lambda}{5} \\ &= \frac{6}{5} k_h \lambda + \frac{4}{5} k_b \lambda - \frac{1}{5} (k_2 + 2k_3) \lambda - \frac{h}{5} \lambda, \end{aligned} \tag{4}$$

where the second inequality follows from (3) and the third inequality holds by (2) and  $p_0 \leq k_h$ .

Note that an edge in  $\hat{T}_s$  is either an edge in  $E(F_s)$  or an internal long edge. Hence  $w(\hat{T}_s) \geq w_T(\tilde{F}_s)$ , where  $\tilde{F}_s$  is obtained by adding to  $F_s$  all the internal long edges whose end vertices are both included in  $V(F_s)$ . Since each internal long edge is of weight greater than  $\frac{\lambda}{5}$  and each edge of  $E(F_s)$  is of weight no greater than  $\frac{\lambda}{5}$ ,  $MST(F_s)$  is identical to  $MST(\tilde{F}_s)$  (to see this, imagine running Kruskal's algorithm on  $\tilde{F}_s$ . It will never add an edge of weight greater than  $\frac{\lambda}{5}$ , since the edges of weight at most  $\frac{\lambda}{5}$  already make  $\tilde{F}_s$  connected.). Therefore,  $w(\hat{T}_s) \geq w_T(\tilde{F}_s) = w_T(F_s)$ . Taking the summation for  $s = l + 1, l + 2, \dots, l + h$  and using inequality (4) prove the lemma.  $\square$

**Lemma 4.** *The Step 2(ii) of Algorithm MCCP1 generates at most  $\frac{24}{5}k_h + \frac{16}{5}k_b - \frac{4}{5}(k_2 + 2k_3)$  trees.*

*Proof.* For each  $s = l + 1, l + 2, \dots, l + h$ , by taking  $B = \frac{\lambda}{4}$  in Lemma 1 we can decompose  $MST(F_s)$  into at most  $\max\{\lfloor \frac{w_T(F_s)}{\lambda/4} \rfloor, 1\}$  trees of weight no more than  $\frac{\lambda}{2}$ . Moreover, since  $F_s$  is a heavy component we have  $w_T(F_s) \geq \frac{\lambda}{2}$ . So it holds that  $\max\{\lfloor \frac{w_T(F_s)}{\lambda/4} \rfloor, 1\} = \lfloor \frac{w_T(F_s)}{\lambda/4} \rfloor \leq \frac{w_T(F_s)}{\lambda/4}$  and the total number of trees generated in Step 2(ii) of Algorithm MCCP1 is at most

$$\sum_{s=l+1}^{l+h} \frac{w_T(F_s)}{\frac{\lambda}{4}} \leq \frac{24}{5}k_h + \frac{16}{5}k_b - \frac{4}{5}(k_2 + 2k_3),$$

where the inequality follows from Lemma 3.  $\square$

By Lemmas 2 and 4, the number of trees generated by Algorithm MCCP1 is at most

$$\begin{aligned} l + \frac{24}{5}k_h + \frac{16}{5}k_b - \frac{4}{5}(k_2 + 2k_3) &\leq 4k_l + \frac{24}{5}k_h + \frac{21}{5}k_b + \frac{1}{5}(k_2 + 2k_3) \\ &\leq 4k_l + \frac{24}{5}k_h + \frac{21}{5}k_b + \frac{2}{5}k_b \\ &= 4k_l + \frac{24}{5}k_h + \frac{23}{5}k_b. \end{aligned}$$

So we have

**Lemma 5.**  $|S| = |T| \leq \frac{24}{5}k$ .

By this lemma and a simple analysis of the complexity of Algorithm MCCP1 we have

**Theorem 1.** *There is an  $O(n^2)$   $\frac{24}{5}$ -approximation algorithm for MCCP.*

## 4 A Faster $\frac{14}{3}$ -Approximation

In this section we present a  $14/3$ -approximation algorithm for MCCP that runs in  $O(n^3)$  time while the Algorithm *MCCP2* in [12] has a time complexity of  $O(n^5)$ . To obtain the faster algorithm, we modify Algorithm *MCCP1* in the following ways. First, we delete the edges with weight greater than  $\frac{\lambda}{6}$  instead of  $\frac{\lambda}{5}$  and redefine the light (heavy) components as components with the minimum spanning tree weight no more than  $\frac{\lambda}{12}$  (greater than  $\frac{\lambda}{12}$ ). Second, we try to merge the minimum spanning trees of some light components to obtain the trees in the final tree cover.

As before let  $F_1, F_2, \dots, F_l$  and  $F_{l+1}, F_{l+2}, \dots, F_{l+h}$  be the light components and heavy components, respectively. To decide which minimum spanning trees of light components to be merged together, we solve a maximum cardinality matching problem on the following graph  $H_0$ .

**Definition 1.** *The graph  $H_0$  has  $l$  vertices  $u_1, u_2, \dots, u_l$ . For  $1 \leq i \leq l$ , the vertex  $u_i$  corresponds to the light component  $F_i$ . For  $1 \leq i < j \leq l$ ,  $(u_i, u_j)$  is an edge of  $H_0$  if and only if  $w_T(F_i) + w_T(F_j) + d(F_i, F_j) \leq \frac{\lambda}{2}$ .*

The new algorithm is described as follows.

### Algorithm *MCCP3*

Step 1. Delete all the edges with weight greater than  $\frac{\lambda}{6}$  in  $G$  to obtain  $G[\frac{\lambda}{6}]$  with light components  $F_1, F_2, \dots, F_l$  and heavy components  $F_{l+1}, F_{l+2}, \dots, F_{l+h}$ .

Step 2. Set  $T := \emptyset$  and  $S := \emptyset$ . Find a maximum cardinality matching  $M_0$  in  $H_0$  and proceed to the following substeps:

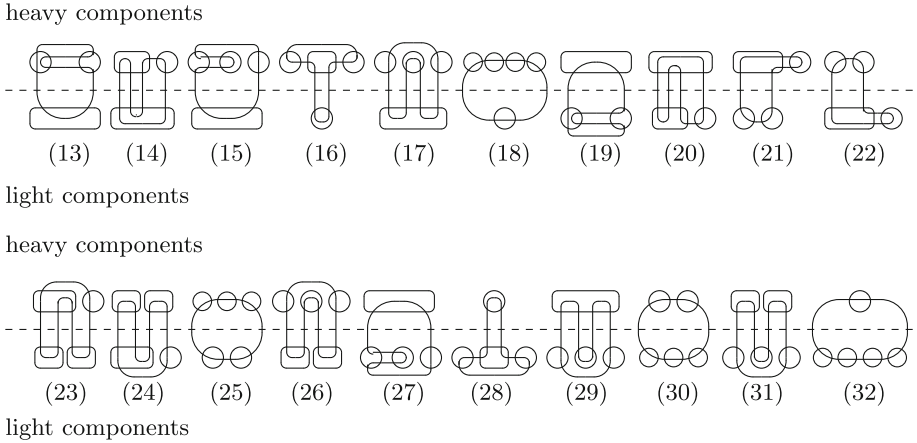
- (i) For any  $(u_i, u_j) \in M_0$ , merge  $MST(F_i)$  with  $MST(F_j)$  by an edge  $e$  in  $G$  corresponding to  $d(F_i, F_j)$  to obtain a tree and put it into  $T$ ;
- (ii) For any vertex  $u_i$  exposed by  $M_0$ , put  $MST(F_i)$  into  $T$ ;
- (iii) For each  $s = l + 1, l + 2, \dots, l + h$ , decompose  $MST(F_s)$  by Lemma 1 into a set of trees of weight at most  $\frac{\lambda}{2}$  and put them into  $T$ .
- (iv) For each tree in  $T$ , double all its edges to obtain an Eulerian graph and shortcut the repeated vertices of the Eulerian tour of this graph to obtain a cycle. Put this cycle into  $S$ .

Step 3. Return the cycle cover  $S$ .

It can be seen that  $S$  is indeed a cycle cover of cost no greater than  $\lambda$ . We next show the approximation ratio of the algorithm is  $14/3$ .

Let  $OPT = k$  and  $C_1^*, C_2^*, \dots, C_k^*$  be the vertex-disjoint optimum cycles. We define light, heavy, bad cycles as in the last section and let  $k_l, k_h, k_b$  be the number of light, heavy and bad cycles, respectively. We redefine a *long edge* as an edge  $e$  of an optimum cycle with  $w(e) > \frac{\lambda}{6}$ . The internal long edges, external long edges and the function  $Ex(\cdot)$  are defined as before except that the graph  $G[\frac{\lambda}{5}]$  is replaced by  $G[\frac{\lambda}{6}]$ . For each optimum cycle  $C$ , we have  $Ex(C) \leq 5$  since  $w(C) \leq \lambda$ . For a bad cycle, besides the configurations shown in Fig. 1 which involve at most four external long edges, there are some extra configurations that involve five external long edges given in Fig. 2.

The bad cycles are partitioned into four categories, i.e.,  $B_1, B_2, B_3, B_4$ , where  $B_j (j = 1, 2, 3, 4)$  is defined as the set of bad cycles incident to exactly  $j$  light components. As shown in Figs. 1 and 2,  $B_1$  contains exactly the bad cycles with configurations (1)–(6) and (13)–(18) while  $B_2$  consists of the bad cycles with configurations (7)–(11) and (19)–(26).  $B_3$  is the set of bad cycles with configurations (12) and (27)–(31).  $B_4$  include the bad cycles with configurations (32). Set  $k_j = |B_j|$  for  $j = 1, 2, 3, 4$ . Therefore,  $k_b = \sum_{j=1}^4 k_j$ .



**Fig. 2.** The configurations of bad cycles with five external long edges.

The following lemma has been proven in [12].

**Lemma 6.** [12] *For any optimum cycle  $C$  with  $Ex(C) = 5$ , each external long edge of  $C$  is of weight no more than  $\frac{\lambda}{3}$ .*

It is the following critical observation that greatly simplifies the analysis of the algorithm.

**Lemma 7.** *For any bad cycle  $C \in B_3 \cup B_4$ , there exists two light components incident to  $C$  such that the distance between these two light components is at most  $\frac{\lambda}{3}$ .*

Next we construct a matching  $M_1$  in  $H_0$ . For each light cycle  $C$  incident to five light components, it holds that  $Ex(C) = 5$ . Let  $F_i$  and  $F_j$  be any two of the five light components connected by an external long edge of  $C$ . By Lemma 6 the weight of the external long edge is at most  $\frac{\lambda}{3}$ . Therefore,  $w_T(F_i) + w_T(F_j) + d(F_i, F_j) \leq \frac{\lambda}{12} + \frac{\lambda}{12} + \frac{\lambda}{3} = \frac{\lambda}{2}$  and  $(u_i, u_j) \in E(H_0)$ . We put  $(u_i, u_j)$  into  $M_1$  unless  $u_i$  or  $u_j$  has already been matched with some other vertex in  $M_1$ . For each bad cycle  $C \in B_3 \cup B_4$ , there exists two light components, say  $F_i, F_j$ , incident to  $C$  such that  $d(F_i, F_j) \leq \frac{\lambda}{3}$  by Lemma 7. As a result,  $(u_i, u_j) \in E(H_0)$ . Again we put  $(u_i, u_j)$  into  $M_1$  unless  $u_i$  or  $u_j$  has already been matched with

some other vertex in  $M_1$ . After that, the construction of  $M_1$  is completed and  $R = \{u_1, u_2, \dots, u_l\} \setminus V(M_1)$  is the set of vertices in  $H_0$  that are left exposed by  $M_1$ . Clearly,  $|R| = l - 2|M_1|$ .

By construction, each light cycle incident to five light components results in either one edge in  $M_1$  and at most three exposed vertices in  $R$  or at most four exposed vertices in  $R$ . And each light cycle incident to at most four light components generates at most four exposed vertices in  $R$ . Each bad cycle in  $B_j (j = 1, 2)$  is incident to exactly  $j$  light components and therefore derives at most  $j$  exposed vertices in  $R$ . For a bad cycle in  $B_3(B_4)$ , there exists two light components such that the distance between them is no more than  $\frac{\lambda}{3}$  and hence derives either one edge in  $M_1$  and at most one exposed vertex (two exposed vertices) in  $R$  or at most two (three) exposed vertices in  $R$ . So we have

$$\begin{aligned} l - |M_1| &= |M_1| + |R| \leq 4k_l + k_1 + 2k_2 + 2k_3 + 3k_4 \\ &= 4k_l + k_b + k_2 + k_3 + 2k_4, \end{aligned} \tag{5}$$

which implies

**Lemma 8.** *Algorithm MCCP3 generates at most  $4k_l + k_b + k_2 + k_3 + 2k_4$  trees in Step 2(i) and (ii).*

By this lemma, we may assume  $h \geq 1$  in the rest of this section. To bound the number of trees generated in Step 2(iii), we give an upper bound on the total weight of the minimum spanning trees of the heavy components.

**Lemma 9.**  $\sum_{s=l+1}^{l+h} w_T(F_s) \leq \frac{7}{6}k_h\lambda + \frac{5}{6}k_b\lambda - \frac{1}{6}(k_2 + 2k_3 + 3k_4)\lambda - \frac{h}{6}\lambda$ .

**Lemma 10.** *The number of trees generated in the Step 2(iii) of Algorithm MCCP3 is at most  $\frac{14}{3}k_h + \frac{10}{3}k_b - \frac{2}{3}(k_2 + 2k_3 + 3k_4)$ .*

By Lemmas 8 and 10 we deduce

**Lemma 11.**  $|S| = |T| \leq \frac{14}{3}k$ .

By this lemma and a simple analysis of the complexity of Algorithm MCCP3 we have

**Theorem 2.** *There is an  $O(n^3)$   $\frac{14}{3}$ -approximation algorithm for MCCP.*

## 5 $\frac{32}{7}$ -Approximation

In this section we present a  $32/7$ -approximation algorithm for MCCP that runs in  $O(n^5)$  time. To obtain the improved algorithm, we modify Algorithm MCCP3 in the following ways. First, we delete the edges with weight greater than  $\frac{\lambda}{7}$  instead of  $\frac{\lambda}{6}$  and redefine the light (heavy) components as components with the minimum spanning tree weight no more than  $\frac{3}{28}\lambda$  (greater than  $\frac{3}{28}\lambda$ ). Second, besides merging the minimum spanning trees of some light components we also

try to connect the minimum spanning trees of some other light components to the heavy components before decomposition operation.

As before let  $F_1, F_2, \dots, F_l$  and  $F_{l+1}, F_{l+2}, \dots, F_{l+h}$  be the light components and heavy components, respectively. Denote  $w_{\min}(F_i)$  with  $1 \leq i \leq l$  by the minimum weight of edges in  $G$  with one vertex in  $V(F_i)$  and the other vertex in  $\bigcup_{s=l+1}^{l+h} V(F_s)$ . To decide which minimum spanning trees of light components to be merged together and which minimum spanning trees of light components to be connected to heavy components, we solve a series of minimum weight perfect matching problems on the following graph  $H_{a,b}$  with weight function  $\tilde{w}(\cdot)$ .

**Definition 2.** Given nonnegative integers  $a, b$  with  $0 \leq a+b \leq l$ , the graph  $H_{a,b}$  has  $l+a+b$  vertices  $u_1, u_2, \dots, u_l, x_1, x_2, \dots, x_a, y_1, y_2, \dots, y_b$ . For  $1 \leq i \leq l$ , the vertex  $u_i$  corresponds to the light component  $F_i$ . If  $w_T(F_i) + w_T(F_j) + d(F_i, F_j) \leq \frac{\lambda}{2}$ , then add an edge  $(u_i, u_j)$  with  $\tilde{w}(u_i, u_j) = 0$  to  $H_{a,b}$ . For all  $i = 1, 2, \dots, l$  and  $j = 1, 2, \dots, a$ , add an edge  $(u_i, x_j)$  with  $\tilde{w}(u_i, x_j) = 0$  to  $H_{a,b}$ . For each  $i = 1, 2, \dots, l$ , if  $w_{\min}(F_i) \leq \frac{\lambda}{2}$ , then for each  $j = 1, 2, \dots, b$ , add an edge  $(u_i, y_j)$  with  $\tilde{w}(u_i, y_j) = w_T(F_i) + w_{\min}(F_i)$  to  $H_{a,b}$ . There are no other edges in  $H_{a,b}$ .

The improved algorithm is described as follows.

**Algorithm MCCP4**

Step 1. Delete all the edges with weight greater than  $\frac{\lambda}{7}$  in  $G$  to obtain  $G[\frac{\lambda}{7}]$  with light components  $F_1, F_2, \dots, F_l$  and heavy components  $F_{l+1}, F_{l+2}, \dots, F_{l+h}$ .

Step 2. For  $a = 0, 1, \dots, l, b = 0, 1, \dots, l - a$  set  $T_{a,b} := \emptyset$  and  $S_{a,b} := \emptyset$ .

If there is a minimum weight perfect matching  $M_{a,b}$  in  $H_{a,b}$ , find it and proceed to the following three substeps:

- (i) For any  $(u_i, u_j) \in M_{a,b}$ , merge  $MST(F_i)$  and  $MST(F_j)$  by an edge  $e$  in  $G$  corresponding to  $(u_i, u_j)$  to obtain a tree and put it into  $T_{a,b}$ ;
- (ii) For any  $(u_i, x_j) \in M_{a,b}$ , put  $MST(F_i)$  into  $T_{a,b}$ ;
- (iii) For any  $(u_i, y_j) \in M_{a,b}$ , connect  $MST(F_i)$  to some heavy component by the edge corresponding to  $w_{\min}(F_i)$ . This results in  $h$  modified heavy components  $F'_{l+1}, F'_{l+2}, \dots, F'_{l+h}$ . For each  $i = l+1, l+2, \dots, l+h$ , decompose  $MST(F'_i)$  by Lemma 1 into a set of trees of weight at most  $\frac{\lambda}{2}$  and put them into  $T_{a,b}$ .
- (iv) For each tree in  $T_{a,b}$ , double all its edges to obtain an Eulerian graph and shortcut the repeated vertices of the Eulerian tour of this graph to obtain a cycle. Put this cycle into  $S_{a,b}$ .

Step 3. Among all the nonempty  $S_{a,b}$ , return the one contains the minimum number of cycles.

It can be seen that each nonempty  $S_{a,b}$  is a cycle cover of cost no greater than  $\lambda$ . We next show the approximation ratio of the algorithm is  $32/7$ .

Let  $OPT = k$  and  $C_1^*, C_2^*, \dots, C_k^*$  be the vertex-disjoint optimum cycles. We define light, heavy, bad cycles as in the last section and let  $k_l, k_h, k_b$  be the number of light, heavy and bad cycles, respectively. We redefine a long edge as an edge  $e$  of an optimum cycle with  $w(e) > \frac{\lambda}{7}$ . The internal long edges, external long edges and the function  $Ex(\cdot)$  are defined as before except that the graph



$G[\frac{\lambda}{6}]$  is replaced by  $G[\frac{\lambda}{7}]$ . For each optimum cycle  $C$ , we have  $Ex(C) \leq 6$  since  $w(C) \leq \lambda$ . And by the triangle inequality each of the external long edges of  $C$  is of weight no more than  $\frac{\lambda}{2}$ .

An external long edge is called a *bridge* if it has one vertex in some light component and the other in some heavy component. The bad cycles are partitioned into six categories, i.e.,  $B_1, B_2, \dots, B_6$ .  $B_j(j = 1, 2, 4, 5)$  is defined as the set of bad cycles incident to exactly  $j$  light components. For a bad cycle  $C$  incident to exactly three light components, if only one of these light components is incident to some bridge,  $C$  is categorized as in  $B_3$ ; otherwise, i.e., at least two of these light components are incident to some bridge,  $C$  is categorized as in  $B_6$ . Set  $k_j = |B_j|$  for  $j = 1, 2, \dots, 6$ . Therefore,  $k_b = \sum_{j=1}^6 k_j$ .

**Lemma 12.** *For any optimum cycle  $C$  with  $Ex(C) = 6$ , each external long edge of  $C$  is of weight no more than  $\frac{2}{7}\lambda$ .*

**Lemma 13.** *For any bad cycle  $C \in B_3$  incident to each light component in  $Q = \{F_{i_1}, F_{i_2}, F_{i_3}\}$ , then there exists two light components  $F_{i_s}, F_{i_t} \in Q$  with  $w_T(F_{i_s}) + w_T(F_{i_t}) + d(F_{i_s}, F_{i_t}) \leq \frac{\lambda}{2}$ .*

**Lemma 14.** *For any bad cycle  $C \in B_4$  incident to each light component in  $Q = \{F_{i_1}, F_{i_2}, F_{i_3}, F_{i_4}\}$ , then there exists two light components  $F_{i_s}, F_{i_t} \in Q$  with  $w_T(F_{i_s}) + w_T(F_{i_t}) + d(F_{i_s}, F_{i_t}) \leq \frac{\lambda}{2}$  and some light component  $F_{i_r} \in Q \setminus \{F_{i_s}, F_{i_t}\}$  that is incident to some bridge.*

Next we construct a matching  $M_1$  in  $H_{a',b'}$  based on the optimum cycles. Rather than give an explicit value for  $a', b'$  in advance, we increase the values of  $a', b'$  whenever we need to match some vertex in  $\{u_1, u_2, \dots, u_l\}$  to some  $x$ -vertex or  $y$ -vertex in  $H_{a',b'}$ . After we complete the construction of  $M_1$ ,  $a', b'$  are defined as the number of  $x$ -vertex or  $y$ -vertex used in the process, respectively.

Initially  $M_1 = \emptyset$ . We add the edges to  $M_1$  in several steps. In each step we consider a particular type of optimum cycles and the components incident to it. In case we add an edge  $(u_i, u_j)$  to  $M_1$  to match a vertex  $u_i \notin V(M_1)$  to  $u_j \in V(M_1)$  (that is,  $u_j$  is already matched to some other vertex in the previous steps), we match  $u_i$  to a new  $x$ -vertex. If  $u_i, u_j \in V(M_1)$ , we actually add no edges to  $M_1$ .

To start with, we consider all light cycles. For each light cycle  $C$  incident to exactly six light components, we must have  $Ex(C) = 6$ . And we can choose four light components  $F_i, F_j, F_s, F_t$  such that  $F_i$  and  $F_j$  is connected by some external long edge  $e_1$  and  $F_s$  and  $F_t$  is connected by another external long edge  $e_2$ . By Lemma 12 each of  $w(e_1)$  and  $w(e_2)$  is at most  $\frac{2}{7}\lambda$ . Given the weight of the minimum spanning tree of each light component is at most  $\frac{3}{28}\lambda$ , we have  $w_T(F_i) + w_T(F_j) + d(F_i, F_j) \leq \frac{3}{28}\lambda \cdot 2 + \frac{2}{7}\lambda = \frac{\lambda}{2}$  and  $w_T(F_s) + w_T(F_t) + d(F_s, F_t) \leq \frac{\lambda}{2}$ . Therefore,  $(u_i, u_j), (u_s, u_t) \in E(H_{a',b'})$ . We put two edges  $(u_i, u_j), (u_s, u_t)$  into  $M_1$  and each of the two vertices corresponding to the remaining two light components incident to  $C$  is matched to a copy of  $x$ -vertex.

For each light cycle  $C$  incident to five light components, it holds that  $Ex(C) \geq 5$  and one of the external long edges, say  $e$ , has a length of at most  $\frac{\lambda}{5}$ .

Suppose  $F_i$  and  $F_j$  be the light components incident to  $e$ . We can add the edge  $(u_i, u_j)$  to  $M_1$  since  $w_T(F_i) + w_T(F_j) + d(F_i, F_j) \leq \frac{3}{28}\lambda \cdot 2 + w(e) \leq \frac{3}{14}\lambda + \frac{\lambda}{5} \leq \frac{\lambda}{2}$ . We also match each of the three vertices corresponding to the remaining three light components incident to  $C$  to a copy of  $x$ -vertex. For each light cycle incident to at most four light components, say  $F_{i_1}, \dots, F_{i_p}$  ( $p \leq 4$ ), we simply match  $F_{i_j}$  ( $j = 1, \dots, p$ ) to a copy of  $x$ -vertex.

For each bad cycle in  $B_1$ , let  $F_i$  be the only light component it is incident to. We simply match  $F_i$  to a copy of  $x$ -vertex. For each bad cycle in  $B_2$ , let  $F_i, F_j$  be the two light components it is incident to. One of these two light component, say  $F_i$ , has to be incident to some bridge. Then we match the vertex  $u_j$  corresponding to the other light component  $F_j$  to a copy of  $x$ -vertex. For each bad cycle in  $B_3$ , by Lemma 13 we can match two of the three vertices corresponding to the three light components incident to it and match the vertex corresponding to the remaining light component to a copy of  $x$ -vertex. For each bad cycle in  $B_4$ , by Lemma 14 we can match two of the four vertices corresponding to the four light components incident to it and match the third vertex corresponding to the third light component to a copy of  $x$ -vertex and guarantee that the last light component is incident to some bridge.

For each bad cycle  $C \in B_5$  incident to five light components  $F_{i_1}, F_{i_2}, \dots, F_{i_5}$ , it has to be incident to exactly one heavy component, say  $F_{i_6}$ , and  $Ex(C) = 6$ . We can assume without loss of generality that  $e_j$  ( $j = 1, 2, \dots, 6$ ) is the external long edge of  $C$  that connects  $F_{i_j}$  with  $F_{i_{j+1}}$  (set  $F_{i_7} = F_{i_1}$  for convenience). Similar to the case of light cycles incident to six light components, we can add edges  $(u_{i_1}, u_{i_2}), (u_{i_3}, u_{i_4})$  to  $M_1$  and match  $u_{i_5}$  to a copy of  $x$ -vertex. For each bad cycle in  $B_6$ , it is incident to three light components and two of them are incident to some bridge, we simply match the vertex corresponding to the remaining light component to a copy of  $x$ -vertex.

Now we have considered all the light and bad cycles. Let  $R$  be the set of vertices in  $\{u_1, u_2, \dots, u_l\}$  whose corresponding light components have not been matched so far. Set  $b' = |R|$ . Without loss of generality we suppose that  $R = \{u_{l-b'+1}, u_{l-b'+2}, \dots, u_l\}$ . It can be seen that for each  $i = 1, 2, \dots, b'$  there exists some bridges connecting  $F_{l-b'+i}$  with some heavy component and  $w_{\min}(F_{l-b'+i}) \leq \frac{\lambda}{2}$ . So we match  $F_{l-b'+i}$  with a copy of  $y$ -vertex. That is, we add the set  $M' = \{(u_{l-b'+1}, y_1), (u_{l-b'+2}, y_2), \dots, (u_l, y_{b'})\}$  of edges to  $M_1$ . This completes the construction of  $M_1$ . Clearly,  $M_1$  is a perfect matching in  $H_{a', b'}$ . Thus we have

**Lemma 15.** *There exists a perfect matching in  $H_{a', b'}$ .*

Denote by  $M'' = M_1 \setminus M'$ . By construction, each light cycle results in at most four edges in  $M''$ . And each bad cycle in  $B_1, B_2, B_3, B_4, B_5, B_6$  induces at most 1, 1, 2, 2, 3, 1 edges in  $M''$ , respectively. As a result,

$$\begin{aligned} |M''| &= \frac{l + a' - b'}{2} \leq 4k_l + k_1 + k_2 + 2(k_3 + k_4) + 3k_5 + k_6 \\ &= 4k_l + k_b + k_3 + k_4 + 2k_5, \end{aligned} \tag{6}$$

which implies

**Lemma 16.** For  $a = a'$  and  $b = b'$ , Algorithm *MCCP4* generates at most  $4k_l + k_b + k_3 + k_4 + 2k_5$  trees in Step 2(i) and (ii).

As before we may assume  $h \geq 1$  in the remaining discussion of this section. To give an upper bound on the number of trees generated in Step 2(iii), we first bound the total weight of the modified minimum spanning trees.

**Lemma 17.** For  $a = a'$  and  $b = b'$ ,

$$\sum_{s=l+1}^{l+h} w_T(F'_s) \leq \frac{8}{7}k_h\lambda + \frac{6}{7}k_b\lambda - (2k_3 + 2k_4 + 4k_5 + h) \cdot \frac{\lambda}{7}.$$

**Lemma 18.** For  $a = a'$  and  $b = b'$ , the number of trees generated in the Step 2(iii) of Algorithm *MCCP4* is at most  $\frac{32}{7}k_h + \frac{24}{7}k_b - \frac{8}{7}(k_3 + k_4 + 2k_5)$ .

By Lemmas 16 and 18 we deduce

**Lemma 19.**  $|S_{a',b'}| = |T_{a',b'}| \leq \frac{32}{7}k$ .

By this lemma and a simple analysis of the complexity of Algorithm *MCCP4* we have

**Theorem 3.** There is an  $O(n^5)$   $\frac{32}{7}$ -approximation algorithm for *MCCP*.

**Acknowledgements.** The authors are grateful to the anonymous referees for their helpful comments. This research is supported in part by the National Natural Science Foundation of China under grants numbers 11671135, 11701363 and the Fundamental Research Fund for the Central Universities under grant number 22220184028.

## References

1. Arkin, E.M., Hassin, R., Levin, A.: Approximations for minimum and min-max vehicle routing problems. *J. Algorithms* **59**, 1–18 (2006)
2. Campbell, A.M., Vandenbussche, D., Hermann, W.: Routing for relief efforts. *Transp. Sci.* **42**, 127–145 (2008)
3. Even, G., Garg, N., Koemann, J., Ravi, R., Sinha, A.: Min-max tree covers of graphs. *Oper. Res. Lett.* **32**, 309–315 (2004)
4. Farbstein, B., Levin, A.: Min-max cover of a graph with a small number of parts. *Discrete Optim.* **16**, 51–61 (2015)
5. Friggstad, Z., Swamy, C.: Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In: *The Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pp. 744–753 (2014)
6. Khani, M.R., Salavatipour, M.R.: Approximation algorithms for min-max tree cover and bounded tree cover problems. *Algorithmica* **69**, 443–460 (2014)
7. Nagarajan, V., Ravi, R.: Approximation algorithms for distance constrained vehicle routing problems. *Networks* **59**(2), 209–214 (2012)
8. Xu, W., Liang, W., Lin, X.: Approximation algorithms for min-max cycle cover problems. *IEEE Trans. Comput.* **64**(3), 600–613 (2015)

9. Xu, Z., Wen, Q.: Approximation hardness of min-max tree covers. *Oper. Res. Lett.* **38**, 408–416 (2010)
10. Xu, Z., Xu, L., Li, C.-L.: Approximation results for min-max path cover problems in vehicle routing. *Nav. Res. Logist.* **57**, 728–748 (2010)
11. Xu, Z., Xu, L., Zhu, W.: Approximation results for a min-max location-routing problem. *Discrete Appl. Math.* **160**, 306–320 (2012)
12. Yu, W., Liu, Z.: Improved approximation algorithms for some min-max cycle cover problems. *Theor. Comput. Sci.* **654**, 45–58 (2016)

# **Parameterized Algorithms**



# Parameterized Algorithms for Minimum Tree Cut/Paste Distance and Minimum Common Integer Partition

Jie You<sup>(✉)</sup>, Jianxin Wang<sup>(✉)</sup>, and Qilong Feng

School of Information Science and Engineering,  
Central South University, Changsha, China  
{youjie, jxwang}@csu.edu.cn

**Abstract.** We study two NP-hard problems formulated in computational biology. The minimum tree cut/paste distance problem asks for the minimum number of cut/paste operations we need to apply to transform a tree to another tree. The minimum common integer partition problem asks for a minimum-cardinality integer partition of a number that refines two given integer partitions of the same number. We give parameterized algorithms for both problems.

## 1 Introduction

In computational biology, we are given a set of data and asked to determine a “ground truth.” Although the ground truth under concern itself can never be known for sure, we do have many good approximations for it. Two families of problems have thus arisen naturally. The first is to find the *distance* between two given objects, which is defined to be the minimum number of (problem-specific) editing operations that one need to apply to transform one object to the other. Based on the type of objects and the allowed editing operations, there are a plethora of such problems studied in literature. The second is to find an object that is *closest* to the given set of objects. Again, for the same type of objects, one can define significantly different versions by using different distance measures and minimizing different objective functions, e.g., the total distances (which is equivalently to the average distance) or the maximum distance between the solution and all candidates. We approach these problems by considering their parameterized complexity. A problem, parameterized by  $k$ , is *fixed-parameter tractable (FPT)* if there is an algorithm running in time  $f(k) \cdot n^{O(1)}$ , where  $n$  is the input size and  $f$  is a computable function depending only on  $k$  [7].

The first problem we study is the minimum tree cut/paste distance problem, where the objects are directed, rooted, and unlabeled trees, and the editing operations are cuts (edge deletions) and pastes (edge additions). In the study of pedigree graphs (i.e., family trees), Kirkpatrick et al. [11] formulated this

---

This work is supported by the National Natural Science Foundation of China under Grants (61420106009, 61232001, 61472449, 61672536).

problem as a simplified case of the pedigree edit distance problem, and showed that both of them are NP-hard. The second problem is the minimum common integer partition problem, which is inspired by computational biology application including log assignment and DNA fingerprint assembly [3, 4, 9]. We say that a multiset  $A$  of positive integers is an *integer partition* of another multiset  $\{a_1, a_2, \dots, a_p\}$  of positive integers if the integers of  $A$  can form  $p$  sub-multisets such that the multiset union of those  $p$  sub-multisets is  $A$ , and the sum of the  $i$ th sub-multiset is  $a_i$  for  $1 \leq i \leq p$ . Given two multisets  $A_1$  and  $A_2$  of positive integers, the *minimum common integer partition* problem asks for an integer partition of both  $A_1$  and  $A_2$  that has the minimum cardinality.

The reduction from the minimum common integer partition problem to the minimum tree cut/paste distance problem given by Kirkpatrick et al. [11] implies that the former is equivalent to a very special case of the latter where both input trees are spiders (i.e., trees of which only the root nodes can have degrees larger than two). The size of an optimal solution for the former problem is the size of an optimal solution for the latter problem plus  $\min\{|A_1|, |A_2|\}$ .

A closely related problem is the minimum common string partition problem [4], which, given two strings  $S_1$  and  $S_2$  with the same length, asks whether they can be partitioned into the same set of substrings with the minimum cardinality. Although its name has a similar pattern as the minimum common integer partition problem, it is actually a distance problem. We can easily extend the definition of the minimum common string partition problem to define it in a similar way as the minimum common integer partition problem, i.e., instead of two strings, we take as input two multisets of strings. In so doing we can consider the minimum common integer partition problem as a special case of the minimum common string partition problem, where all letters in the strings are the same, and hence only their lengths matter. On the other hand, the minimum common string partition problem can be considered as a special case of the minimum tree cut/paste distance problem: There is an easy reduction from the minimum common string partition problem to the minimum tree cut/paste distance problem such that their optimal solutions have the same size.

Damaschke [5] first considered their parameterized complexity and presented an FPT algorithm for minimum common string partition with a combined parameter, i.e., the number  $k$  of substrings and the so-called “repetition number.” Jiang et al. [10] followed suit by considering another combined parameter,  $k$  and the maximum occurrence of a letter. Whether it is FPT parameterized by only  $k$  had remained open until recently resolved by Bulteau and Komusiewicz [2], who devised a very complicated algorithm. Since the aforementioned reduction from minimum common string partition to minimum tree cut/paste distance preserves the solution size  $k$ , an FPT algorithm for the latter with parameter  $k$  would immediately imply an FPT algorithm for the former with parameter  $k$ . This suggests that designing such an algorithm would be very challenging. Therefore, we step back and consider a combined parameter, which consists of  $k$  and the number  $\ell$  of leaves of the input trees. As usual,  $n$  denotes

the number of nodes in the input trees. Our first result is an FPT algorithm for the minimum tree cut/paste distance problem with this combined parameter.

**Theorem 1.** *The minimum tree cut/paste distance problem can be solved in  $2^{O(k \log(\ell+k))} \cdot n^{O(1)}$  time.*

The selection of the combined parameter is well justified if we turn to the special case where both input trees are spiders. In a nontrivial instance, the optimal solution is at least half of the number of leaves in the spiders (Proposition 9), so Theorem 1 already implies the fixed-parameter tractability of the minimum common integer partition with parameter  $k$ . With the parameter-preserved reduction from the minimum minimum common integer partition problem [11], this actually settles its fixed-parameter tractability. This algorithm obtained as such is nevertheless inefficient. Our second result is a direct and more efficient FPT algorithm for minimum common integer partition.

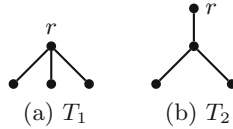
**Theorem 2.** *The minimum common integer partition problem can be solved in  $16^k \cdot n^{O(1)}$  time.*

The minimum common integer partition problem can be easily generalized to allow more than two multisets in the input. Our algorithm can also be easily adapted to the general setting. This general version has been extensively studied from the aspect of approximation algorithms [3, 13, 14]. The best-known approximation ratio is  $0.6t$  [13], where  $t$  is the number of multisets given in the input. In particular, this implies a 1.2-approximation for the version studied in this paper. The minimum tree cut/paste distance problem is a relatively new member of the large family of tree edit distance problems. Based on the particular applications, the trees might be rooted or unrooted, directed or undirected, labeled or unlabeled; another common restriction is to consider only binary trees. The editing operations considered include deleting nodes (or adding nodes viewed from the other way), relabeling nodes [12], and pruning/regrafting subtree [8]. See the survey of DasGupta et al. [6] for a in-depth coverage of related work. In the aforementioned reduction from minimum common string partition to minimum tree cut/paste distance, the resulting trees have a large number of leaves, and thus it is not clear how our algorithm can be used in minimum common string partition. We leave as an open problem to devise an FPT algorithm for minimum tree cut/paste distance with only parameter  $k$ , and a more efficient parameterized algorithm for minimum common string partition.

## 2 Preliminaries and Hardness Results

In this paper, all trees are unlabeled and rooted, hence, they have a special node called the root. Recall that in a rooted tree, there is a parent-child relationship between every pair of adjacent vertices, and thus the edge directions are implicitly decided—they are directed from the parent to the child. For notational convenience, without explicitly specifying the edge directions in the paper, we





**Fig. 1.** Illustration that directions matter.

will use the terminology of undirected trees. We start from a formal definition of the minimum tree cut/paste distance problem.

Minimum tree cut/paste distance  
*Input:* two rooted trees  $T_1$  and  $T_2$  with  $|V(T_1)| = |V(T_2)|$ , and a non-negative integer  $k$ ;  
*Task:* can  $T_1$  be isomorphic to  $T_2$  by applying at most  $k$  edge deletions and at most  $k$  edge additions to  $T_1$ ?

Note that the definition does not require us to preserve the roots. We say a tree  $T_1$  is *transformed* to another tree  $T_2$  if and only if  $T_1$  is isomorphic to  $T_2$  after applying some edge deletions and additions to it. Moreover, edge deletions (resp., additions) corresponds to edge *cut* (resp., *paste*) operations. An easy fact is that to transform a tree to another tree, we need exactly the same number of cut operations and paste operations. However, it is worth stressing that the edge directions do make the problem different from its undirected variant. To transform tree  $T_1$  to tree  $T_2$  in Fig. 1, for example, we need one cut and one paste operations, though these two trees are already equivalent ignoring directions.

Now let us consider a problem that is closely related to the minimum tree cut/paste distance problem, called the minimum common integer partition problem. Let  $n$  be a positive integer. A multiset  $A = \{a_1, a_2, \dots, a_p\}$  of  $p$  positive integers is an *integer partition* of  $n$  if  $\sum_{i=1}^p a_i = n$ , where  $p$  is denoted by  $|A|$ . Abusing notation, we say that another integer partition  $A'$  is an *integer partition of  $A$*  if the integers of  $A'$  can form  $p$  sub-multisets, i.e.,  $\{A'_1, \dots, A'_p\}$ , such that (i) for any  $i \neq j \in [1, p]$ ,  $A'_i \cap A'_j = \emptyset$ ; (ii)  $\bigcup_{i \in [1, p]} A'_i = A'$ ; and (iii) for any  $i \in [1, p]$ ,  $A'_i$  is an integer partition of  $a_i$ . Note that  $|A'| \geq |A|$  since  $A$  is an integer partition of itself. Given two integer partitions  $A_1$  and  $A_2$  of the same integer  $n$ , the *minimum common integer partition* problem asks for a common integer partition of  $A_1$  and  $A_2$  with the minimum cardinality.

Minimum common integer partition  
*Input:* two integer partitions  $A_1$  and  $A_2$  of integer  $n$ , and a nonnegative integer  $k$ ;  
*Task:* is there an integer partition  $A$  of both  $A_1$  and  $A_2$  such that  $|A| \leq \min\{|A_1|, |A_2|\} + k$ ?

Kirkpatrick et al. [11] have presented the following reduction from minimum common integer partition to minimum tree cut/paste distance. Let  $(\{a_1, a_2,$

$\dots, a_p\}$ ,  $\{b_1, b_2, \dots, b_q\}, k_{\text{cip}}$ ) be a given instance of minimum common integer partition. We construct tree  $T_1$  (resp.,  $T_2$ ) by adding for each  $i$  with  $1 \leq i \leq p$  (resp.,  $1 \leq j \leq q$ ) a distinct path from its root  $r_1$  (resp.,  $r_2$ ) of length  $a_i$  (resp.,  $b_j$ ). Note that both  $T_1$  and  $T_2$  have  $1 + \sum_{i=1}^p a_i$  nodes; moreover,  $\ell(T_1) = p$  and  $\ell(T_2) = q$ , where  $\ell(T_i)$  is the number of leaves in tree  $T_i$ .

**Proposition 1** ([11]). *The instance  $(\{a_1, a_2, \dots, a_p\}, \{b_1, b_2, \dots, b_q\}, k_{\text{cip}})$  is a YES-instance of minimum common integer partition if and only if  $(T_1, T_2, k_{\text{cip}})$  is a YES-instance of minimum tree cut/paste distance.*

A node different from the root is a *leaf* if its degree is one—the root itself, regardless its degree, is not considered a leaf. Let  $\ell(T)$  denote the number of leaves in tree  $T$ . A tree is called a *spider* if all nodes different from the root and leaves have degree precisely two. Since both  $T_1$  and  $T_2$  constructed above are spider trees, the minimum tree cut/paste distance problem is already NP-hard on spider trees.

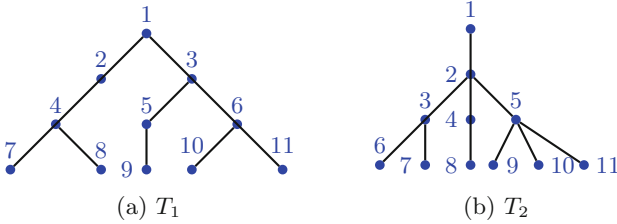
### 3 An Algorithm for Minimum Tree Cut/Paste Distance

We start from the parameterized version of the minimum tree cut/paste distance problem with a combined parameter that consists of the editing number  $k$  and  $\ell := \max\{\ell(T_1), \ell(T_2)\}$ . Recall  $\ell(T)$  is the number of leaves of tree  $T$ . It is easy to use induction to verify that  $k$  cut and  $k$  paste operations can only change the number of leaves by at most  $k$ . Hence if  $T_2$  has less than  $\ell(T_1) - k$  or more than  $\ell(T_1) + k$  leaves, then we can safely return “NO.” We may henceforth assume that  $\ell(T_1) - k \leq \ell(T_2) \leq \ell(T_1) + k$ .

Let  $T$  be a rooted and unlabeled tree, and  $U$  be a set of nodes of it. We use  $T[U]$  to denote the sub-forest (or sub-tree) of  $T$  induced by the nodes of  $U$ . Recall that node  $v$  is an *ancestor* of node  $u$  if  $v$  is visited by the path from the root of  $T$  to  $u$ ; hence  $v$  is an ancestor of itself. A node of  $T$  that has at least two children is called a *joint*. If all the edges from every joint to its children are removed, then every node will have its degree at most two; in other words, we end with a collection of paths, and such a path is called a *pipelines* of  $T$ . We take the liberty to consider a pipeline as a set of nodes; in this sense, all those pipelines of  $T$  makes a partition of  $V(T)$ , marked as  $\mathcal{P}(T)$ . For a node set  $X$  of  $V(T)$ , we use  $\mathcal{P}(T)|_X = \{X \cap V_i \mid \forall V_i \in \mathcal{P}(T), X \cap V_i \neq \emptyset\}$  to denote a partition of node set  $X$ . Specially, if  $X$  is a sub-set of nodes of pipeline  $p$  of  $T$ , and  $T[X]$  is connected (namely, a consecutive path), then we call  $T[X]$  a *segment* of pipeline  $p$ . Note that a pipeline of a tree  $T$  may become a segment of some pipeline of tree  $T'$ , where  $T'$  is a tree that is obtained by cutting some edges from  $T$ .

Figure 2 shows an instance  $(T_1, T_2, k = 2)$  for the minimum tree cut/paste distance problem, where both  $T_1$  and  $T_2$  have 11 nodes. The pipelines of  $T_1$  are  $\{1\}, \{2, 4\}, \{7\}, \{8\}, \{3\}, \{5, 9\}, \{6\}, \{10\}$ , and  $\{11\}$ , the pipelines of  $T_2$  are  $\{1, 2\}, \{3\}, \{4, 8\}, \{5\}, \{6\}, \{7\}, \{9\}, \{10\}$ , and  $\{11\}$ . There are four joints in  $T_1$ , which are 1, 3, 4, and 6, and three joints in  $T_2$ , which are 2, 3, and 5. One optimal solution for the instance is that edges  $\langle 1, 2 \rangle$  and  $\langle 2, 4 \rangle$  are cut, and edges  $\langle 4, 3 \rangle$

and  $\langle 4, 2 \rangle$  are added. Let  $X = \{2\}$ , then  $T_1[X]$  is a segment of the pipeline of  $T_1$  whose node set is  $\{2, 4\}$ .



**Fig. 2.** An instance  $(T_1, T_2, k = 2)$  for the minimum tree cut/paste distance problem.

Let  $p$  be a path of a tree  $T$ . The *length* of  $p$ , denoted by  $len(p)$ , is defined to be the number of nodes in  $V(p)$ . Since  $T$  is rooted, there is only one node that is an ancestor of all nodes in  $V(p)$ , which is called the *top* node of  $p$ , marked as  $t(p)$ . Moreover, there is only one node that all nodes in  $V(p)$  are its ancestors, which is called the *bottom* node of  $p$ , marked as  $b(p)$ . Note that the bottom node and top node can be identical if  $p$  consists of only one node. Specially, if  $p$  is a pipeline of  $T$  and the root of  $T$  is in  $V(p)$ , then we call  $p$  is a *root-pipeline*. A well-known fact states that the number of joints in a tree is at most  $\ell(T) - 1$ , thus, there are at most  $2\ell(T) - 1$  pipelines of a tree.

Since the tree  $T_1$  will become a forest during edge deletion process, we will consider a slightly more generalized version of the minimum tree cut/paste distance problem, the *minimum cut/paste distance between a forest and a tree*, which is formally defined as follows:

Minimum cut/paste distance between a forest and a tree  
*Input:* A forest  $F$ , a tree  $T$ , and two nonnegative integers,  $k_-$  and  $k_+$ , where  $|V(F)| = |V(T)|$ , and  $k_+ = k_- + |F| - 1$ ;  
*Task:* can  $F$  be isomorphic to  $T$  by applying at most  $k_-$  edge deletions and at most  $k_+$  edge additions to  $F$ ?

A solution to the minimum cut/paste distance between a forest and a tree problem can be equivalently viewed as a one-to-one mapping  $\phi : V(F) \mapsto V(T)$ : For an edge (resp., a non-edge)  $uv$  of  $F$ , it is cut (resp., added) by the solution if  $\phi(u)\phi(v)$  is a non-edge (resp., an edge). Moreover, we say that  $u$  (resp,  $v$ ) is *touched* (by a cut/paste operation) if there is an edge (resp., a non-edge)  $uv$  that participates in a cut/paste operation. For the instance showed in Fig. 2, we have the mapping  $\phi$  corresponding to the given optimal solution is that  $\phi(1) = 1, \phi(2) = 11, \phi(3) = 2, \phi(4) = (5), \phi(5) = 4, \phi(6) = 3, \phi(7) = 9, \phi(8) = (10), \phi(9) = 8, \phi(10) = 6,$  and  $\phi(11) = 8$ .

Let  $(T_1, T_2, k)$  be an instance of the minimum tree cut/paste distance problem. Now we fix a hypothetical solution  $E_- \uplus E_+$  applied to  $T_1$ ; it can be two

arbitrary sets of  $k$  edges if  $(T_1, T_2, k)$  is a NO-instance. Considering the special sub-set  $E_-^*$  of edges in  $E_-$ , which connect joints and their children in  $T_1$ . For sure,  $|E_-^*|$  should be bounded by  $k$ . Hence, we can correctly guess the  $E_-^*$  from  $E(T_1)$  in  $O((2\ell(T_1) - 1)^k)$ -time. Now we have the following proposition.

**Proposition 2.** *For any optimal solution, there is a unique set  $E_-^*$  of edges in  $E(T_1)$  such that no child of any joint of  $F_1 = T_1 - E_-^*$  will be cut-off from its parent. Moreover, the set  $E_-^*$  of edges can be obtained in  $2^{O(k \log \ell)} n^{O(1)}$ -time.*

Let  $E_-^* \subseteq E(T_1)$  be a set of edges satisfying Proposition 2 for some optimal solution, and  $F_1 = T_1 - E_-^*$ . Now we have  $(F_1, T_2, k_-, k_+)$  is an instance of the minimum cut/paste distance between a forest and a tree problem, where  $k_- = k - |E_-^*|$  and  $k_+ = k$ . We say an edge is *contained* in a pipeline if and only if the two nodes of this edge are both in the pipeline. Next we will show that for any optimal solution, all nodes touched by the solution are contained in at most  $c_1 k$  pipelines of  $F_1$ . Moreover, the images of all nodes of those pipelines of  $F_1$  form at most  $c_2 k$  pipelines of  $T_2$ , where  $c_1$  and  $c_2$  are constants. Recall that a set  $V$  of nodes forms a set of pipelines of  $T$  if  $\mathcal{P}(T)|_V \subseteq \mathcal{P}(T)$ .

**Lemma 1.** *For some optimal solution of  $(F_1, T_2, k_-, k_+)$ , there is a set  $P$  of nodes in  $V(F_1)$ , such that (i)  $|\mathcal{P}(F_1)|_P| \leq 2k + 1$ , (ii)  $\mathcal{P}(F_1)|_P \subseteq \mathcal{P}(F_1)$ , and (iii) only nodes of  $P$  are touched by the solution. Moreover, let  $\phi$  be the mapping corresponding to the optimal solution, and  $Q$  denote node set  $\bigcup_{v \in P} \phi(v)$ . Then  $\mathcal{P}(T_2)|_Q \subseteq \mathcal{P}(T_2)$ , and  $|\mathcal{P}(T_2)|_Q| \leq 4k + 1$ .*

**Corollary 1.** *Given a YES-instance  $(F_1, T_2, k_-, k_+)$ . There is an algorithm that outputs a family  $\mathcal{X}$  of set pairs,  $\{(P_1, Q_1), \dots, (P_h, Q_h)\}$  with  $|\mathcal{X}| = 2^{O(k \log \ell)}$ , in  $2^{O(k \log \ell)} n^{O(1)}$ -time, and at least one pair  $(P, Q) \in \mathcal{X}$  satisfies Lemma 1.*

Let  $(P, Q) \in \mathcal{X}$  be a pair of node sets from  $V(F_1)$  and  $V(T_2)$ , respectively, satisfying Lemma 1 for some optimal solution. As the assumption on  $F_1$ , there are some nodes in  $P$  that are joints of  $F_1$ , and their images in  $T_2$  should be joints as well. Moreover, the children of those joints in  $F_1$  will not be cut off. Hence, their images should be non-root top nodes in  $T_2$ . Let  $J$  be the set of nodes in  $P$ , which is defined as  $\{v \mid \forall v \text{ is a joint of } F_1, v \in P\}$ . Let  $D$  be the set of nodes in  $P$  which is defined as  $\{v \mid \forall v \text{ is a non-root top node of } F_1, v \in P\}$ . Then we guess a mapping for  $J \cup D$ .

**Proposition 3.** *Let  $(P, Q) \in \mathcal{X}$  be a pair of node sets from  $V(F_1)$  and  $V(T_2)$ , respectively, satisfying Lemma 1 for some optimal solution, and  $\phi$  be the corresponding mapping. Let  $J \subseteq P$  and  $D \subseteq P$  be the node sets denoted above. Then there is an algorithm that outputs a family of set pairs  $\mathcal{M} = \{(J'_1, D'_1), \dots, (J'_x, D'_x)\}$  with  $|\mathcal{M}| = 2^{O(k \log k)}$  in  $2^{O(k \log k)} n^{O(1)}$ -time, such that there is a set pair  $(J', D') \in \mathcal{M}$  with  $\bigcup_{v \in J} \phi(v) = J'$  and  $\bigcup_{v \in D} \phi(v) = D'$ .*

Let  $(J', D')$  be a set pair of  $\mathcal{M}$  satisfying Proposition 3. Let  $P^*$  and  $Q^*$  denote  $P/(J \cup D)$  and  $Q/(J' \cup D')$ , respectively. Next we will show how to find a mapping from  $P^*$  to  $Q^*$ , which corresponds to some optimal solution. Let  $\mathcal{A} = \mathcal{P}(F_1)|_{P^*}$

and  $\mathcal{B} = \mathcal{P}(T_2)|_{Q^*}$  be two partitions of  $P^*$  and  $Q^*$ , and  $\phi$  be the mapping for the optimal solution. Now we introduce an auxiliary edge-weighted bipartite graph  $G = (L, R; E)$  under the mapping  $\phi$ , where vertices of  $L$  and  $R$  correspond to elements in  $\mathcal{A}$  and  $\mathcal{B}$ , respectively. For any two vertices  $v \in L$  and  $u \in R$ , let  $A_v \in \mathcal{A}$  and  $B_u \in \mathcal{B}$  be their corresponding node sets. Obviously,  $A_v$  (resp.,  $B_u$ ) forms either a pipeline or a segment of a pipeline of  $F_1$  (resp.,  $T_2$ ). Hence, let  $p_v$  and  $q_u$  denote the corresponding pipelines in  $F_1$  and  $T_2$ , respectively. If  $u$  and  $v$  have an edge incident to them in  $G$ , then we say  $\phi$  maps a segment of  $p_v$  to a segment of  $q_u$ . Moreover, the weight  $w(uv)$  is denoted by the length of the segments.

Let  $p$  and  $q$  be paths of  $F_1$  and  $T_2$  respectively. The *correlation* sets of  $p$  and  $q$  under some mapping  $\phi$  are two non-empty nodes,  $X \subseteq V(p)$  and  $Y \subseteq V(q)$ , where for every node  $v \in V(p)$  with  $\phi(v) \in V(q)$ , node  $v$  is in  $X$  and  $Y = \bigcup_{u \in X} \phi(u)$ . Moreover, we say some node set  $X$  of a pipeline is *orderly* mapped to a node set  $Y$  of another pipeline if for any edge  $uv$  of  $F_1[X]$ ,  $\phi(u)\phi(v)$  is an edge in  $T_2$  as well. For example, let  $a_1a_2 \cdots a_t$  and  $b_1b_2 \cdots b_t$  denote two paths,  $p_1$  and  $p_2$ , of  $F_1$  and  $T_2$  respectively. Node set  $V(p_1)$  is orderly mapped to node set  $V(p_2)$  if  $\phi(a_i) = b_i$  for  $1 \leq i \leq t$ .

**Proposition 4.** *There is some optimal solution with mapping  $\phi$ , such that for the correlation sets  $X$  and  $Y$  of two pipelines  $p \in \mathcal{P}(F_1)$  and  $q \in \mathcal{P}(T_2)$ ,  $X$  and  $Y$  form two segments of  $p$  and  $q$  respectively, and  $X$  is orderly mapped to  $Y$ .*

Roughly say, Proposition 4 suggests us to consider such an optimal solution that maps a node set  $X$  of some pipeline in  $F_1$  to a node set  $Y$  of some pipeline in  $T_2$ , and  $X, Y$  form two segments of their pipelines. Moreover, the mapping between the two segments are orderly mapped. Therefore, we may henceforward assume that if we say a node set of pipeline  $p$  is mapped to a node set of pipeline  $q$ , then it means that a segment of  $p$  is orderly mapped to a segment of  $q$ .

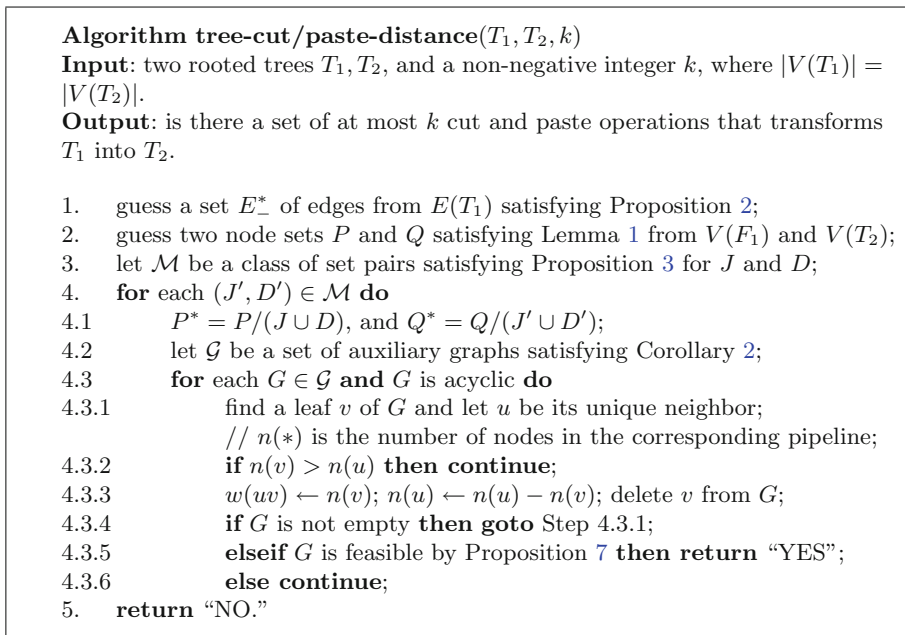
**Proposition 5.** *For any optimal solution, the auxiliary graph  $G$  contains no more than  $4k + 1$  edges.*

**Corollary 2.** *For any pair of node sets  $P^*$  and  $Q^*$  denoted above, there is a family  $\mathcal{G}$  of auxiliary graphs, where  $|\mathcal{G}| = O((8k)^{8k})$  and  $\mathcal{G}$  can be found in  $O((8k)^{8k})$ -time.*

**Proposition 6.** *There is an auxiliary graph  $G \in \mathcal{G}$  that is acyclic, if the instance given is a YES-instance.*

Now we have the following proposition, with which we can obtain a solution to the minimum tree cut/paste distance problem from some auxiliary graph in polynomial time. We say an auxiliary graph is *feasible* if the corresponding solution of it is a feasible solution (namely, the cost is bounded by  $k$ , and  $T_1$  can be transformed into  $T_2$ ), and it is *infeasible* otherwise.

**Proposition 7.** *For an acyclic auxiliary graph  $G \in \mathcal{G}$ , we either get a solution or answer the auxiliary graph  $G$  is infeasible, in polynomial time.*



**Fig. 3.** The algorithm for minimum tree cut/paste distance.

Putting them together, we obtain the algorithm for minimum tree cut/paste distance (Fig. 3); here by “guess” we mean to find the desired object by enumeration. We are now ready to prove Theorem 1.

*Proof (Proof of Theorem 1).* We use the algorithm described in Fig. 3 for illustration. We verify first its correctness. Step 1 finds all those edges in  $T_1$  that are leaving joints for some optimal solution. Step 2 finds two “correct” node sets,  $P$  and  $Q$ , from  $F_1$  and  $T_2$  respectively, which is done via enumerating all possible pairs of such node sets. For a pair  $(P, Q)$ , step 3 enumerates all possible  $(J', D')$  for  $(J, D)$ , and the mapping between them. Since Proposition 3, at least one of those mapping will match the solution. Step 4.2 builds a family  $\mathcal{G}$  of auxiliary graphs for the nodes in  $P/(J \cup D)$  and  $Q/(J' \cup D')$ , and step 4.3 gives the mapping for each auxiliary graphs of  $\mathcal{G}$ . According to Propositions 5 and 6, there is acyclic  $G \in \mathcal{G}$  that matches the solution. Step 4.3 either obtains a feasible solution or realizes the solution basing this auxiliary graph is infeasible, thus, it will continue to try next auxiliary graph. The correctness of step 4.3 is guaranteed by Proposition 7. If we try all possible  $G$  in  $\mathcal{G}$ , and we do not find a feasible solution, then we say it is a NO-instance. Otherwise, we return a feasible solution.

We now analyze the running time. Step 1 can be finished in  $2^{O(k \log \ell)} \cdot n^{O(1)}$ -time since Proposition 2. Step 2 can be done in  $2^{O(k \log \ell)} \cdot n^{O(1)}$ -time according to Corollary 1. Step 3 can be done in  $2^{O(k \log k)} \cdot n^{O(1)}$ -time since Proposition 3.

Step 4.1 can be done in constant time, and step 4.2 can be done in  $2^{O(k \log k)} \cdot n^{O(1)}$  since Corollary 2. Step 4.3 can be finished in polynomial time since Proposition 7. Together all steps, we have the algorithm can be finished in  $2^{O(k \log(\ell+k))} \cdot n^{O(1)}$ .

We would like to remark that there is a massive constant in the exponential part  $O(k \log(\ell+k))$ . Moreover, the analysis of the time complexity is far from tight, and it is done in this way for the purpose of simplicity.

## 4 An Algorithm for Minimum Common Integer Partition

We now turn to the minimum common integer partition problem. Instead of presenting an algorithm for it directly, we work on the minimum tree cut/paste distance problem restricted to spider trees. Together with the reduction by Kirkpatrick et al. [11] (Proposition 1), a parameterized algorithm for minimum tree cut/paste distance implies a parameterized algorithm for minimum common integer partition. Recall that in a spider tree  $T$  only the root  $r$  of  $T$  has degree more than two. Let  $d$  be the degree of  $r$  and  $\{v_1, \dots, v_d\}$  be the children of  $r$ , then we call each edge  $rv_i$  a *top-edge* of  $T$ . Let  $(T_1, T_2, k)$  be an instance of the minimum tree cut/paste distance problem on spider trees. The degrees of the roots  $r_1$  of  $T_1$  and  $r_2$  of  $T_2$  are  $d_1$  and  $d_2$  respectively. To avoid triviality, we may assume  $|V(T_1)| = |V(T_2)|$ , and without loss of generality  $d_1 \geq d_2 \geq 3$ .

Using greedy algorithm, it is easy to devise a solution with at most  $d_1 - 1$  cut operations, which maps the root node  $r_1$  of  $T_1$  to the root node  $r_2$  of  $T_2$ .

**Lemma 2.** *There always exists an optimal solution for  $(T_1, T_2, k)$  such that (1) it needs at most  $d_1 - 1$  cut operations, and (2) maps  $r_1$  to  $r_2$ .*

This algorithm given above is indeed the 2-approximation algorithm mentioned by Woodruff [14]. It provides an upper bound for the optimal solutions. Henceforward, let us assume that all optimal solutions mentioned following both map  $r_1$  to  $r_2$ . The remaining thing is to show a mapping from nodes in  $V(T_1)/\{r_1\}$  to nodes in  $V(T_2)/\{r_2\}$ . Let  $\mathcal{P}$  and  $\mathcal{Q}$  denote  $\{p_1, \dots, p_{d_1}\}$  and  $\{q_1, \dots, q_{d_1}\}$ , which form partitions of node sets  $V(T_1)/\{r_1\}$  and  $V(T_2)/\{r_2\}$ . We call them the *non-trivial pipeline sets* of  $T_1$  and  $T_2$ , respectively. Obviously, every pipeline in the non-trivial pipeline set of any spider tree is not a root-pipeline.

Next we will show that if there are pipelines  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$  that have the same length, then we may delete both of them, and consider the remaining instance  $(T_1 - V(p), T_2 - V(q), k)$ .

**Proposition 8.** *Let  $(T_1, T_2, k)$  be an instance of minimum tree cut/paste distance on spider trees, and  $\mathcal{P}$  and  $\mathcal{Q}$  be the non-trivial pipeline sets. If there are pipelines  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$  that have the same length, then we have  $(T_1, T_2, k)$  is a YES-instance if and only if  $(T_1 - V(p), T_2 - V(p), k)$  is a YES-instance.*

Assume that no pipelines  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$  have the same length for instance  $(T_1, T_2, k)$ . Since one cut and paste operations only change the lengths of two

pipelines in  $\mathcal{P}$ . Therefore, we have the following proposition is trivially true. Recall that  $d_1 \geq d_2 \geq 3$ .

**Proposition 9.** *Let  $(T_1, T_2, k)$  be an instance of minimum tree cut/paste distance on spider trees such that no pipelines  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$  have the same length, where  $\mathcal{P}$  and  $\mathcal{Q}$  are the non-trivial pipeline sets of  $T_1$  and  $T_2$ , respectively. Then any optimal solution has size at least  $d_1/2$ .*

**Dynamic Programming.** Now we begin to show the dynamic programming algorithm. Let  $(T_1, T_2, k)$  be an instance of minimum tree cut/paste distance on spider trees such that no pipelines  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$  have the same length, where  $\mathcal{P}$  and  $\mathcal{Q}$  are the non-trivial pipeline sets of  $T_1$  and  $T_2$ , respectively. There are  $2^{d_1+d_2+1}$  slots in the dynamic programming table, and each slot can be presented by a tuple  $(\mathcal{P}', \mathcal{Q}', f)$ , where  $\mathcal{P}' \subseteq \mathcal{P}$ ,  $\mathcal{Q}' \subseteq \mathcal{Q}$ , and  $f \in \{0, 1\}$ . For a slot  $S = (\mathcal{P}', \mathcal{Q}', f)$ , we use  $\Delta(S)$  to denote  $|\mathcal{P}'| - |\mathcal{Q}'|$ , and  $\text{opt}(S)$  to mark the minimum number of cut operations this slot used, where  $P' = \bigcup_{p \in \mathcal{P}'} V(p)$  and  $Q' = \bigcup_{q \in \mathcal{Q}'} V(q)$ . A slot is *good* to some optimal solution, if it can reach the optimal solution.

**Lemma 3.** *There is an optimal solution with the mapping  $\phi$  for  $(T_1, T_2, k)$ , such that if a slot  $S = (\mathcal{P}', \mathcal{Q}', f)$  is good to the solution, then (1) when  $\Delta(S) > 0$ , only a segment with length  $|\Delta(S)|$  of some pipeline in  $\mathcal{P}'$  is unmapped; (2) when  $\Delta(S) < 0$ , only a segment with length  $|\Delta(S)|$  of some pipeline in  $\mathcal{Q}'$  is unmapped; and (3) when  $f = 1$ , the unmapped segment contains the top node of its pipeline, otherwise, it contains the bottom node.*

Now we show the dynamic programming equations. Initially, let  $S = (\emptyset, \emptyset, *)$  and  $\text{opt}(S)$  be 0. For any slot  $S = (\mathcal{P}', \mathcal{Q}', f)$  satisfying one of the following x cases, we use the corresponding dynamic programming equations, for all  $p \in \mathcal{P}/\mathcal{P}'$  and  $q \in \mathcal{Q}/\mathcal{Q}'$ .

**Case-1:** If  $\Delta(S) = 0$ , then we use  $S_0$  to denote  $(\mathcal{P}' \cup \{p\}, \mathcal{Q}' \cup \{q\}, 0)$ , and

$$\begin{aligned} \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S)\} & \Delta(S) = 0, \text{len}(p) < \text{len}(q) \\ \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S) + 1\} & \Delta(S) = 0, \text{len}(p) > \text{len}(q) \end{aligned} \quad (1)$$

**Case-2:** If  $\Delta(S) > 0$  and  $f = 0$ , then we use  $S_1, S_0$  and  $S^*$  to denote  $(\mathcal{P}', \mathcal{Q}' \cup \{q\}, 1)$ ,  $(\mathcal{P}', \mathcal{Q}' \cup \{q\}, 0)$ , and  $(\mathcal{P}' \cup \{p\}, \mathcal{Q}' \cup \{q\}, 0)$ , respectively, and

$$\begin{aligned} \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S) + 1\} & \text{len}(q) < \Delta(S) \\ \text{opt}(S_1) &= \min\{\text{opt}(S_1), \text{opt}(S)\} & \text{len}(q) \geq \Delta(S) \\ \text{opt}(S^*) &= \min\{\text{opt}(S^*), \text{opt}(S) + 1\} & \text{len}(p) < \text{len}(q) \leq \Delta(S) \end{aligned} \quad (2)$$

**Case-3:** If  $\Delta(S) > 0$  and  $f = 1$ , then we use  $S_0$  to denote  $(\mathcal{P}', \mathcal{Q}' \cup \{q\}, 0)$ , and

$$\begin{aligned} \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S) + 1\} & \text{len}(q) < \Delta(S) \\ \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S)\} & \text{len}(q) \geq \Delta(S) \end{aligned} \quad (3)$$



**Case-4:** If  $\Delta(S) < 0$  and  $f = 0$ , then we use  $S_1, S_0$  and  $S^*$  to denote  $(\mathcal{P}' \cup \{p\}, \mathcal{Q}', 1)$ ,  $(\mathcal{P}' \cup \{p\}, \mathcal{Q}', 0)$ , and  $(\mathcal{P}' \cup \{p\}, \mathcal{Q}' \cup \{q\}, 0)$ , respectively, and

$$\begin{aligned} \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S)\} & \text{len}(q) \leq |\Delta(S)| \\ \text{opt}(S_1) &= \min\{\text{opt}(S_1), \text{opt}(S) + 1\} & \text{len}(p) > |\Delta(S)| \\ \text{opt}(S^*) &= \min\{\text{opt}(S^*), \text{opt}(S) + 1\} & \text{len}(q) < \text{len}(p) \leq |\Delta(S)| \end{aligned} \quad (4)$$

**Case-5:** If  $\Delta(S) < 0$  and  $f = 1$ , then we use  $S_0$  to denote  $(\mathcal{P}' \cup \{p\}, \mathcal{Q}', 0)$ , respectively, and

$$\begin{aligned} \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S)\} & \text{len}(q) \leq |\Delta(S)| \\ \text{opt}(S_0) &= \min\{\text{opt}(S_0), \text{opt}(S) + 1\} & \text{len}(q) > |\Delta(S)| \end{aligned} \quad (5)$$

**Proposition 10.** *Case 1–5 are safe for the minimum tree cut/paste distance problem on spider trees.*

*Proof (Proof of Theorem 2).* According to the dynamic programming equations listed above, we add pipelines for each slot, and it can be done in  $O(d_1 d_2)$ -time. Moreover, the correctness of the dynamic programming algorithm is kept by Lemma 3 and Proposition 10. Since Proposition 9,  $k \geq d_1/2 \geq d_2/2$ . The running time of the dynamic programming algorithm is bounded by  $O(2^{2k+2k+1} k^2) = O(16^k) n^{O(1)}$ .

Combined with the reduction of Kirkpatrick et al. [11], the dynamic programming algorithm implies an  $O(16^k)$ -time algorithm for the minimum common integer partition problem.

## References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Boston (1974)
2. Bulteau, L., Komusiewicz, C.: Minimum common string partition parameterized by partition size is fixed-parameter tractable. In: Chekuri, C. (ed.) Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 102–121. SIAM (2014). <https://doi.org/10.1137/1.9781611973402.8>
3. Chen, X., Liu, L., Liu, Z., Jiang, T.: On the minimum common integer partition problem. ACM Trans. Algorithms, **5**(1), article no, 12 (2008). <https://doi.org/10.1145/1435375.1435387>
4. Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: Assignment of orthologous genes via genome rearrangement. IEEE/ACM Trans. Comput. Biol. Bioinform. **2**(4), 302–315 (2005). <https://doi.org/10.1145/1100863.1100950>
5. Damaschke, P.: Minimum common string partition parameterized. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS, vol. 5251, pp. 87–98. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-87361-7\\_8](https://doi.org/10.1007/978-3-540-87361-7_8)
6. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Wang, L., Zhang, L.: Computing distances between evolutionary trees. In: Pardalos, P.M., Du, D.-Z., Graham, R.L. (eds.) Handbook of Combinatorial Optimization, pp. 747–781. Springer, New York (2013). [https://doi.org/10.1007/978-1-4419-7997-1\\_52](https://doi.org/10.1007/978-1-4419-7997-1_52)

7. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Undergraduate Texts in Computer Science. Springer, Heidelberg (2013). <https://doi.org/10.1007/978-1-4471-5559-1>
8. Finden, C.R., Gordon, A.D.: Obtaining common pruned trees. *J. Classif.* **2**(1), 255–276 (1985)
9. Zheng, F., Chen, X., Vacic, V., Nan, P., Zhong, Y., Jiang, T.: MSOAR: a high-throughput ortholog assignment system based on genome rearrangement. *J. Comput. Biol.* **14**(9), 1160–1175 (2007). <https://doi.org/10.1089/cmb.2007.0048>. A preliminary version appeared in RECOMB 2006
10. Jiang, H., Zhu, B., Zhu, D., Zhu, H.: Minimum common string partition revisited. *J. Comb. Optim.* **23**(4), 519–527 (2012). <https://doi.org/10.1007/s10878-010-9370-2>
11. Kirkpatrick, B., Reshef, Y., Finucane, H., Jiang, H., Zhu, B., Karp, R.M.: Comparing pedigree graphs. *J. Comput. Biol.* **19**(9), 998–1014 (2012). <https://doi.org/10.1089/cmb.2011.0254>
12. Tai, K.-C.: The tree-to-tree correction problem. *J. ACM* **26**(3), 422–433 (1979). <https://doi.org/10.1145/322139.322143>
13. Tong, W., Lin, G.: An improved approximation algorithm for the minimum common integer partition problem. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 353–364. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-13075-0\\_28](https://doi.org/10.1007/978-3-319-13075-0_28)
14. Woodruff, D.P.: Better approximations for the minimum common integer partition problem. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX/RANDOM -2006. LNCS, vol. 4110, pp. 248–259. Springer, Heidelberg (2006). [https://doi.org/10.1007/11830924\\_24](https://doi.org/10.1007/11830924_24)



# Guarding Polyhedral Terrain by $k$ -Watchtowers

Nitesh Tripathi<sup>1</sup>, Manjish Pal<sup>2</sup>, Minati De<sup>1</sup>(✉) ,  
Gautam Das<sup>3</sup>, and Subhas C. Nandy<sup>4</sup>

<sup>1</sup> Indian Institute of Science, Bangalore, India  
{nitesht,minati}@iisc.ac.in

<sup>2</sup> National Institute of Technology, Meghalaya, Shillong, India

<sup>3</sup> Indian Institute of Technology, Guwahati, Guwahati, India

<sup>4</sup> Indian Statistical Institute, Kolkata, Kolkata, India

**Abstract.** The discrete  $k$ -watchtower problem for a polyhedral terrain  $\mathcal{T}$  with  $n$  vertices is to find  $k$  vertical segments, called watchtowers, of smallest common height, whose bottom end-points (bases) lie on some vertices of  $\mathcal{T}$ , and every point of  $\mathcal{T}$  is visible from the top end-point of at least one of those vertical segments. Agarwal et al. [1] proposed a polynomial time algorithm using parametric search technique for this problem with  $k = 2$ . Surprisingly, no result is known for the problem when  $k > 2$ . In this paper, we propose an easy to implement algorithm to solve  $k$ -watchtower problem in  $\mathbb{R}^3$  for a fixed constant  $k$ . Our algorithm does not use parametric search.

**Keywords:** Watchtower problem · Polyhedral terrains · Visibility

## 1 Introduction

A polyhedral terrain in  $\mathbb{R}^3$  is a connected 3D polyhedral surface such that for each point  $v = (x, y, z)$  on the surface,  $z = g(x, y)$  for some linear function  $g$  [9]. In other words, any vertical line intersects a terrain at most once and the orthogonal projection of a terrain on the  $XY$ -plane is a (bounded) planar subdivision. In general, a polyhedral terrain in  $\mathbb{R}^d$  is the graph of a continuous, piecewise-linear  $(d - 1)$ - variate function [1].

The problem of placing watchtowers on a polyhedral terrain is a matter of great interest due to its application in surveillance, navigation, computer vision, modelling and graphics, geographic information system, etc. Here the objective is to place a given ( $k$ ) number of watchtowers on the vertices of a polyhedral terrain such that every point in the surface of the terrain is visible from at least one of the watchtowers and the maximum height among these watchtowers is minimized. This is also known as *discrete  $k$ -watchtower problem* where base of

---

M. De—Supported by DST-INSPIRE Faculty Grant (DST-IFA14-ENG-75).

each watchtower is restricted to the vertices of the terrain. In the *continuous* version, base of watchtower can be placed anywhere in the terrain.

Cole and Sharir [3] showed that the problem of finding minimum number of guards, where guards are to be placed on the terrain without any elevation, to guard the terrain is NP-hard. Sharir [7] proposed polynomial time algorithm for the continuous one watchtower placement problem for polyhedral terrain in  $\mathbb{R}^3$  that runs in  $O(n \log^2 n)$  time. Later the time complexity of the problem was improved to  $O(n \log n)$  by Zhu [10]. Agarwal et al. [1] proposed a deterministic polynomial time algorithm for the discrete two watchtower problem for polyhedral terrain in  $\mathbb{R}^3$ . The time complexity of their algorithm is  $O(n^{11/3} \text{polylog}(n))$ , and it uses the parametric search technique of Meggido [5]; here  $n$  is the number of vertices of the polyhedral terrain. Agarwal et al. [1] mentioned the  $k$ -watchtower problem for  $k > 2$  as open. For the continuous version of the problem in  $\mathbb{R}^3$ , no result is known for  $k > 1$ .

In this paper, we propose a general polynomial time algorithm for discrete  $k$ -watchtower problem for any fixed integer  $k > 2$ . In Sect. 3, we first develop an algorithm to decide whether it is feasible to guard the entire terrain using  $k$ -watchtowers for a given height  $h$ , that runs in  $O(n^{k+3} k^2 \alpha^2(n) \log n)$  time. We use this decision procedure to do a binary search over the height to find the optimum height. As the domain of height is continuous, we need to discretise it. In Sect. 4, we describe the details of this discretisation. The running time of our algorithm is  $O(n^{k+3} k^2 \alpha^2(n) \log^2 n + n^7 \alpha^3(n) \log n)$ . The strength of our algorithm is that it is simpler to implement as we do not use parametric search.

## 2 Preliminaries

We use  $[k]$  as a shorthand notation of  $\{1, 2, \dots, k\}$ . Without loss of generality, we assume that each facet of the polyhedral terrain is a triangle. In other words, a terrain in  $\mathbb{R}^3$  is denoted by  $\mathcal{T}(V, E, F)$ , where  $V$ ,  $E$  and  $F$  denote the set of vertices, edges and facets of  $\mathcal{T}$ , respectively. The boundary of a facet  $f \in F$  is denoted by  $\delta f$ .

A watchtower is a vertical line segment whose bottom end point/base lies on a vertex of  $\mathcal{T}$ . We use  $u(h)$  to denote a watchtower based at a vertex  $u \in \mathcal{T}$  and at a height  $h$  from  $u$ . A point  $p \in \mathcal{T}$  is said to be *visible* from the watchtower  $u(h)$  if the line segment  $\overline{p, u(h)}$  lies fully above the terrain.

We say that  $k$  watchtowers  $u_1(h), u_2(h), \dots, u_k(h)$  *guard* the entire terrain  $\mathcal{T}$  if each point on the terrain  $\mathcal{T}$  is visible from at least one of those  $k$  watchtowers; we refer to  $(u_1, u_2, \dots, u_k)$  as a *guard  $k$ -tuple* at height  $h$ .

The *invisibility region*  $H_{u(h)}(f)$  consists of all the points on the facet  $f$  that are not visible from  $u(h)$ . The region  $f \setminus H_{u(h)}(f)$  is referred to as *visibility region* of  $u(h)$  in the facet  $f$ . Note that if  $H_{u(h)}(f) \neq \emptyset$ , then  $H_{u(h)}(f)$  is a collection of *invisibility polygons*. We denote each vertex and edge of an invisibility polygon as *breakpoint* and *invisibility segment*, respectively. Similarly,  $H_{u(h)}(e)$  consists of all the points on the edge  $e \in E$  that are not visible from  $u(h)$ . Note that if  $H_{u(h)}(e) \neq \emptyset$ , then  $H_{u(h)}(e)$  is a collection of *invisibility intervals* on the edge  $e$ . We use  $H_{u(h)}$  to denote all the points of  $\mathcal{T}$  that are not visible from  $u(h)$ .

**Computing  $H_{u(h)}(f)$  :** Observe that boundary of  $H_{u(h)}(f)$  is formed by shadows of some edges of  $\mathcal{T}$  on the facet  $f$  assuming a light source at  $u(h)$ . As noted by Agarwal et al. [1, Sect. 5],  $H_{u(h)}(f)$  is the upper envelope of  $O(n)$  line segments, where each line segment is generated due to intersection of facet  $f$  with another plane defined by  $u(h)$  and an edge in  $\mathcal{E}$ . Here  $\mathcal{E} = E \cup \{\text{line segment } \overline{v, v'} \mid v \in V\}$ , where  $v'$  is a projection of  $v$  on the  $XY$  plane (assuming that all the points of  $V$  are above the  $XY$  plane). Thus, the combinatorial complexity of  $H_{u(h)}(f)$  is  $O(n\alpha(n))$ , where  $\alpha$  is inverse Ackerman function [8]. Due to the result of Hershberger [4], we can compute the region  $H_{u(h)}(f)$  in  $O(n \log n)$  time.

It is easy to see that for an edge  $e$  that is shared by two facets  $f_1$  and  $f_2$ , we can obtain the sorted list of invisibility intervals of  $H_{u(h)}(e)$  after computing both  $H_{u(h)}(f_1)$  and  $H_{u(h)}(f_2)$ , and spending  $O(n \log n)$  time.

However, Agarwal et al. [1, Sect. 5] mentioned that considering all the facets, the combinatorial complexity of the region  $H_{u(h)}$  is  $O(n^2)$ , and we can also compute the region  $H_{u(h)}$  in nearly quadratic time.

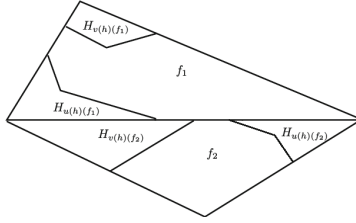
### 3 The Decision Procedure

The decision version of the problem is as follows: *Given a polyhedral terrain  $\mathcal{T}$ , height  $h \in \mathbb{R}$  and a constant  $k \in \mathbb{N}$ , decide whether there exists a guard  $k$ -tuple at height  $h$  for the terrain  $\mathcal{T}$ .*

**Lemma 1.** *A  $k$ -tuple  $(u_1, u_2, \dots, u_k) \in V^k$  is a guard  $k$ -tuple at height  $h$  for the terrain  $\mathcal{T}$  if and only if the following two properties are satisfied:*

- (i) Each edge is collectively visible from  $k$  watchtowers  $u_1(h), u_2(h), \dots, u_k(h)$  i.e. there exist no edge  $e \in E$  such that there is a common point in the intersection of  $k$  invisibility intervals on  $e$  due to these  $k$  watchtowers. Formally,  $\bigcap_{i=1}^k H_{u_i(h)}(e) = \emptyset, \forall e \in E$ .
- (ii) Each facet is collectively visible from  $k$  watchtowers  $u_1(h), u_2(h), \dots, u_k(h)$  i.e. there does not exist a facet  $f$  such that there is a common point in the intersection of  $k$  invisibility polygons on  $f$  due to these  $k$  watchtowers. Formally,  $\bigcap_{i=1}^k H_{u_i(h)}(f) = \emptyset, \forall f \in F$ .

*Proof.* The necessity of the above statement is trivially true. Let's consider sufficiency condition i.e. if above two properties are satisfied then the entire terrain  $\mathcal{T}(V, E, F)$  is visible from the given  $k$  watchtowers. The property (ii) ensures that each facet  $f \in F$  is visible from at least one of the  $k$  watchtowers. However, it is not sufficient to guard the entire terrain as there may exist some part of edge or some vertices which are not visible by any of the watchtowers. For example, in Fig. 1 for two watchtowers; the left portion the shared edge between two facets is not visible by either of the two watchtowers, though both the facets are visible. This situation can be generalized for  $k$  watchtowers as well. The property (i) ensures that each edge and vertex of the terrain is visible. Therefore, satisfying these two properties imply that all the elements in set  $V, E$  and  $F$  of  $\mathcal{T}$  are visible from the said  $k$  watchtowers.  $\square$



**Fig. 1.** Facets  $f_1$  and  $f_2$  are visible but part of the shared edge  $e$  is not visible from both  $u(h)$  and  $v(h)$

The decision procedure proceeds in two steps. In the first step, it discards all  $k$ -tuples  $(u_1, u_2, \dots, u_k) \in V^k$  that do not satisfy the property (i) of the Lemma 1. Considering the remaining  $k$ -tuples, in the second step, it rejects each of those  $k$ -tuples that violates property (ii) of the Lemma.

**First Step:** Given an edge  $e$  and  $k$ -watchtowers of height  $h$  at  $u_1, u_2, \dots, u_k \in V$ , we can compute  $H_{u_i(h)}(e)_{end} \forall i \in [k]$  as mentioned in Sect. 2 in  $O(nk \log n)$  time. Note that in each list, the end-points are already in sorted order. To check the existence of a point  $p \in e$  such that  $p \in \cap_{i=1}^k H_{u_i(h)}(e)$ , we need to merge these sorted  $k$  lists. Using a min-heap of size  $k$ , we can do this in  $O(nk \log k)$  time. If such a point  $p \in e$  exists, then  $e$  is not completely visible by the  $k$ -watchtowers  $\{u_1(h), u_2(h), \dots, u_k(h)\}$ . For a given  $k$ -tuple, we check this observation for all edges  $e \in E$  in  $O(n^2 k \log n)$  time, and reject the tuple if we find an edge which is not guarded. Since we need to repeat this process for all possible  $k$ -tuples in  $V$ , the overall time complexity of this step is  $O(n^{k+2} k \log n)$ .

**Second Step:** For a given facet  $f$  and  $k$  watchtowers of height  $h$  at  $u_1, u_2, \dots, u_k \in V$ , first we compute  $H_{u_i(h)}(f)$  for all  $i \in [k]$ . Now, we need to test whether  $\cap_{i=1}^k H_{u_i(h)}(f) = \emptyset$  or not.

Let  $S_i(f)$  be the set of all edges of  $H_{u_i}(f)$ , and  $S(f) = \cup_{i=1}^k S_i(f)$ . As number of line segments in  $S_i(f)$  is  $O(n\alpha(n))$ , we have  $|S(f)| = O(nk\alpha(n))$ , where  $|A|$  denotes the number of elements in set  $A$ . Let  $I_{S(f)}$  be the set of points generated by pairwise intersection of the line segments in  $S(f)$ . Here,  $|I_{S(f)}| = O(n^2 k^2 \alpha^2(n))$ . The necessary and sufficient condition that  $k$  watchtowers can guard a given facet  $f$  is as follows.

**Lemma 2.** *A facet  $f \in F$  is guarded by  $u_1(h), \dots, u_k(h)$  if and only if there does not exist any point  $p \in I_{S(f)}$  satisfying the following:*

- $p$  is inside the invisibility regions (boundary included)  $H_{u_i(h)}(f) \forall i \in [k]$ .

*Proof.* The necessity of above statement is trivially true. So, let us consider the sufficiency condition i.e. if the above condition is true then the facet  $f$  is completely visible from  $k$  watchtowers. For a contradiction, let us assume that there exists a point  $p'$  at the facet  $f$  which is not visible by any of the  $k$  watchtowers.

This means that this point  $p'$  is inside or on the boundary of common intersection region  $R$  of  $H_{u_i(h)}(f)$ ,  $i \in [k]$ . The extreme points of this closed region  $R$  are in the set  $I_{S(f)}$  and the proof follows.  $\square$

For each  $i \in [k]$ , we separately create a data structure  $D_i$  by triangulating the interior of the polygons in  $H_{u_i(h)}(f)$  and also the exterior of  $H_{u_i(h)}(f)$ . In  $D_i$ , each interior and exterior triangles are given weight 1 and 0, respectively. Since the combinatorial complexity of each  $H_{u_i(h)}(f)$  is  $O(n\alpha(n))$ , the size of each of these data structures is  $O(n\alpha(n))$ , and can be created in  $O(n\alpha(n) \log n)$  time, and point location can be performed in  $O(\log n)$  time [2].

Now, for each point  $p \in I_{S(f)}$ , we search  $p$  in all the data structures  $D_i$ ,  $i \in [k]$ . In each  $D_i$ , we identify the triangle in which  $p$  lies, and add its weight in a counter  $\chi(p)$ . If  $\chi(p) = k$ ,  $p$  is not guarded. Thus, for each point  $p \in I_{S(f)}$ , we spend  $O(k \log n)$  time to decide whether  $q$  is inside common invisibility region  $\cap_i^k H_{u_i(h)}(f)$ . We need to do this test for all the points in  $\cup_{f \in F} I_{S(f)}$ . Since  $|I_{S(f)}| = O(n^2 k^2 \alpha^2(n))$ , the total number of intersection points in  $\cup_{f \in F} I_{S(f)}$  is  $O(n^3 k^2 \alpha^2(n))$ .

Thus, Step 2 takes total  $O(n^{k+3} k^2 \alpha^2(n) \log n)$  time, and we have the following.

**Lemma 3.** *Given a height  $h \in \mathbb{R}$  and a polyhedral terrain  $\mathcal{T}$ , we can decide whether there exists  $k$ -watchtowers of height  $h$  that can guard the terrain  $\mathcal{T}$  in  $O(n^{k+3} k^2 \alpha^2(n) \log n)$  time. We can also report all possible sets of  $k$  watchtowers of height  $h$  that can guard the terrain within same amount of time.*

## 4 Algorithm via Discretisation of Height

Given a polyhedral terrain  $\mathcal{T}$ , in this section, we discuss the algorithm for finding the minimum height  $h^*$  for which we can find  $k$ -watchtowers of height  $h^*$  that can guard  $\mathcal{T}$ .

*Overview of the Algorithm.* Our algorithm consists of two phases. In each phase, we enumerate a set of discrete heights, and perform binary search on them using the decision procedure described in the previous section.

In the first phase, we consider each vertex  $u \in V$  as a possible location of a watchtower. Starting from  $h = 0$ , if we increase the height of the watchtower continuously until  $u(h)$  sees the whole terrain, the invisibility region  $H_{u(h)}(f)$  on each facet  $f \in F$  shrinks continuously. During this process, the shadow of an edge appears or disappears at some specific heights, called *critical heights*. Considering each vertex as a possible location of a watchtower, we enumerate all the critical heights. The number of such critical heights is  $O(n^4)$ , where  $n$  is the number of vertices of the watchtower. We perform binary search among the sorted list  $L_{critical}$  of these critical heights. For each choice of critical height, we test whether this height is feasible for guarding  $\mathcal{T}$  by any tuple of  $k$  watchtowers (see Sect. 3). Thus, we get a pair of consecutive heights  $h', h'' \in L_{critical}$  such that  $h'$  is not feasible but  $h''$  is feasible, and the optimum height  $h^* \in (h', h'']$ .

Note that we have at least one  $k$ -tuple  $(u_i, u_2, \dots, u_k) \in V^k$  such that each point on  $\mathcal{T}$  is visible by at least one of the watchtowers  $\{u_i(h'') \mid i \in [k]\}$ , and there is no common intersection region of the invisible regions  $H_{u_i(h'')}, i \in [k]$ . We may further decrease the height from  $h''$  as long as there is no point  $p \in \mathcal{T}$  that is invisible from all of the watchtowers. This event will occur when all of the invisible regions  $H_{u_i(h)}$  has a common intersection point. In this situation, we can not decrease the height of watchtowers corresponding to the  $k$ -tuple  $(u_i, u_2, \dots, u_k)$  to guard the whole terrain. In the second phase, we enumerate all possible (contact) heights where this type of events might occur considering all possible positions of watchtowers. As in the first phase, we sort this set of contact heights to obtain a sorted list  $L_{contact}$  of (contact) heights. Finally, to obtain the optimum  $h^*$ , we perform a binary search using the decision procedure discussed in Sect. 3.

#### 4.1 Phase 1: Critical-Height Generation

Recall that a typical invisibility region  $H_{u(h)}(f)$  is a collection of simple invisibility polygons. For each invisibility polygon, at least one invisibility segment is a portion of the boundary of  $f$ . Now, let us consider the breakpoints of  $H_{u(h)}(f)$  that are not the end-points of  $\delta f$ . These breakpoints can, in general, be categorized as follows:

- Category-1:** This type of breakpoint is generated by the projection of each vertex of  $\mathcal{T}$  on the facet  $f$ .
- Category-2:** This type of breakpoint is generated by the intersection of the projections of each pair of edges in  $E \setminus \{\text{edges of } \delta f\}$  of  $\mathcal{T}$  on the facet  $f$ .
- Category-3:** This type of breakpoint is generated by the intersection of the projection of edge  $e \in E \setminus \{\text{edges of } \delta f\}$  of  $\mathcal{T}$  on the facet  $f$  and an edge of the facet  $f$ .

Thus, each breakpoint of  $H_{u(h)}(f)$  is associated with either a vertex  $v \in \mathcal{T}$ , or two edges from  $E \setminus \delta f$ , or one edge from  $\delta f$  and another from  $E \setminus \delta f$ .

Initially, let's consider that height  $h = 0$ , and a watchtower is based at  $u \in \mathcal{T}$ . Now, we increase the height continuously until  $u(h)$  sees the entire terrain. During this process, we observe some distinct heights where the topology (set of breakpoints) of  $H_{u(h)}(f)$  changes. We refer to such a height as a *critical height*. We can classify the events corresponding to these as follows.

- Type-A Event:** When a breakpoint  $p$  of Category-1 appears/disappears from the facet  $f$  at some height  $h$ , we call this event as Type-A event.
- Type-B Event:** When two breakpoint  $p, q$  of Category-2 merge to form a single breakpoint of  $H_{u(h)}(f)$ ; we call such an event as Type-B Event. This happens when two breakpoints that are consecutive on the boundary of invisibility region (but not on  $\delta f$ ) merge to form a single vertex.
- Type-C Event:** When two breakpoint  $p, q$ , such that at least one of them is Category-3, merge to form a single breakpoint of  $H_{u(h)}(f)$ , we call such an event as Type-C Event.



It is easy to see that no other event can change the topology of  $H_{u(h)}(f)$ . Considering all the facets  $f \in F$  and all positions  $u \in V$ , first we analyze the Type-A events; next we consider Type-B and Type-C events together.

**Type-A Events:** Let  $p$  be a breakpoint of Category-1. It is projection of a vertex  $p' \in V$  due to a watchtower based at a vertex  $u \in V$ . Assume that we are continuously increasing the height of the watchtower based at  $u$ , and the breakpoint  $p$  appears at some critical height  $h$ . This happens when the tip of the watchtower  $u(h)$ , the vertex  $p' \in V$  and some edge  $e \in E$  become coplanar (see Fig. 2). Consider the plane defined by  $p'$  and  $e$ . Note that this plane intersects the vertical line passing through  $u$  at a height  $h$ .

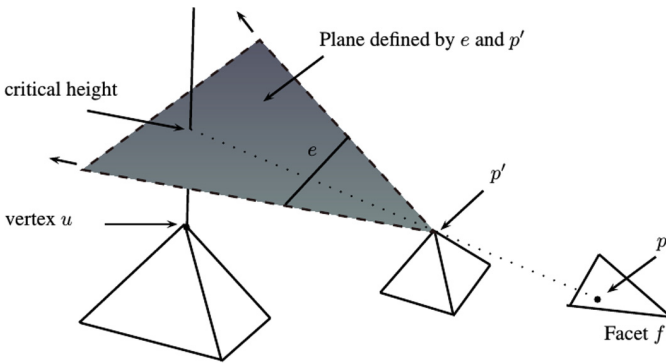


Fig. 2. Computing critical heights for Type-A events

Let  $adj(u)$  be the set of edges in  $E$  that are adjacent to  $u$ , i.e.,  $adj(u) = \{uy|uy \in E, y \in V \setminus \{u\}\}$ . Each vertex-edge pair  $(v, e)$ ,  $v \in V \setminus \{u\}$  &  $e \in E \setminus adj(u)$  defines a plane. Thus, we have  $O(n^2)$  such planes. Intersection of each plane with the vertical line at  $u \in V$  gives a critical height at  $u$  for Type-A event. Considering all the vertices  $u \in V$ , we have  $O(n^3)$  critical heights corresponding to all Type-A event, and all of these can be enumerated in  $O(n^3)$  time.

**Type-B and Type-C Events:** We first consider the Type-B events. Let  $p_1, p_2$  be two Category-2 breakpoints that are consecutive on the boundary of  $H_{u(h)}(f)$ . Let  $e_1, e_2$  be the edges of  $H_{u(h)}(f)$  incident to  $p_1$ , and  $e_2, e_3$  be the edges of  $H_{u(h)}(f)$  incident to  $p_2$ . Let  $e_1, e_2, e_3$  are projections of edges  $e'_1, e'_2, e'_3 \in E$ , respectively. Assume that we are continuously increasing the height of the watchtower based at  $u$ ; at some critical height  $h'$ , we notice that  $p, q$  merge to form a single breakpoint. Clearly, this event happens when the projections of  $e'_1, e'_2$  and  $e'_3$  intersect at a common point on  $f$ . In other words, this happens for a viewpoint from where the edges  $e'_1, e'_2$  and  $e'_3$  appear to intersect at a single point. Plantinga and Dyer [6] showed that this viewing direction is:

$$d = [(p_{11} + t(p_{11} - p_{12}) - p_{21}) \times (p_{21} - p_{22})] \times [(p_{11} + t(p_{11} - p_{12} - p_{21}) \times (p_{21} - p_{22})],$$

where  $p_{i1}, p_{i2}$  are the two end points of the edge  $e'_i$ ,  $i \in [3]$ , and  $0 \leq t \leq 1$ . Note that  $d$  is given as Cartesian coordinates, and it is a quadratic function on  $t$ . Thus, for a given vertex  $v = (x, y, z) \in V$ , we can find the corresponding critical heights due to a triple of edges  $e'_1, e'_2$  and  $e'_3$  in  $O(1)$  time.

Assuming a vertex  $u \in V$  as a possible position of watchtower, we need to consider each triple of edges  $e'_1, e'_2, e'_3 \in E$ , define the corresponding viewing direction  $d$  and compute its intersection with the vertical line at  $u \in V$ . Thus, the number of Type-B events for a vertex  $u \in V$  is  $O(n^3)$ . Considering all the vertices in  $V$ , we have in total  $O(n^4)$  Type-B events.

Note that a Type-C event corresponds to the event when at least one of the edges from  $e_1, e_2$  and  $e_3$  of  $H_{u(h)}(f)$  is a part of the facet  $f$ . Thus, this event corresponds to the viewing direction from where  $e'_1, e'_2$  and  $e'_3$  appear to intersect at a single point, and one of the edges is on the boundary of the facet  $f$ . It is easy to note that during the process of enumerating all critical heights corresponding to the Type-B events, we also enumerated critical heights corresponding to Type-C events.

Let  $L_{critical}$  be the sorted list of the union of all the critical heights generated by Type-A, Type-B and Type-C events considering all the vertices  $u \in V$  together. It is easy to see the following.

**Lemma 4.** *Given a polyhedral terrain  $\mathcal{T}$  with  $n$  vertices, the size of the sorted list  $L_{critical}$  is  $O(n^4)$ , and this can be computed in  $O(n^4 \log n)$  time.*

**Lemma 5.** *If  $h_i, h_{i+1}$  ( $h_{i+1} > h_i$ ) are two consecutive critical heights in  $L_{critical}$ , then the topology of  $H_{u(h)}(f)$  will be same for any  $h \in [h_i, h_{i+1})$ , where  $u \in V$  and  $f \in F$ .*

**Lemma 6.** *It is not possible that the optimum height  $h^*$  is larger than the maximum height in the list  $L_{critical}$ .*

**Lemma 7.** *A pair of consecutive heights  $(h', h'')$  in  $L_{critical}$  such that  $h'$  is not feasible but  $h''$  is feasible, can be computed in  $O(n^{k+3}k^2\alpha^2(n) \log^2 n)$  time.*

*Proof.* We can find  $(h', h'')$  by performing a binary search on the sorted list  $L_{critical}$ . As each decision of the binary search takes  $O(n^{k+3}k^2\alpha^2(n) \log n)$  time (Lemma 3), the lemma follows.  $\square$

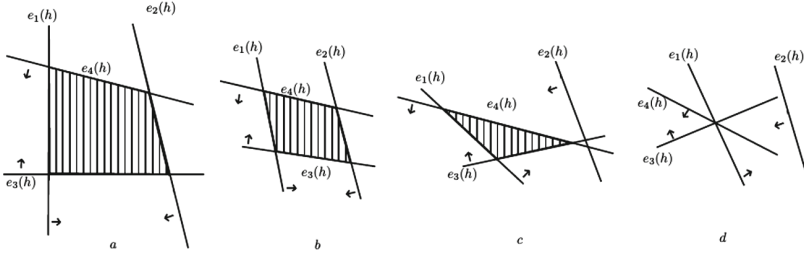
## 4.2 Phase 2: Contact-Height Generation

Let  $(u_1, u_2, \dots, u_k) \in V^k$  be a guard  $k$ -tuple at the optimal height  $h^*$ . From above discussion, we know that  $h^* \in (h', h'']$ , and for a fixed watchtower base  $u_i \in V$  and facet  $f \in F$ , the topology of invisibility region  $H_{u_i(h)}(f)$  remain same for any height  $h \in [h', h'')$  (see Lemma 5). Since  $h^* > h'$ , from Lemma 1, we know that at least one of the following happens:

- there exists a facet  $f \in \mathcal{T}$  such that the common invisibility region  $\cap_i^k H_{u_i(h')}(f)$  is not empty,

- there exists an edge  $e \in E$  that has non-empty common invisibility interval, in other words,  $\cap_{i=1}^k H_{u_i}(h')(e) \neq \emptyset$ .

Note that the common invisibility region  $\cap_{i=1}^k H_{u_i}(h')(f)$  is a collection of polygons. If we increase the height continuously from  $h'$  then the common invisibility region shrinks continuously, and at some specific height it converges into a point (see Fig. 3). When this event occurs, then at least three invisibility segments from  $\cup_{i=1}^k H_{u_i}(h')(f)$  meet at a point.



**Fig. 3.** Shrinking (a–d) of common invisibility region  $\cap_{i=1}^k H_{u_i}(h')(f)$  due to increase in heights (arrow indicates the half-plane that is not visible)

Similarly, the common invisibility interval  $\cap_{i=1}^k H_{u_i}(h')(e)$  is a collection of intervals along the edge  $e$ . If we increase the height continuously from  $h'$  then the common invisibility interval shrinks continuously, and at some specific height it converges into a point. When this event occurs, then at least two invisibility segments from  $\cup_{i=1}^k H_{u_i}(h')(f_1) \cup \cup_{i=1}^k H_{u_i}(h')(f_2)$  meet at a point on the edge  $e \in E$ , where the facets  $f_1$  and  $f_2$  share the edge  $e$ .

In this phase, we enumerate each height  $h \in (h', h'')$  where any of the following events occur:

**Type-D Event:** Any three invisibility segments  $e_1, e_2, e_3$  from  $\cup_{u_i \in V} H_{u_i}(h')(f)$  intersect at a point;  $e_1, e_2, e_3$  must be due to at least two (at most three) watchtowers based at two (three) different vertices. Note that, if  $e_1, e_2, e_3$  are due to a single watchtower then  $e_1, e_2, e_3$  are three edges of an invisibility region  $H_{u_i}(h')(f)$ . This case is already considered as a Type-B event in Phase 1.

**Type-E Event:** Any two invisibility segments  $e_1, e_2$  from  $\cup_{u_i \in V} H_{u_i}(h')(f_1) \cup \cup_{u_i \in V} H_{u_i}(h')(f_2)$  and the edge  $e_3$  intersect at a point, where  $e_3$  is shared by both the facets  $f_1$  and  $f_2$ ;  $e_1, e_2$  must be due to at least two watchtowers based at two different vertices. Note that if  $e_1, e_2$  are due to a single watchtower then this case is already considered as a Type-C event in Phase 1.

We refer each height corresponding to the above two types of events as *contact height*. From the above discussions, it is easy to prove the following.

**Lemma 8.** *If  $h^* \neq h''$ , then  $h^*$  must be a contact height.*

*Proof.* Let  $(u_1, u_2, \dots, u_k) \in V^k$  be a guard  $k$ -tuple at the optimal height  $h^*$ .

If  $h^* \neq h''$ , then the optimum height  $h^*$  depends on the following two cases; one of them is sure to occur.

**Case 1:** for a facet  $f$ , the common invisibility region  $\cap_{i=1}^k H_{u_i(h')}(f)$  converges to a point, and for each facet  $f' \in F \setminus f$  and edge  $e' \in E$ , we have  $\cap_{i=1}^k H_{u_i(h')}(f') = \emptyset$  and  $\cap_{i=1}^k H_{u_i(h')}(e') = \emptyset$ .

**Case 2:** for an edge  $e$ , the common invisibility interval  $\cap_{i=1}^k H_{u_i(h')}(e)$  converges to a point, and for each facet  $f' \in F$  and edge  $e' \in E \setminus e$ , we have  $\cap_{i=1}^k H_{u_i(h')}(f') = \emptyset$  and  $\cap_{i=1}^k H_{u_i(h')}(e') = \emptyset$ .

In Case 1, we know that at least three invisibility segments from  $\cup_{i=1}^k H_{u_i(h')}(f)$  meet at a point. If more than three invisibility segments of  $\cup_{i=1}^k H_{u_i(h')}(f)$  meet at a point, then each 3-tuple of those segments produce the same contact height  $h^*$ . With a similar argument, for Case 2, we can show that  $h^*$  is a contact height.  $\square$

Let  $e$  be an invisibility segment of  $H_{u_i(h')}(f)$  due to the watchtower  $u_i(h')$ . Note that  $e$  must be along the line of intersection between the facet  $f$  and a plane defined by  $u_i(h')$  and an edge  $e' \in \mathcal{E}$ . Recall that  $\mathcal{E} = E \cup \{\text{line segment } \overline{v, v'} \mid v \in V\}$ , where  $v'$  is a projection of  $v$  on the  $XY$  plane (assuming that all the points of  $V$  are above the  $XY$  plane). We use  $e(h)$  to denote the position of the invisibility segment  $e$  at height  $h \in (h', h'']$ , i.e.,  $e(h)$  lies along the intersection of the facet  $f$  and the plane defined by the edge  $e'$  and the point  $u_i(h)$ . Using elementary geometry, we prove the following three lemmata.

**Lemma 9.** *Let  $(a, b, c + h)$  be the Cartesian co-ordinates of the watchtower  $u_i(h)$ . Let  $p_j = (a_j, b_j, c_j)$ ,  $j \in [2]$  be the two end points of the edge  $e' \in \mathcal{E}$ . Let  $l\mathbf{x} + m\mathbf{y} + n\mathbf{z} = D$  be the equation of the plane containing the facet  $f$ . The vector form of equation of the line containing the segment  $e(h)$  is:  $\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle = \langle 0, \frac{f_3(h) - \frac{c}{n}D}{f_2(h) - \frac{c}{n}m}, \frac{f_3(h) - \frac{f_2(h)}{m}D}{C - \frac{f_2(h)}{m}n} \rangle + t\langle nf_2(h) - mC, lC - nf_1(h), mf_1(h) - lf_2(h) \rangle$ , where*

$$\begin{aligned} f_1(h) &= (b - b_1)(c_2 - c_1) - (c + h - c_1)(b_2 - b_1), \\ f_2(h) &= (c + h - c_1)(a_2 - a_1) - (a - a_1)(c_2 - c_1), \\ C &= (a - a_1)(b_2 - b_1) - (b - b_1)(a_2 - a_1) \text{ and } f_3(h) = a_1 f_1(h) + b_1 f_2(h) + c_1 C. \end{aligned}$$

*Proof.* Let  $\mathbf{P}$  be the plane defined by the point  $u_i(h)$  and the edge  $e'$ . We have two vectors  $\overrightarrow{p_1}, u_i(h) = \langle a - a_1, b - b_1, c + h - c_1 \rangle$  and  $\overrightarrow{p_1}, \overrightarrow{p_2} = \langle a_2 - a_1, b_2 - b_1, c_2 - c_1 \rangle$  that lie completely in the plane  $\mathbf{P}$ . So, the vector  $\overrightarrow{n_1} = \overrightarrow{p_1}, u_i(h) \times \overrightarrow{p_1}, \overrightarrow{p_2}$  is orthogonal to the plane  $\mathbf{P}$ . We have  $\overrightarrow{n_1} = \langle f_1(h), f_2(h), C \rangle$ , where

$$\begin{aligned} f_1(h) &= (b - b_1)(c_2 - c_1) - (c + h - c_1)(b_2 - b_1), \\ f_2(h) &= (c + h - c_1)(a_2 - a_1) - (a - a_1)(c_2 - c_1), \text{ and} \\ C &= (a - a_1)(b_2 - b_1) - (b - b_1)(a_2 - a_1). \end{aligned}$$

The equation of the plane  $\mathbf{P}$  is  $f_1(h)(\mathbf{x} - a_1) + f_2(h)(\mathbf{y} - b_1) + C(\mathbf{z} - c_1) = 0$ , i.e.,

$$f_1(h)\mathbf{x} + f_2(h)\mathbf{y} + C\mathbf{z} = f_3(h) \tag{1}$$

Here  $f_3(h) = a_1f_1(h) + b_1f_2(h) + c_1C$  is a linear function of  $h$ .

We know that the equation of the plane  $\mathbf{F}$  containing the facet  $f$  is

$$l\mathbf{x} + m\mathbf{y} + n\mathbf{z} = D \tag{2}$$

This planes have normal vector  $\vec{n}_2 = \langle l, m, n \rangle$ . Let  $L$  denote the line of intersection of the two planes  $\mathbf{P}$  and  $\mathbf{F}$ .

Then, the direction vector of the line  $L$  is:  $\vec{v} = \vec{n}_1 \times \vec{n}_2 = \langle f_4(h), f_5(h), f_6(h) \rangle$ , where  $f_4(h) = nf_2(h) - mC$ ,  $f_5(h) = lC - nf_1(h)$  and  $f_6(h) = mf_1(h) - lf_2(h)$ .

Now, we need to find a point  $p$  on the line  $L$  to construct the equation of the line. We consider  $p_0$  to be a point on the line  $L$  with  $x = 0$ . Thus, substituting  $x = 0$  in the Eqs. 1 and 2 of the planes, we get  $p_0 = (0, \frac{f_3(h) - \frac{C}{n}D}{f_2(h) - \frac{C}{n}m}, \frac{f_3(h) - \frac{f_2(h)}{m}D}{C - \frac{f_2(h)}{m}n})$ .

Thus the vector form of the line  $L$  is  $\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle = \vec{p}_0 + t\vec{v}$

$$= \langle 0, \frac{f_3(h) - \frac{C}{n}D}{f_2(h) - \frac{C}{n}m}, \frac{f_3(h) - \frac{f_2(h)}{m}D}{C - \frac{f_2(h)}{m}n} \rangle + t\langle f_4(h), f_5(h), f_6(h) \rangle$$

$$= \langle 0, \frac{f_3(h) - \frac{C}{n}D}{f_2(h) - \frac{C}{n}m}, \frac{f_3(h) - \frac{f_2(h)}{m}D}{C - \frac{f_2(h)}{m}n} \rangle + t\langle nf_2(h) - mC, lC - nf_1(h), mf_1(h) - lf_2(h) \rangle.$$

□

**Lemma 10.** *Let  $e_i$  be an invisibility segment of  $H_{u_i(h')}(f)$  due to the watchtower  $u_i(h')$ , for each  $i \in [3]$ . The number of contact heights  $h \in (h', h'')$  generated due to the meeting of the invisibility segments  $e_1(h), e_2(h)$  and  $e_3(h)$  at a single point can be at most three, and all of them can be enumerated in  $O(1)$  time.*

*Proof.* According to Lemma 9, the vector form of equation of the line  $L_i$  on the plane  $F$  containing  $e_i(h)$  is as follows:

$$\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle = \langle 0, \frac{f_{i,3}(h) - \frac{C_i}{n}D}{f_{i,2}(h) - \frac{C_i}{n}m}, \frac{f_{i,3}(h) - \frac{f_{i,2}(h)}{m}D}{C_i - \frac{f_{i,2}(h)}{m}n} \rangle + t\langle nf_{i,2}(h) - mC_i, lC_i - nf_{i,1}(h), mf_{i,1}(h) - lf_{i,2}(h) \rangle,$$

where each  $f_{i,j}(h), i, j \in [3]$  is a linear function of  $h$ .

Projecting  $L_i$  on the  $XY$  plane, we get the line  $L'_i$  as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle 0, \frac{f_{i,3}(h) - \frac{C_i}{n}D}{f_{i,2}(h) - \frac{C_i}{n}m} \rangle + t\langle nf_{i,2}(h) - mC_i, lC_i - nf_{i,1}(h) \rangle.$$

Converting this vector form of equation in slope-intercept form, we get

$$\mathbf{y} = \frac{lC_i - nf_{i,1}(h)}{nf_{i,2}(h) - mC_i} \mathbf{x} + \frac{f_{i,3}(h) - \frac{C_i}{n}D}{f_{i,2}(h) - \frac{C_i}{n}m}$$

$$= \frac{lC_i - nf_{i,1}(h)}{nf_{i,2}(h) - mC_i} \mathbf{x} + \frac{nf_{i,3}(h) - C_iD}{nf_{i,2}(h) - mC_i}.$$

For the sake of simplicity, we rewrite this expression of  $L'_i$  as:  $\mathbf{y} = \frac{\phi_{i,1}(h)}{\phi_{i,3}(h)}\mathbf{x} + \frac{\phi_{i,2}(h)}{\phi_{i,3}(h)}$ , where each  $\phi_{i,j}(h)$ ,  $i, j \in [3]$  is a linear function on  $h$ .

Now, to get the contact height  $h$  at which the invisibility segments  $e_1(h)$ ,  $e_2(h)$  and  $e_3(h)$  intersect at a single point, we find the equation of area of the triangle formed by  $L'_1, L'_2$  and  $L'_3$  and find roots of this equation.

We know that the area of triangle formed by three lines  $\mathbf{y} = m_1\mathbf{x} + c_1$ ,  $\mathbf{y} = m_2\mathbf{x} + c_2$  and  $\mathbf{y} = m_3\mathbf{x} + c_3$  is given by

$$\Delta = \frac{\begin{vmatrix} -m_1 & 1 & -c_1 \\ -m_2 & 1 & -c_2 \\ -m_3 & 1 & -c_3 \end{vmatrix}^2}{2.K_1.K_2.K_3}$$

Where  $K_1 = \begin{vmatrix} -m_2 & 1 \\ -m_3 & 1 \end{vmatrix}$ ,  $K_2 = -\begin{vmatrix} -m_1 & 1 \\ -m_3 & 1 \end{vmatrix}$  and  $K_3 = \begin{vmatrix} -m_1 & 1 \\ -m_2 & 1 \end{vmatrix}$  are the cofactors of  $-c_1, -c_2, -c_3$ , respectively, in the above matrix.

Using above formula, the area of triangle formed by  $L'_1, L'_2$  and  $L'_3$  is  $\Delta = \frac{\Phi_1(h)^2}{\Phi_2(h)\Phi_3(h)\Phi_4(h)}$ , where  $\Phi_1(h)$  is a cubic function in  $h$ , and  $\Phi_2(h)$ ,  $\Phi_3(h)$  and  $\Phi_4(h)$  are quadratic function in  $h$ .

We need to solve one cubic equation  $\Phi_1(h) = 0$  to find the heights where the area  $\Delta = 0$ . Thus, we have at most three distinct heights at which  $L_1, L_2$  and  $L_3$  intersect at a common point.  $\square$

**Lemma 11.** *Let  $e_1$  be an invisibility segment of  $H_{u_i(h')}(f_1)$  due to the watchtower  $u_i(h')$ , and  $e_2$  be an invisibility segment of  $H_{u_j(h')}(f_2)$  due to the watchtower  $u_j(h')$ ,  $i \neq j$ . Let  $e_3 \in E$  be the edge that is shared by both the facets  $f_1$  and  $f_2$ . The number of contact heights  $h \in (h', h'')$  generated due to the meeting of the invisibility segments  $e_1(h), e_2(h)$  and the edge  $e_3$  at a common point can be at most one, and it can be enumerated in  $O(1)$  time.*

*Proof.* Let  $l_i\mathbf{x} + m_i\mathbf{y} + n_i\mathbf{z} = D_i$  be the equation of the plane  $F_i$  containing the facet  $f_i$ ,  $i \in [2]$ . According to Lemma 9, the vector form of equation of the line  $L_i$  on the plane  $F_i$  containing  $e_i(h)$  is as follows:

$$\begin{aligned} &\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle \\ &= \left\langle 0, \frac{f_{i,3}(h) - \frac{C_i D}{n_i}}{f_{i,2}(h) - \frac{C_i}{n_i} m_i}, \frac{f_{i,3}(h) - \frac{f_{i,2}(h)}{m_i} D_i}{C_i - \frac{f_{i,2}(h)}{m_i} n_i} \right\rangle + t \langle n_i f_{i,2}(h) - m_i C_i, l_i C_i - n f_{i,1}(h), m_i f_{i,1}(h) - l_i f_{i,2}(h) \rangle, \end{aligned}$$

where each  $f_{i,j}(h)$ ,  $i \in [2], j \in [3]$  is a linear function of  $h$ .

Let the vector form of equation of the line containing the edge  $e_3$  be  $\langle x, y, z \rangle = \langle \alpha, \beta, \lambda \rangle + t \langle p, q, r \rangle$ .

It is easy to see that  $\rho_i = \left( \frac{\alpha f_i(h)}{f_i(h) - p}, \frac{\alpha q + \beta(f_i(h) - p)}{f_i(h) - p}, \frac{\alpha r + \lambda(f_i(h) - p)}{f_i(h) - p} \right)$  is the intersection point of the two lines  $L_i$  and  $e_3$ , where  $f_i(h) = n_i f_{i,2}(h) - m_i C_i$ . The length of the line segment  $\overline{\rho_1, \rho_2}$  is  $\tau(h) = \frac{\alpha \sqrt{q^2 + r^2 + 1} (f_2(h) - f_1(h))}{(f_1(h) - p)(f_2(h) - p)}$ . When  $L_1, L_2$  and  $e_3$  intersect at a common point then the length of  $\tau(h) = 0$ . So, we need

to solve a linear equation:  $f_2(h) - f_1(h) = 0$  on  $h$  to find the critical heights corresponding to this event. Thus the lemma follows.  $\square$

**Lemma 12.** *Total number of contact height is  $O(n^7\alpha^3(n))$ , and the sorted list  $L_{contact}$  of all the contact heights can be enumerated in  $O(n^7\alpha^3(n)\log n)$  time.*

*Proof.* For each triple of invisibility segments from  $\cup_{u_i \in V} H_{u_i(h')}(f)$ , we have at most three distinct contact heights due to Type-D events, and we can enumerate them in  $O(1)$  time (From Lemma 10). Since the total number of invisibility segments in  $\cup_{u_i \in V} H_{u_i(h')}(f)$  is  $O(n^2\alpha(n))$ , the total number of contact heights (due to Type-D events) for the facet  $f$  is  $O(n^6\alpha^3(n))$ . Considering all facets, total number of contact heights due to Type-D events is  $O(n^7\alpha^3(n))$ , and we can enumerate all of them in  $O(n^7\alpha^3(n))$  time. Using similar argument and Lemma 11, it is easy to see that the total number of contact heights due to Type-E events is  $O(n^5\alpha^2(n))$ , and we can enumerate them in  $O(n^5\alpha^2(n))$  time. As we need sorting to obtain the sorted list  $L_{contact}$  containing all the contact heights, the total time complexity is  $O(n^7\alpha^3(n)\log n)$ .  $\square$

**Theorem 13.** *Given polyhedral terrain  $\mathcal{T}$  in  $\mathbb{R}^3$  with  $n$  vertices and a fixed integer  $k$ , we can find a guard  $k$ -tuple for the terrain  $\mathcal{T}$  with minimum height in  $O(n^{k+3}k^2\alpha^2(n)\log^2 n + n^7\alpha^3(n)\log n)$  time.*

## 5 Conclusion

We propose a simple to implement algorithm for  $k$ -watchtower problem, for any fixed integer  $k$ . Note that the time complexity for 3-watchtower problem using our algorithm is  $O(n^7\alpha^3(n)\log n)$  where the dominating term is the number of contact heights enumerated in the phase 2 of the algorithm. A natural direction of future work is to improve the time complexity for  $k = 3$ . On the other hand, for  $k > 3$ , the time needed by the decision procedure is the bottleneck of our algorithm. Whether one can improve the time complexity of the decision procedure further remains open.

**Acknowledgement.** The authors wish to acknowledge anonymous reviewer for useful comments on the previous version of the paper.

## References

1. Agarwal, P.K., Bereg, S., Daescu, O., Kaplan, H., Ntafos, S.C., Sharir, M., Zhu, B.: Guarding a terrain by two watchtowers. *Algorithmica* **58**(2), 352–390 (2010)
2. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: *Computational Geometry: Algorithms and Applications*, 3rd edn. Springer, Santa Clara (2008). <https://doi.org/10.1007/978-3-540-77974-2>
3. Cole, R., Sharir, M.: Visibility problems for polyhedral terrains. *J. Symb. Comput.* **7**(1), 11–30 (1989)
4. Hershberger, J.: Finding the upper envelope of  $n$  line segments in  $O(n \log n)$  time. *Inf. Process. Lett.* **33**(4), 169–174 (1989)

5. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. *J. ACM (JACM)* **30**(4), 852–865 (1983)
6. Plantinga, W.H., Dyer, C.R.: Visibility, occlusion, and the aspect graph. *Int. J. Comput. Vis.* **5**(2), 137–160 (1990)
7. Sharir, M.: The shortest watchtower and related problems for polyhedral terrains. *Inf. Process. Lett.* **29**(5), 265–270 (1988)
8. Sharir, M., Agarwal, P.K.: *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, Cambridge (1995)
9. Zhu, B.: Computational geometry in two and a half dimensions. Ph.D. thesis, School of Computer Science, McGill University (1994)
10. Zhu, B.: Computing the shortest watchtower of a polyhedral terrain in  $O(n \log n)$  time. *Comput. Geom.* **8**(4), 181–193 (1997)





# Some (in)tractable Parameterizations of Coloring and List-Coloring

Pranav Arora<sup>1</sup>, Aritra Banik<sup>1</sup>, Vijay Kumar Paliwal<sup>1(✉)</sup>,  
and Venkatesh Raman<sup>2</sup>

<sup>1</sup> Indian Institute of Technology Jodhpur, Jodhpur, India  
arorapranav53@gmail.com, aritrabanik@gmail.com, getvijaypaliwal@gmail.com

<sup>2</sup> The Institute of Mathematical Sciences, HBNI, Chennai, India  
vraman@imsc.res.in

**Abstract.** Graph Coloring and its generalization list coloring are fundamental graph optimization problems with various applications. Most versions of the problems are hard in several paradigms including approximation and parameterized complexity. We consider a few versions of the problems that are polynomial time solvable, and try to extend the notion of feasible algorithms by parameterizing suitably in the paradigm of parameterized complexity. More specifically,

- It is known that given a planar graph with any list of size 5 for each vertex, there is a proper coloring of the graph such that each vertex gets its color from its list. We show that if the graph is  $k$  vertices away from a planar graph, then deciding whether such a coloring exists is PARA-NP-hard when parameterized by  $k$ , i.e. it is NP-hard for even constant values of  $k$ . It is known that any graph with maximum degree 3 is 3-colorable unless the graph is a 4-clique. We show that if the graph is  $k$  vertices away from a maximum degree 3 graph, then determining whether it is 3-colorable is PARA-NP-hard when parameterized by  $k$ .
- It is known that if each vertex has a list of size 2, then the list coloring which asks whether there is a coloring respecting the lists is polynomial time solvable. We show that if only  $k$  vertices have lists of size more than 2, then the problem becomes W[1]-hard.
- It is known that determining whether a graph on  $n$  vertices is  $n - k$  colorable, is fixed-parameter tractable on  $k$ . We consider the list coloring variation of it where each vertex has a list of size  $n - k$  and we ask whether the graph has a coloring respecting the lists of colors. We show that the problem has an  $XP$  algorithm, i.e. an algorithm with runtime  $n^{O(k)}$ . At least this shows that the problem cannot be PARA-NP-hard unless  $P = NP$ . We leave open the question whether the problem is fixed-parameter tractable.
- Finally, it is known that 2-LIST COLORING is polynomial time solvable. If there is no such coloring, then we address the following natural question: are there  $k$  vertices or edges whose removal results in a feasible coloring. We show that these versions are fixed-parameter tractable when parameterized by  $k$ . These generalize the odd cycle transversal problem and edge-bipartization problem which are well-studied problems particularly in parameterized complexity.

# 1 Introduction

## 1.1 Problems and Motivation

The graph coloring problem is one of the fundamental combinatorial optimization problems with applications in scheduling, register allocation, pattern matching and many other active research areas. Given a graph  $G = (V, E)$ , the graph coloring problem is a way to assign colors/labels to vertices of a graph such that no two adjacent vertices share the same color. Such a coloring is also known as a *proper* coloring. The smallest number of colors needed to color a graph  $G$  is called its chromatic number, and is denoted by  $\chi(G)$ . Determining whether a graph is 3-colorable is NP-hard [7] while the 2-coloring problem has a linear time algorithm. It is even hard to approximate the chromatic number in polynomial time. The 3-coloring problem remains NP-complete even on 4-regular planar graphs [6]. There are some generalizations and variations of ordinary graph colorings which are motivated by practical applications. For example, sometimes a set of feasible colors  $L(v)$  is attached with each vertex  $v \in V$  and we may require that the vertex  $v$  be colored from a color from  $L(v)$ . This takes us to the *list coloring* problem, and clearly this is a generalization of standard graph coloring, and hence is considerably harder.

### LIST COLORING PROBLEM

**Input:** A graph  $G = (V, E)$  and a LIST of  $|V|$  many set of colors,  $L(v)$  for each  $v \in V$

**Question:** Find an assignment of colors  $c : V \rightarrow \cup_{v \in V} L(v)$  such that for any vertex  $v$ ,  $c(v) \in L(v)$  and for any two adjacent vertices  $u$  and  $v$ ,  $c(v) \neq c(u)$

A LIST is  $\ell$ -REGULAR if each set contains exactly  $\ell$  colors.  $\ell$ -REGULAR LIST COLORING problem is to decide whether  $G = (V, E)$  has a coloring that respects  $L$ , where  $L$  is  $\ell$ -REGULAR. A graph  $G$  is  $\ell$ -CHOOSABLE if for every  $\ell$ -regular LIST  $L$  of  $G$ , there exists a coloring which respects  $L$ .

### CHOOSABILITY PROBLEM

**Input:** A graph  $G = (V, E)$  and an integer  $\ell$

**Question:** Determine whether  $G$  is  $\ell$ -CHOOSABLE

Cai [2] in one of the earliest papers on parameterizations of graph coloring studied various parameterizations. He showed surprisingly that it is NP-hard to determine whether a graph that is two vertices away from a bipartite graph is 3-colorable. A graph is  $k$  vertices away from a graph satisfying a property (say planar or bipartite), if there are  $k$  vertices in the graph whose removal results in a graph satisfying the property. We say that such a graph is planar  $+kv$  if it has  $k$  vertices whose deletion results in a planar graph. We consider such parameterizations in this paper for graph coloring and list coloring and give hardness and FPT (fixed-parameter tractable) results.

We begin with the notions of parameterized complexity before we explain our results.

## 1.2 Parameterized Complexity

The goal of Parameterized Complexity is to find ways of solving NP-hard problems more efficiently than brute force: here the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. A *parameterization* of a problem is assigning a positive integer parameter  $k$  to each input instance. Formally we say that a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm  $\mathcal{A}$  (called a *fixed parameter algorithm*), a computable function  $f$ , and a constant  $c$  such that, given problem instance  $(x, k)$ ,  $\mathcal{A}$  correctly solves the problem in time bounded by  $f(k) \cdot |x, k|^c$ , where  $x$  is the input and  $k$  is the parameter [4]. There is also an accompanying theory of parameterized intractability using which one can identify parameterized problems that are unlikely to admit FPT algorithms. These are essentially proved by showing that the problem is W-hard.

A parameterized problem is called *slice-wise polynomial* (XP) if there exists an algorithm  $\mathcal{A}$ , two computable functions  $f, g$  such that, given problem instance  $(x, k)$ , the algorithm  $\mathcal{A}$  correctly solves the problem in time bounded by  $f(k) \cdot |x, k|^{g(k)}$ , where  $x$  is the input and  $k$  is the parameter [4]. The complexity class containing all slice-wise polynomial problems is called XP. We say that a parameterized problem is *para-NP-hard* if the problem is NP-hard for some fixed constant value of the parameter. *para-NP-hard* problems are not in XP unless  $P = NP$ .

## 1.3 Our Results

We start with parameterization of simple polynomial time solvable cases of COLORING and LIST COLORING. Thomassen [15] showed that every planar graph is 5-choosable. Therefore, by definition of choosability, planar graph is always a *Yes* instance for 5-REGULAR LIST COLORING. Our first result is that 5-REGULAR LIST COLORING is *para-NP-hard* for planar  $+kv$  graphs.

Garey et al. [7] showed that COLORING is polynomial time solvable for graphs with maximum degree 3, actually all such graphs other than 4-clique are 3-colorable. Now an interesting question is, if we are given a graph which has a set of  $k$  vertices whose deletion makes the graph maximum degree 3, how hard is COLORING on this graph. First we define the problem formally. We denote graph of maximum degree  $d$  by  $G_d$ .

$G_3 + kv$ COLORING	<b>Parameter:</b> $k$
<b>Input:</b> A graph $G = (V, E)$ with a set $S \subseteq V$ , $ S  = k$ where $G \setminus S$ is maximum degree 3 graph.	
<b>Question:</b> Determine whether $G$ is 3-colorable.	

We show this also to be *para-NP-hard*.

In case of general graphs no results are known for LIST COLORING. Observe that for general graph  $k$ -REGULAR LIST COLORING is *para-NP-hard* as the problem is a generalization of  $k$ -coloring. It is known that if we ask whether the graph can be properly colored with  $(n - k)$  colors, then the problem is FPT [4]. We consider a similar variation on LIST COLORING.

$(n - k)$ -REGULAR LIST COLORING

**Parameter:**  $k$

**Input:** A graph  $G = (V, E)$  and a  $(n - k)$ -REGULAR LIST  $L$ , where  $n = |V|$

**Question:** Determine whether there exists a feasible coloring of  $G$  with respect to  $L$

We show that  $(n - k)$ -REGULAR LIST COLORING is in XP which is the most technical part of the paper. It remains as an open problem whether the problem is fixed-parameter tractable.

It is known [5] that 2-REGULAR LIST COLORING is polynomial time solvable for general graphs. As we cannot see a formal proof, we give it here for completeness. We have the following theorem.

**Theorem 1.** 2-REGULAR LIST COLORING is polynomial time solvable for general graphs.

*Proof.* Let  $G = (V, E)$  be a graph and  $L$  be a 2-regular list assignment for  $G$ . This problem can be reduced to 2-CNF SAT which is polynomial time solvable [1]. For each  $u \in V$ , define a variable  $x_u$ .  $x_u$  is set to true if the first color from its list is assigned to vertex  $u$ , otherwise  $x_u$  is set to false. For each edge  $(u, v) \in E$ , add clauses depending on the common colors in  $L(u)$  and  $L(v)$ . No clauses need to be added if there are no common colors. There are four cases:

1. First color of  $L(u)$  and first color of  $L(v)$  is same. Add clause  $(\overline{x_u} \vee \overline{x_v})$ .
2. First color of  $L(u)$  and second color of  $L(v)$  is same. Add clause  $(\overline{x_u} \vee x_v)$ .
3. Second color of  $L(u)$  and first color of  $L(v)$  is same. Add clause  $(x_u \vee \overline{x_v})$ .
4. Second color of  $L(u)$  and second color of  $L(v)$  is same. Add clause  $(x_u \vee x_v)$ .

These clauses ensure that adjacent vertices do not get the same color because if any two adjacent vertices get the same color then at least one of the clauses will be falsified. Suppose that there exists an assignment that satisfies the 2-CNF SAT formula. If  $x_u$  is true, then  $u$  is assigned the first color in its list, otherwise it is assigned the second color. Conversely, if there exists a coloring of  $G$  with the given  $L$ , then  $x_u$  is set to true if it is assigned the first color from its list, otherwise it is set to false. Now, since no two adjacent vertices are assigned same color, none of the clauses will be falsified.  $\square$

Two natural questions are: (1) if a graph  $G = (V, E)$  is not colorable with respect to a 2-regular list  $L$  then is it possible to color at least  $k$  vertices of the graph respecting  $L$ ? This problem is known to be W[1]-hard [9]. In this paper we address the second natural question (which is sometimes called ‘parameteric dual’ defined below).

Vertex Parameterized 2-REGULAR LIST COLORING

**Parameter:**  $k$

**Input:** A graph  $G = (V, E)$  with a 2-regular list assignment  $L$ .

**Question:** Determine whether it is possible to color  $n - k$  vertices of  $G$  with respect to  $L$ .

This is the same as asking whether we can delete  $k$  vertices and obtain a list coloring of the graph. If the lists (of size 2) of each vertex is the same, then note

that this problem is the same as asking whether there are  $k$  vertices whose deletion results in a bipartite graph, which is the well-studied odd cycle transversal (OCT) problem [11, 14]. We show that Vertex Parameterized 2-REGULAR LIST COLORING, which is a generalization of OCT, is also FPT.

We also consider a closely related problem that is whether we can delete  $k$  edges such that resultant graph can be colored satisfying the given 2-regular list assignment. We define the problem formally as follows:

Edge Parameterized 2-REGULAR LIST COLORING **Parameter:**  $k$   
**Input:** A graph  $G = (V, E)$  with a 2-regular list assignment  $L$ .  
**Question:** Can we delete at most  $k$  edges such that the resultant graph can be colored respecting  $L$ ?

This problem is generalization of the well known edge bipartization problem. We show that this problem is also FPT.

As mentioned earlier, 2-REGULAR LIST COLORING is polynomial time solvable. In a 2-REGULAR LIST COLORING instance, all vertices have lists of colors of size exactly 2. Another parameterization we consider is that if only  $k$  vertices have lists of colors of size more than 2, then how does the complexity of LIST COLORING change. We first define the problem formally.

Parameterized 2-REGULAR LIST COLORING **Parameter:**  $k$   
**Input:** A graph  $G = (V, E)$  and, a set  $S \subseteq V$ ,  $|S| \leq k$  and a list assignment  $L$  such that the size of lists of colors of vertices in  $V \setminus S$  is at most 2.  
**Question:** Determine whether it is possible to color the graph respecting  $L$

We show that Parameterized 2-REGULAR LIST COLORING is  $W[1]$ -hard. On the other hand, if the maximum size of lists of colors for the vertices in  $S$  is  $m$ , then we show that if  $m$  bounded by a constant or is also a parameter, then the problem becomes FPT.

## 1.4 Organization of the Paper

In Sect. 2 we show the PARA-NP-hardness results on planar  $+kv$  graphs and  $G_3 + kv$  graphs. Next we show that Parameterized 2-REGULAR LIST COLORING is  $W[1]$ -hard and that it becomes FPT with stronger parameterization or constant upper bound on the size of lists of colors. In Sect. 3 we present the XP algorithm for  $(n - k)$ -REGULAR LIST COLORING. In Sect. 4 we show that the Vertex Parameterized 2-REGULAR LIST COLORING and Edge Parameterized 2-REGULAR LIST COLORING are FPT.

## 1.5 Related Work

Besides the results of Cai [2] mentioned in the introduction, there are other works on parameterizing the vertex coloring problems on graphs which are close to some special families. Formally if  $F$  is a graph family, then  $F + kv$  is another family of graphs, which have only  $k$  vertices whose deletion results into  $F$  graph.

Now for example, COLORING problem, when parameterized by  $k$ , is W[1]-hard for chordal +  $kv$  graphs, interval +  $kv$  graphs and complete +  $kv$  graphs [12].

It is known that not every graph is  $k$ -CHOOSABLE. Therefore a natural question is, given a graph  $G$  and a LIST can we find a coloring in polynomial time. In a recent paper [5] Dabrowski et al. have addressed many such LIST COLORING problems.

## 2 Hardness Results

**Theorem 2.** 5-REGULAR LIST COLORING is PARA-NP-hard on planar +  $kv$  graphs.

*Proof.* We will prove that 5-REGULAR LIST COLORING is PARA-NP-hard on planar +  $1v$  graphs. It is known that 4-REGULAR LIST COLORING is NP-complete for planar graphs even if every list contains four colors from  $\{1, 2, 3, 4, 5\}$ [5]. We will show a polynomial reduction from this problem to our problem.

Suppose we are given an instance  $G = (V, E)$  with the 4-regular list assignment. We create 5 copies of this instance and call them  $G_1, G_2, G_3, G_4$  and  $G_5$ . Now we add a new color 6 in lists of all vertices of  $G_1$ . Similarly, we add colors 7, 8, 9 and 10 to all vertices of  $G_2, G_3, G_4$  and  $G_5$  respectively. Now we add a special vertex  $s$  with list of colors  $\{6, 7, 8, 9, 10\}$ . We make this vertex  $s$  adjacent to all vertices in  $G_i \forall 1 \leq i \leq 5$ . The resultant graph is planar +  $1v$  graph because each  $G_i$  is planar and union of  $G_i$ s is also a planar graph. Also each vertex in the new instance has 5 colors in the lists. Hence this coloring instance is 5-REGULAR LIST COLORING instance on planar +  $1v$  graph. We show that the new instance is colorable if and only if the original instance is an Yes instance.

If the original instance is a Yes instance, then each copy of the original vertices can be given the same colors and the vertex  $s$  can be given any color from its list. Conversely, if the new instance can be list colored, then we can color the original instance as follows: Say the vertex  $s$  is colored with color 6, then no vertex in  $G_1$  can get color 6, so we can color the original instance from the colors that corresponding vertices in  $G_1$  are colored from. Similarly, if  $s$  were colored from color 7, 8, 9 or 10, we would have considered  $G_2, G_3, G_4$  or  $G_5$  respectively.

Therefore 5-REGULAR LIST COLORING is NP-hard on planar +  $1v$  graph.  $\square$

**Theorem 3.**  $G_3 + kv$  COLORING is PARA-NP-hard.

*Proof.* We will show that 3-COLORING is NP-hard on  $G_3 + 3v$  graph.

It is known that LIST COLORING problem is NP-complete for 3-regular planar bipartite graphs that have list assignment in which each list is one of  $\{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}$  and all neighbours of each vertex with three colors in its list have two colors in their lists [3]. We will show a polynomial time reduction from this problem. Say we are given an instance  $G = (V, E)$  of this problem. We add three vertices  $a, b, c$ . We will add edge  $(a, v)$  for all  $v \in V$  which have  $\{2, 3\}$  in the list. We add edge  $(b, v)$  for all  $v \in V$  which have  $\{1, 3\}$  in the list. Similarly we add edge  $(c, v)$  for all  $v \in V$  which have  $\{1, 2\}$  in the list. Since the

given instance was a 3-regular planar bipartite graph, the resultant graph will be a  $G_3 + 3v$  graph. Now we claim that this new graph is 3-colorable if and only if the given instance is a Yes instance. If the original instance can be colored respecting the list assignment, then we can color vertices in  $V$  with the same colors and  $a$  with 1 because any neighbor of  $a$  will not have gotten color 1, since they had  $\{2, 3\}$  in their lists. Similarly we can color  $b$  with 2 and  $c$  with color 3.

Conversely, if the new instance is 3-colorable, then we can color the original instance respecting the given list assignment as follows: Say vertex  $a$  gets color  $i$  and so  $b$  and  $c$  cannot get the same color. Now say the set of vertices that get color  $i$  is  $V_a$ , then we will color them with color 1 in original instance. Similarly, we can color vertices in  $V_b$  with color 2 and vertices in  $V_c$  with color 3.

Therefore, 3-COLORING is NP-hard on  $G_3 + 3v$  graph which proves PARANP-hardness of COLORING on  $G_3 + kv$  graphs.  $\square$

**Theorem 4.** *Parameterized 2-REGULAR LIST COLORING is W[1]-hard.*

*Proof.* We will show a parameterized reduction from  $k$ -INDEPENDENT SET problem which is known to be W[1]-hard [4]. We are given an instance  $G = (V, E)$ . Now the problem is to find whether there exists an independent set of size at least  $k$ . With each vertex  $v_i \in V$ , we associate a list of colors  $\{i, 0\}$ . Now, we add a clique of size  $k$ , in which each vertex has list of colors  $\{1, 2, 3, \dots, n\}$ . Also, we make each vertex of this clique adjacent to all vertices of  $V$ . It can be observed that the new instance is an instance of original problem which we wanted to prove W[1]-hard because only  $k$  vertices have lists of size more than 2. Now we prove the claim that there exists an independent set of size at least  $k$  if and only if this graph can be colored respecting the list assignment.

Let  $S \subseteq V$  be an independent set of size  $k$  in  $G$ , then the vertices in  $S$  can be colored from color 0 and therefore the vertices of clique can use the  $k$  colors corresponding to the second colors in the lists of those  $k$  vertices in  $S$ . The remaining vertices of  $V \setminus S$  can be colored from the first colors in their respective lists. So the new instance can be colored.

Conversely, say the new instance can be colored respecting the list assignment. Since the vertices in clique must use  $k$  colors out of  $\{1, 2, 3, \dots, n\}$ , so at least  $k$  vertices in  $V$  must have been colored with the color 0 which is the desired independent set of size at least  $k$ . Hence, Parameterized 2-REGULAR LIST COLORING is W[1]-hard.  $\square$

Now, we will show that if the size of lists of colors for these  $k$  vertices is also bounded by  $m$  and  $m$  is also a parameter, then the problem becomes FPT.

**Theorem 5.** *Parameterized 2-REGULAR LIST COLORING is FPT with parameter  $(k, m)$ , where  $m$  is the maximum size of list of colors for the vertices in  $S$ .*

*Proof.* Consider the vertices in  $S$ . There are at most  $m^k$  ways to color these vertices. We try all these possibilities in at most  $m^k$  iterations and in each iteration, for the remaining vertices, the problem becomes 2-LIST COLORING

which is polynomial time solvable as shown in previous section. So if  $m$  is a constant or parameter, the algorithm is FPT algorithm and therefore the problem is FPT.  $\square$

### 3 $(n - k)$ -REGULAR LIST COLORING Is in XP

It is known that determining whether the vertices of a graph can be colored properly with  $n - k$  colors is fixed-parameter tractable when parameterized by  $k$  [4]. The exact parameterized complexity class of  $(n - k)$ -REGULAR LIST COLORING is still unknown. In this section we present, an XP algorithm for this problem.

Let  $G = (V, E)$  be any graph and  $L$  be a  $(n - k)$ -regular list assignment such that  $L(v)$  denotes the list of colors corresponding to the vertex  $v$ . We begin by observing the following rule which can be applied to the graph repeatedly until no longer possible.

**Reduction Rule 5.** *Delete any vertex with degree less than  $(n - k)$ .*

**Observation 6.** *Reduction Rule 5 is safe and can be implemented in polynomial time.*

*Proof.* Consider a vertex  $v$  which has degree  $d_v$  less than  $(n - k)$ . Since each vertex has  $(n - k)$  colors in its list, vertex  $v$  also has  $(n - k)$  colors. Coloring requires that no adjacent vertices get the same color, so  $v$  cannot be colored with the colors that its neighbours are colored from. Since  $v$  has less than  $(n - k)$  neighbours, so at least one color will always be left for  $v$  to be colored. Therefore it can be observed that  $G \setminus v$  can be colored respecting the list assignment if and only if  $G$  can be colored too. Checking the degrees of vertices and deleting a vertex from the graph are well known to be implemented in polynomial time.  $\square$

The following more general claim follows from the proof of correctness of the reduction rule above, as we can greedily color the vertices.

**Lemma 1.** *If the size of the list of every vertex is more than its degree then there exists a list coloring respecting the lists.*

We keep applying Reduction Rule 5 till it is no longer applicable and hence from now onwards we assume that every vertex has degree at least  $(n - k)$ .

We need the following simple observation which we use in our algorithm.

**Lemma 2.** *LIST COLORING is polynomial time solvable on a clique.*

*Proof.* We are given a clique  $C$  of size  $n$  and a list assignment  $L$  of colors. Since, in a clique, no two vertices can be assigned same color, therefore at most one vertex can be colored by any color. We create a bipartite graph, with all the vertices of  $C$  in one part, and a set of vertices, one corresponding to each distinct color in the lists of vertices in  $C$ , in the other part. We add an edge between a vertex  $v$  which corresponds to a vertex in  $C$  and a vertex  $x$  which corresponds



to a color  $x$ , if and only if the color  $x$  appears in the list of  $v$ . Now, it can be observed that the size of maximum matching gives us the maximum number of vertices in  $C$  that can be colored satisfying  $L$ . Finding maximum matching in a bipartite graph is polynomial time solvable [8]. Therefore LIST COLORING is polynomial time solvable on cliques.  $\square$

**Lemma 3.** *If there exists a set of  $k$  colors using which it is possible to color at least  $2k$  vertices of  $G$  respecting  $L$ , then there is a feasible coloring for  $G$  with respect to  $L$ .*

*Proof.* Let  $C_k$  be the set of  $k$  colors, from which it is possible to color  $s$  vertices, where  $s \geq 2k$ . Now, we assume that we will not use any color of  $C_k$  to color any other vertex. So we can delete these  $k$  colors from the lists of other vertices. Also we can delete those  $s$  vertices from the graph, because we will not use their colors in the remaining graph. Now the size of the list of colors of any vertex will decrease by at most  $k$ . Hence each vertex will still have at least  $n - 2k$  colors in its list. Since we are left with at most  $n - s$  vertices in the graph and  $s \geq 2k$ , each vertex can have degree at most  $n - 2k - 1$  and has at least  $n - 2k$  colors in its list. So from Lemma 1, there is a feasible coloring respecting  $L$ .  $\square$

**Lemma 4.** *If there does not exist a set of  $k$  colors using which it is possible to color more than or equal to  $2k$  vertices of  $G$  respecting  $L$ , then, in time  $n^{O(k)}$ , we can determine whether  $G$  has a feasible coloring with respect to  $L$  or not.*

*Proof.* Let  $R$  be the set of colors that are used by more than one vertex in the optimal coloring if it exists. Since there does not exist any set of  $k$  colors using which at least  $2k$  vertices can be colored,  $|R| < k$ . Let  $C$  be the union of the lists of colors of all vertices in  $V$ . Clearly  $|C| < n(n - k)$ , and hence the number of all possible subsets of  $C$  of size  $k$  is less than  $n^{2k}$ . Therefore, in time  $n^{O(k)}$ , we can guess a set  $S$  of colors of size  $k$  such that  $R \subseteq S$ . Now, we will guess all possible sets of vertices that can be colored from the colors in  $S$ .

We denote the set of vertices that have color  $i$  in their lists by  $V_i$ . It can be observed that the size of any  $V_i$  can be at most  $n$ . Also, since the degree of each vertex is at least  $(n - k)$ , the total number of independent sets in any  $V_i$  is at most  $n \cdot 2^k$  (as a vertex is non-adjacent to at most  $k$  other vertices). Therefore the total possible sets of vertices, that can be colored from the colors in  $S$ , is at most  $(n \cdot 2^k)^k$ . Since, we are sure that colors other than those in  $S$  don't color more than one vertex in the graph, the remaining vertices get a unique color. We can remove the colors of  $S$  from the lists of the remaining vertices and make them all adjacent to each other (as no pair of them will be colored with the same color anyway). Thus the problem now reduces to list coloring a clique which can be solved in polynomial time using Lemma 2. It can be observed that we have maintained the overall complexity upper bounded by  $n^{O(k)}$ .  $\square$

**Theorem 7.** *In time  $n^{O(k)}$  given  $G = (V, E)$  and  $(n - k)$  regular list  $L$ , algorithm 1 determines whether there exist a color assignment respecting  $L$ .*

*Proof.* Let the total number of colors be  $r$ . Now the idea is to build  $r$  graphs, one corresponding to each color, which will be induced from the vertices which have that corresponding color in their list. Now, the task is simply to find sets of independent vertices from the graphs corresponding to each color such that the union covers all vertices. Since we know that each vertex has degree at least  $(n - k)$ , there can be at most  $n \cdot 2^k$  independent sets in each graph. So we iterate over all possibilities of choosing  $k$  colors and then the vertices that can be colored from them (by choosing independent sets from their corresponding graphs) in  $n^{O(k)}$  time. We now have two possibilities.

1. Suppose, we can choose  $k$  colors such that we can color at least  $2k$  vertices from those colors, then we can return Yes (from Lemma 3).
2. In the other case, there cannot be more than  $k$  colors which contribute to more than 1 vertex in the optimum solution. Now from Lemma 4, we keep guessing  $k$  colors and assume that those colors which contribute to more than 1 color in the optimum solution will be covered in some iteration. In that case, since the remaining colors contribute at most one vertex, finding the size of the maximum matching in the constructed graph provides the solution.

Observe that the run time of the Algorithm is dominated by line number 13 and 15 which are  $n^{O(k)}$ . □

## 4 Deleting $k$ Vertices or Edges to Satisfy 2-REGULAR LIST COLORING

**Theorem 8.** *Vertex Parameterized 2-REGULAR LIST COLORING is FPT.*

*Proof.* We modify the proof of Theorem 1 for this. Let  $G = (V, E)$  be a graph and  $L$  be a 2-regular list for  $G$ . We reduce our problem to All-but- $k$  2-SAT. All-but- $k$  2-SAT is to determine whether in given a 2-CNF formula, it is possible to remove at most  $k$  clauses so that the resulting 2-CNF formula is satisfiable. It is known that All-but- $k$  2-SAT is FPT [10, 13]. From  $G$  we create an equivalent 2-CNF formula  $\mathcal{F}$  as follows.

For each  $u \in V$ , we make two variables  $x_u$  and  $y_u$ . So, we have  $2|V|$  variables. We will set  $x_u$  to true if the vertex  $u$  is assigned the first color in its list and  $y_u$  to true and  $x_u$  to false if vertex  $u$  is assigned the second color in its list. Now we construct a 2-CNF formula by adding two types of clauses.

- **Type 1:** For each vertex  $u \in V$ , we add the clause  $(\overline{x_u} \vee \overline{y_u})$ . Also for each edge  $(u, v)$ , we add clauses such that both vertices do not get the same color. For example, if the list for vertex  $u$  is  $\{a, b\}$  and that for vertex  $v$  is  $\{b, c\}$ , then we add clause  $(\overline{y_u} \vee \overline{x_v})$ .

After adding the clauses corresponding to all vertices and edges, we repeat each of these Type 1 clauses  $k + 1$  times.

- **Type 2:** We add clauses  $(x_u \vee y_u)$  for all  $u \in V$ .

**Algorithm 1.**  $(n - k)$ -REGULAR LIST COLORING

---

```

1: Input : Graph  $G = (V, E)$  and  $(n - k)$  regular list assignment  $L$  such that  $L(v)$ 
   denotes the list of colors corresponding to the vertex  $v$ .
2: Parameter :  $k$ .
3: Output : Yes if  $G$  can be colored respecting the given list assignment, No
   otherwise.
4: Recursively delete vertices of degree less than  $n - k$ .
5:  $R \leftarrow \cup_{v \in V} L(v)$ 
6:  $r \leftarrow |R|$ 
7: for  $i \leftarrow 1$  to  $r$  do
8:    $V_i \leftarrow \{v \in V \mid i \in L(v)\}$ 
9:    $E_i \leftarrow \{(u, v) \in E \mid u \in V_i \text{ and } v \in V_i\}$ 
10:   $C_i \leftarrow (V_i, E_i)$ 
11: end for
12:  $C \leftarrow \bigcup_r^{i=1} C_i$ 
13: for each  $S \subseteq C$  such that  $|S| = k$  do
14:   Let  $S$  be  $\{S_1, S_2, S_3, \dots, S_k\}$ 
15:   for each possible combination  $\{T_1, T_2, T_3, \dots, T_k\}$  where  $T_i \subseteq V(S_i) \forall 1 \leq i \leq k$ 
   and  $T_i$  is independent in  $S_i$  do
16:      $T \leftarrow \bigcup_k^{i=1} T_i$ 
17:     if  $|T| \geq 2k$  then
18:       return Yes;
19:     else
20:        $G' \leftarrow G \setminus T$ 
21:        $C_M \leftarrow$  New vertices corresponding to colors in  $G'$ 
22:        $V_M \leftarrow V(G') \cup C_M$ 
23:       construct  $G_M$  from vertices  $V_M$  with adding edges between the vertex  $v$ ,
   which corresponds to a vertex from  $G'$ , and a vertex corresponding to color  $c$  if and
   only if  $c \in L(v)$ 
24:        $M \leftarrow$  maximum matching in  $G_M$ 
25:       if  $|T| + |M| \geq n$  then
26:         return Yes;
27:     end for
28: end for
return No

```

---

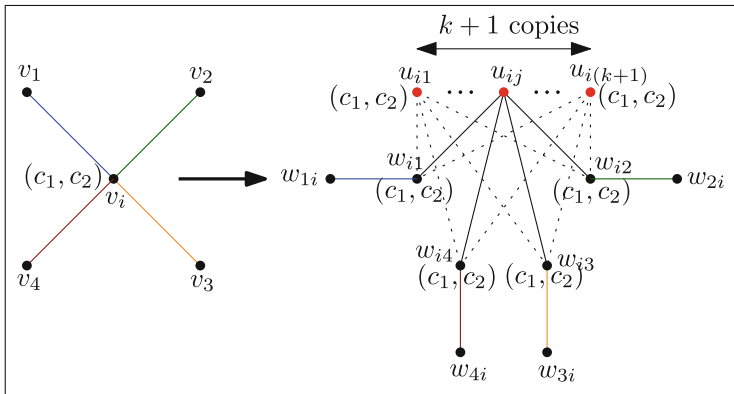
The resultant formula is the equivalent 2-CNF formula. Now, we claim that  $\exists S \subseteq V, |S| \leq k$ , such that the graph induced from  $V - S$  is list colorable if and only if at least  $(m - k)$  clauses of the resultant expression can be satisfied, where  $m$  is the total number of clauses. We can note that  $m \leq (k + 1)(2|E|) + |V| + |V|$ . Suppose that there exists an assignment of variables that satisfies at least  $m - k$  clauses i.e. at most  $k$  clauses are falsified by this assignment. Now, these  $k$  clauses can be Type 2 only, because each of Type 1 clauses appears  $k + 1$  times. These  $k$  clauses will be corresponding to  $k$  vertices in  $V$ . Since the rest of all the clauses in SAT expression can be satisfied, so in our graph, we can delete those  $k$  vertices, to color the remaining  $n - k$  vertices. Conversely suppose that there exists a  $S \subseteq V, |S| \leq k$ , such that induced graph from  $V \setminus S$  can be colored. We can

assign *false* to both  $x_u$  and  $y_u$  for each such vertex  $u \in S$ . So at most  $k$  Type 2 clauses will be falsified. It can easily be seen that the rest all clauses can be satisfied.

Therefore the FPT reduction is proven. Since we know that All-but- $k$  2-SAT is FPT, so this problem is also FPT.  $\square$

As mentioned before, this result generalizes the result that the odd cycle transversal problem is fixed-parameter tractable. There is a related edge version (called edge bipartization) which asks whether we can delete  $k$  edges to make a graph bipartite. An analogous list-coloring version is whether we can delete  $k$  edges from a given graph to satisfy a 2-regular list coloring. As in the case of edge bipartization [16], we give a parameter preserving reduction from the edge version to the vertex version to show

**Theorem 9.** *Given a 2-regular list coloring instance, determining whether there are  $k$  edges from the graph whose removal results in a list 2-coloring of the graph is fixed-parameter tractable.*



**Fig. 1.** Example reduction from  $G_1$  to  $G_2$

*Proof.* The proof goes along the lines of the parameter preserving reduction from edge bipartization to odd cycle transversal. Given an instance  $G_1 = (V_1, E_1)$  of Edge Parameterized 2-REGULAR LIST COLORING problem, we will reduce to equivalent instance  $G_2 = (V_2, E_2)$  of Vertex Parameterized 2-REGULAR LIST COLORING problem. From  $G_1$  we construct  $G_2$  as follows.  $V_2$  consists of two types of vertices:

**Type 1:**  $V_I = \{u_{ij} \mid v_i \in V_1, 1 \leq j \leq k + 1\}$

**Type 2:**  $V_{II} = \{w_{ij}, w_{ji} \mid \forall (v_i, v_j) \in E_1\}$

For a sample reduction please refer to Fig. 1. Now we set  $V_2 = V_I \cup V_{II}$ . Note that we have named the vertices in  $V_I$  by a  $u$  and those in  $V_{II}$  by a  $w$ . We can call vertices in  $V_I$  by  $u$  vertices and those in  $V_{II}$  by  $w$  vertices. The edges in  $E_2$  also consist of two sets:

**Type 1:**  $E_I = \{ \{u_{ij}, w_{im}\} \mid v_i \in V_1, 1 \leq j \leq k + 1, (v_i, v_m) \in E_1 \}$

**Type 2:**  $E_{II} = \{ \{w_{ab}, w_{ba}\} \mid e_l = \{v_a, v_b\} \in E_1 \}$

Now we set  $E_2 = E_I \cup E_{II}$ . We give the list assignment of colors in  $G_2$  in the following manner. List of colors for each  $u_{ij}$  is same as the list of  $v_i$  in  $G_1$ . Also list for  $w_{ij}$  is same as the list of  $v_i$  in  $G_1$ . Construction of equivalent Vertex Parameterized 2-REGULAR LIST COLORING instance is complete now. Now we show that the edge parameterized instance  $G_1$  can be colored if and only if vertex parameterized instance  $G_2$  can be colored.

First assume there exists a set of  $k$  edges  $E_k$  in  $G_1$  such that after deleting  $E_k$  from  $G_1$ , remaining graph can be colored. Then, for each edge  $e_l = (v_a, v_b) \in E_k$ , we can delete any one of  $w_{ab}$  or  $w_{ba}$ . This step deletes exactly  $k$  vertices. Now, in remaining  $G_2$ , say the list of colors for the vertex  $u_{ij}$  is  $\{c_1, c_2\}$  which is same as of  $v_i$  in  $G_1$ . If  $v_i$  is colored with  $c_1$ , then we color  $u_{ij}$  with  $c_2$  and if  $v_i$  is colored with  $c_2$ , then we color  $u_{ij}$  with  $c_1$ . For each vertex  $w_{ij}$ , we can color it with the same color as of  $v_i$ . It can be observed that there won't be any violations in the coloring.

Now for other way proof, assume there exists a set  $S_k \subset V_2$  of at most  $k$  vertices whose deletion makes graph  $G_2$  colorable. It can be observed that for each  $i$ , the set  $\{u_{ij} \mid 1 \leq j \leq k + 1\}$  will be colored with same color, because otherwise there will be no color left for some  $w$  vertex. Also it should be noted that deleting any  $u$  vertex does not help in coloring, because in that case we need to delete  $k + 1$  copies to get rid of the constraint caused by it. So, all vertices in  $S_k$  are  $w$  vertices. Now, we can delete the  $k$  edges in graph  $G_1$  which correspond to the  $k$  vertices in  $S_k$ . It can then be observed that the vertex  $v_i$  in  $G_1$  can be colored from the color which is not used by  $u_{ij}$  and this doesn't violate any coloring constraint.  $\square$

## 5 Conclusions

We have filled up some gaps in the parameterizations of coloring and list-coloring and shown FPT results and hardness results for some natural parameterization. The main open problem we leave with is whether  $(n - k)$ -REGULAR LIST COLORING is fixed-parameter tractable.

## References

1. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.* **8**(3), 121–123 (1979)
2. Cai, L.: Parameterized complexity of vertex colouring. *Discrete Appl. Math.* **127**(3), 415–429 (2003)
3. Chudnovsky, M.: Coloring graphs with forbidden induced subgraphs. *Proc. ICM* **4**, 291–302 (2014)
4. Cygan, M., Fomin, F.V., Kowalik, L., Lokshantov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: *Parameterized Algorithms*. Springer, Cham (2015). <https://doi.org/10.1007/978-3-319-21275-3>

5. Dabrowski, K.K., Dross, F., Johnson, M., Paulusma, D.: Filling the complexity gaps for colouring planar and bounded degree graphs. In: Lipták, Z., Smyth, W.F. (eds.) IWOCA 2015. LNCS, vol. 9538, pp. 100–111. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-29516-9\\_9](https://doi.org/10.1007/978-3-319-29516-9_9)
6. Dailey, D.P.: Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Math.* **30**(3), 289–293 (1980)
7. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. *Theor. Comput. Sci.* **1**(3), 237–267 (1976)
8. Hopcroft, J.E., Karp, R.M.: An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* **2**(4), 225–231 (1973)
9. Khot, S., Raman, V.: Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.* **289**(2), 997–1008 (2002)
10. Lokshtanov, D., Narayanaswamy, N.S., Raman, V., Ramanujan, M.S., Saurabh, S.: Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms* **11**(2), 15:1–15:31 (2014)
11. Lokshtanov, D., Saurabh, S., Sikdar, S.: Simpler parameterized algorithm for OCT. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 380–384. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-10217-2\\_37](https://doi.org/10.1007/978-3-642-10217-2_37)
12. Paulusma, D.: Open problems on graph coloring for special graph classes. In: Mayr, E.W. (ed.) WG 2015. LNCS, vol. 9224, pp. 16–30. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53174-7\\_2](https://doi.org/10.1007/978-3-662-53174-7_2)
13. Razgon, I., O’Sullivan, B.: Almost 2-SAT is fixed-parameter tractable. *J. Comput. Syst. Sci.* **75**(8), 435–450 (2009)
14. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. *Oper. Res. Lett.* **32**(4), 299–301 (2004)
15. Thomassen, C.: Every planar graph is 5-choosable. *J. Comb. Theory Ser. B* **62**(1), 180–181 (1994)
16. Wernicke, S.: On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems. Ph.D. thesis. Universität Tübingen, Tübingen (2003)



# Kernelization for $P_2$ -Packing: A Gerrymandering Approach

Wenjun Li<sup>1</sup>, Junjie Ye<sup>2</sup>(✉), and Yixin Cao<sup>2</sup>(ID)

<sup>1</sup> Changsha University of Science and Technology, Changsha, China  
liwenjun@csu.edu.cn

<sup>2</sup> Department of Computing, Hong Kong Polytechnic University, Hong Kong, China  
{junjie.ye,yixin.cao}@polyu.edu.hk

**Abstract.** The  $P_2$ -packing problem asks for whether a graph contains  $k$  vertex-disjoint paths each of length two. We continue the study of its kernelization algorithms, and develop a  $5k$ -vertex kernel.

## 1 Introduction

Packing problems make one of the most important family of problems in combinatorial optimization. One example is  $H$ -packing for a fixed graph  $H$ , i.e., to find the maximum number of vertex-disjoint copies of  $H$  from a graph  $G$ . It is trivial when  $H$  consists of a single vertex, and it is the well-known maximum matching problem, which can be solved in polynomial time, when  $H$  is an edge. The problem can be easily reduced to the maximum matching problem when each component of  $H$  has at most two vertices. The smallest  $H$  on which the  $H$ -packing problem is NP-complete is  $P_2$ , the graph on three vertices and two edges [11]. The  $P_2$ -packing problem is thus a natural starting point of investigating  $H$ -packing problems in general, and has been extensively studied [7, 9, 10, 12].

In the parameterized setting, the  $P_2$ -packing problem asks whether a graph  $G$  contains  $k$  vertex-disjoint paths each of length two. Recall that given an instance  $(G, k)$ , a *kernelization algorithm* produces in polynomial time an equivalent instance  $(G', k')$ — $(G, k)$  is a yes-instance if and only if  $(G', k')$  is a yes-instance—such that  $k' \leq k$ . The size of  $G'$  is upper bounded by some function of  $k'$ , and  $(G', k')$  is a *polynomial kernel* when the function is a polynomial function. Prieto and Sloper [13] first developed a  $15k$ -vertex kernel for the  $P_2$ -packing problem, which were improved to  $7k$  [14] and then  $6k$  [2]. We further improve it to  $5k$ .

---

Supported in part by NSFC under grant 61502054, RGC under grant PolyU 252026/15E, NSFC under grant 61572414, Natural Science Foundation of Hunan Province under grant 2017JJ3333 and Scientific Research Fund of Hunan Provincial Education Department under grant 17C0047.

**Theorem 1.** *The  $P_2$ -packing problem has a  $5k$ -vertex kernel.*

Although our improvement seems modest, it is a solid step toward the ultimate goal of this line of research, a kernel of only  $3k$  vertices. Note that the problem remains NP-hard when  $G$  has exactly  $3k$  vertices. Indeed, what Kirkpatrick and Hell [11] proved is that it is NP-hard to decide whether a graph can be partitioned into vertex-disjoint  $P_2$ 's. The existence of a  $3k$ -vertex kernel for the  $P_2$ -packing problem would indicate that it is practically equivalent to the  $P_2$ -partition problem. Moreover, our algorithm implies directly an approximation algorithm of ratio  $5/3$ ; a  $4.5k$ -vertex kernel, provided that it satisfies certain properties, would imply an approximation algorithm of ratio 1.5, better than the best known ratio  $1.5 + \epsilon$  [6]. We remark that the problem is MAX-SNP-hard [8], and remains so even on bipartite graphs with maximum degree three [12].

We also note that there are efforts on a simplified version of the problem: Chang et al. [1] claimed a  $5k$ -vertex kernel for the problem on net-free graphs, which however contains a critical bug, according to Xiao and Kou [15].

A very natural tool for the problem is a generalization of the crown structure; it was first used by Prieto and Sloper [13], and all later work follows suit. If there exists a set  $C$  of vertices such that there are precisely  $|N(C)|$  vertex-disjoint  $P_2$ 's in the subgraph induced by  $N(C) \cup C$ , but  $G[C]$  does not contain any  $P_2$ , then we may take the  $|N(C)|$  paths and consider the subgraph  $G - N(C) \cup C$ . This remains our main reduction rule; the difficulty, hence one of our contributions, is how to find such a structure if one exists.

As all the previous kernelization algorithms for the problem, we start from finding a maximal  $P_2$ -packing  $\mathcal{P}$  in a greedy way. Let  $V(\mathcal{P})$  denote the vertices on paths in  $\mathcal{P}$ , and we call other vertices, i.e.,  $V(G) \setminus V(\mathcal{P})$ , extra vertices. We may assume that the graph contains no component of one or two vertices. Then each component in  $G - V(\mathcal{P})$  has to be connected with  $V(\mathcal{P})$ . By some classic results from matching theory, as long as the number of extra vertices is large enough, a reducible structure can be identified. This leads to the first kernel of  $15k$  vertices [13], and is the starting point of all later work.

The later improvements follow a similar scheme. It tries to find a  $P_2$ -packing larger than  $\mathcal{P}$  using local search; once the local search gets stuck, a careful study of the configuration would reveal new reducible structures. Wang et al. [14] observed that a pair of adjacent extra vertices is more helpful for the local search than two nonadjacent ones. They used two simple exchange rules to consolidate extra vertices that are only adjacent to vertices in  $V(\mathcal{P})$ . For example, if the two ends of a path on five vertices are extra vertices, (i.e., the three vertices in the middle are picked to be a path in  $\mathcal{P}$ ,) they would change it so that the two vertices not picked are adjacent. The key idea of Chen et al. [2] is that extra vertices adjacent to the ends of a path in  $\mathcal{P}$  are usually more helpful than those adjacent to the middle vertex of the path.

Our local search procedure is more systematic and comprehensive; it actually subsumes observations from both Wang et al. [14] and Chen et al. [2]. After the initial step very similar to [13], if no reducible structure has been found, we assign the extra vertices to paths in  $\mathcal{P}$  such that each path receives a small number of them. Each path, together with assigned vertices, defines a unit. We



put units with at least five vertices into two categories, depending on whether there is a vertex that participates in all  $P_2$ 's inside this unit. We introduce several nontrivial exchange rules to migrate vertices from “large” units to “small” units. Their applications may lead to (1) a larger  $P_2$ -packing than  $\mathcal{P}$ , or (2) a reducible structure, whereupon we repeat the procedure with a larger number of units or a smaller graph respectively. After they are exhaustively applied, a unit contains at most six vertices, and the number of six-vertex units is upper bounded by the number of small units (on four or three vertices). The bound on the size of the kernel follows immediately.

## 2 Preliminaries

All graphs discussed in this paper are undirected and simple. The vertex set and edge set of a graph  $G$  are denoted by  $V(G)$  and  $E(G)$  respectively. For a set  $U \subseteq V(G)$  of vertices, we denote by  $G[U]$  the subgraph induced by  $U$ , whose vertex set is  $U$  and whose edge set comprises all edges of  $G$  with both ends in  $U$ . We use  $G - U$  as a shorthand for  $G[V(G) \setminus U]$ , and it is further simplified as  $G - v$  when  $U$  contains a single vertex  $v$ . A *component* is a maximal connected induced subgraph, and an *edge component* is a component on two vertices.

**Reduction Rule 1.** *If a component  $C$  of  $G$  has at most 6 vertices, delete  $C$  and decrease  $k$  by the maximum number of vertex-disjoint  $P_2$ 's in  $C$ .*

The following technical definition would be crucial for our main reduction rule.

**Definition 1.** *Let  $C$  be a set of vertices and  $N(C) = \{v_1, v_2, \dots, v_\ell\}$ . We say that  $C$  is a reducible set (of  $G$ ) if the maximum degree in  $G[C]$  is at most one and one of the following holds.*

- (i) *There are  $\ell$  edge components  $\{C_1, \dots, C_\ell\}$  in  $G[C]$  such that  $v_i$  is adjacent to  $C_i$  for  $1 \leq i \leq \ell$ .*
- (ii) *There are  $2\ell$  components  $\{C_1, \dots, C_{2\ell}\}$  in  $G[C]$  such that  $v_i$  is adjacent to  $C_{2i-1}$  and  $C_{2i}$  for  $1 \leq i \leq \ell$ .*

For readers familiar with previous work, a remark is worthwhile here. Our reducible set is a generalization of the well-known crown decomposition [3]. Our definition (i) coincides with the “fat crown” defined in [13]. Our definition (ii) coincides with the “double crown” defined in [13] when each component of  $G[C]$  is a single vertex. In definition (ii), however, we allow a mixed of single-vertex components and edge components.<sup>1</sup>

**Reduction Rule 2.** *If there is a reducible set  $C$ , delete  $N(C) \cup C$  and decrease  $k$  by  $|N(C)|$ .*

<sup>1</sup> One may define the reducible set in a way that an edge component is regarded as two single-vertex components. This definition might reveal more reducible sets. However, it would slightly complicate our presentation without helping our analysis in the worst case, and hence we choose to use the simpler one.

We use  $\text{opt}(G)$  to denote the maximum number of vertex-disjoint  $P_2$ 's in graph  $G$ .

**Lemma 1.** *If  $C$  is a reducible set of graph  $G$ , then  $\text{opt}(G) = \text{opt}(G - N(C) \cup C) + |N(C)|$ .*

*Proof.* Let  $A = N(C)$  and  $G' = G - A \cup C$ . For each vertex  $v \in A$ , we can pick in  $G[C]$  an edge component or two vertices from two components to form a  $P_2$ . Thus we have  $|A|$  vertex-disjoint  $P_2$ 's using only vertices in  $A \cup C$ . Together with a maximum  $P_2$ -packing of  $G'$ , we have  $\text{opt}(G) \geq \text{opt}(G') + |A|$ .

On the other hand, any  $P_2$ -packing of  $G$  contains at most  $|A|$  vertex-disjoint  $P_2$ 's involving vertices in  $A$ . Hence  $\text{opt}(G) \leq \text{opt}(G - A) + |A|$ . By definition, the maximum degree in  $G[C]$  is at most one, and hence vertices of  $C$  participate in no  $P_2$  in  $G - A$ . Therefore,  $\text{opt}(G - A) = \text{opt}(G')$  and  $\text{opt}(G) \leq \text{opt}(G') + |A|$ .  $\square$

To identify reducible sets, we will rely on tools from maximum matching in bipartite graphs. In several steps of our algorithm, we will formulate an auxiliary bipartite graph  $B$ ; to avoid confusion, we use nodes to refer to elements in  $V(B)$ . The two sides of  $B$  are denoted by  $L$  and  $R$ . The Hall's theorem states that there is a matching of  $B$  saturating  $L$  if and only if  $|L'| \leq |N(L')|$  for all  $L' \subseteq L$  [4]. We will use the Hopcroft-Karp algorithm [5]: In polynomial time we can find either a matching of  $B$  saturating  $L$  or an inclusion-wise minimal set  $L' \subseteq L$  such that  $|N(L')| < |L'|$ . Note that there is a matching between  $N(L')$  and  $L'$  that saturates  $N(L')$ : Otherwise by the Hall's theorem, there exists  $R' \subseteq N(L')$  such that  $|R'| > |N(R') \cap L'|$ , but then  $L^* = L' \setminus N(R')$  also satisfies  $|N(L^*)| < |L^*|$ , contradicting the minimality of  $L'$ . Here  $N(R') \cap L'$  is nonempty because every node in  $R'$  is a neighbor of some node in  $L'$ .

**Lemma 2** ([5]). *Given a bipartite graph  $B$ , we can find in polynomial time a matching saturating  $L$  or a set  $L' \subseteq L$  such that there is a matching between  $N(L')$  and  $L'$  that saturates  $N(L')$ .*

### 3 The Unit Partition

Following the standard starter, our first step is to find a maximal  $P_2$ -packing  $\mathcal{P}$  of the input graph  $G$ . We will use these paths as “seeds” to partition  $V(G)$  into fewer than  $k$  units. We then locally change the units so that they satisfy certain properties. During the process, if we find (1) a  $P_2$ -packing larger than  $\mathcal{P}$ , or (2) a reducible set, then we restart the procedure with a new  $P_2$ -packing, or a new graph respectively.

Denote by  $V(\mathcal{P})$  the set of vertices in the paths in  $\mathcal{P}$ . The maximality of  $\mathcal{P}$  guarantees that each component of the subgraph  $G - V(\mathcal{P})$  is either a single vertex or an edge. We construct an auxiliary bipartite graph  $B_1$  as follows:

- for each component  $C$  of  $G - V(\mathcal{P})$ , introduce a node  $u_C$  into  $L$ ;
- for each vertex  $v \in V(\mathcal{P})$ , introduce two nodes  $v^1, v^2$  into  $R$ ; and

- add edges  $u_C v^1$  and  $u_C v^2$  if vertex  $v$  is adjacent to component  $C$  (of  $G - V(\mathcal{P})$ ) in  $G$ .

**Lemma 3.** *If there is no matching of  $B_1$  saturating all nodes in  $L$ , then we can find in polynomial time a reducible set.*

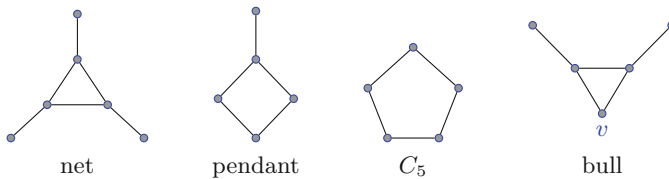
*Proof.* By Lemma 2, we find in polynomial time a subset  $L' \subseteq L$  such that there is a matching of  $B_1$  between  $N_{B_1}(L')$  and  $L'$  that saturates all nodes in  $N_{B_1}(L')$ . Let  $C'$  be the vertices in the components represented by nodes in  $L'$ , and let  $A'$  be the set of vertices represented by nodes in  $N_{B_1}(L')$ . We claim that  $C'$  is a reducible set. Note that for each vertex  $v \in V(\mathcal{P})$ , the set  $N_{B_1}(L')$  contains either both or neither of  $\{v^1, v^2\}$ . For each  $v \in A'$ , the two components in  $G[C']$ , whose nodes are matched to  $v^1$  and  $v^2$ , are adjacent to  $v$ . By the construction of  $B_1$ ,  $G[C']$  has maximum degree at most one and  $N(C') = A'$ . Hence  $C'$  is a reducible set.  $\square$

In the following, we may assume that we have a matching  $M$  of  $B_1$  saturating all nodes in  $L$ . For a path  $P \in \mathcal{P}$  on vertices  $u, v, w$ , we create a *unit* that contains  $u, v, w$ , and all vertices in those components matched to nodes  $u^1, u^2, v^1, v^2, w^1, w^2$  by  $M$ . Abusing the notation, we also use unit to refer to the subgraph induced by it. The path  $P$  is the *base path* of this unit. Since all nodes in  $L$  are matched in  $M$ , the collection of units is a partition of the vertex set  $V(G)$ , and we call it an *unreduced unit partition*.

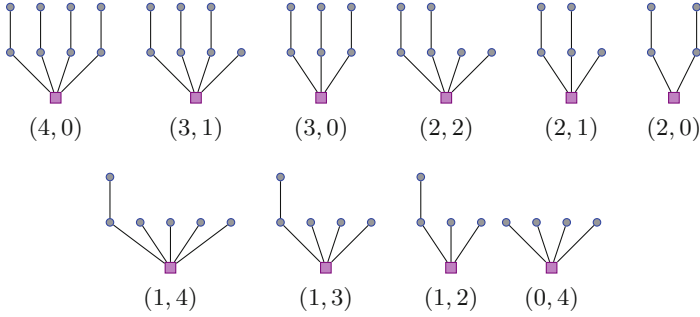
**Exchange Rule 1.** *If a unit contains two vertex-disjoint  $P_2$ 's, then replace the base path of this unit with these two  $P_2$ 's.*

Exchange Rule 1 enlarges the  $P_2$ -packing and significantly slashes the number of possible configurations of units. In the following we may assume that a unit has precisely one vertex-disjoint  $P_2$ .

We say that a unit is *democratic* if it contains one of the graphs in Fig. 1 as a subgraph. We call a democratic unit a *net*-, *pendant*-,  $C_5$ -, or *bull*-unit if it contains net, pendant,  $C_5$ , or bull but none of the previous ones as a subgraph. This order ensures, among others, that a bull-unit has to be an induced bull. (Indeed, only pendant-unit can have extra edges.) A bull-unit contains a unique vertex of degree 2, which we call the *nose* of the bull-unit. It is easy to check that there remains a  $P_2$  in a democratic unit after any vertex removed.



**Fig. 1.** Four subgraphs characterizing democratic units. A pendant-unit may contain extra edges not shown here, while the other three cannot. The vertex  $v$  is the nose of the bull-unit.



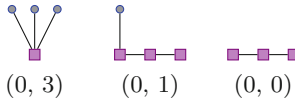
**Fig. 2.** Ten subgraphs characterizing despotic units. The square vertices are the core vertices, while the round vertices make the peripheral. Each edge component of the peripheral is a twig, and each single-vertex component is a leaf. Each unit in the first row has at least two twigs, while the second row has at most one. For the units with the same number of twigs, they are ordered by the number of leaves (hence the total number of vertices). Note that a unit may contain extra edges.

A unit that has more than four vertices but is not democratic is called *despotic*. Each graph  $F$  in Fig. 2 has a special vertex  $v$  (the square vertex at the bottom) such that its removal leaves a graph of maximum degree at most one. In other words, each component of  $F - v$  is an edge or an isolated vertex; we call them a *twig* and a *leaf* respectively. We label  $F$  by a pair  $(a, b)$ , which are the numbers of, respectively, twigs and leaves of  $F$ . We say that a despotic unit is an  $(a, b)$ -unit if it can be made, by deleting edges, graph  $(a, b)$  but not graph  $(a', b')$  with  $a' > a$ . A consequence of enforcing this order (of maximizing twigs) is that there cannot be any edge between two leaves in any unit: For example, if an edge is added to connect the two leaves of graph  $(2, 2)$ , then it also contains graph  $(3, 0)$  as a subgraph. For a unit  $U$ , we also use  $d_1(U)$  and  $d_2(U)$  to denote the numbers of, respectively, twigs and leaves; i.e.,  $d_1(U) = a$  and  $d_2(U) = b$  when  $U$  is an  $(a, b)$ -unit. The special vertex is the *core*, while all other vertices (including twigs and leaves) the *peripheral*, of the unit.

In passing we should mention that although we draw twigs in the way that only one vertex in a twig is adjacent to the core, we do not actually differentiate them (disregarding whether only one or both of them are adjacent to the core).

We are left with the *small units* (of three or four vertices), which turn out to be singular in our algorithm. Although they are the smallest units, great care is needed to deal with them. Recall that our aim is to bound the number of vertices by summing all units in final graph; hence we would like to maximize the number of small units. In this sense, the role of a small unit as an “exporter” would be marginal, and hence we do not categorize them into many types. We abuse the notation to denote them in a similar way as despotic units. A four-vertex unit is a  $(0, 3)$ -unit if it has precisely three edges and all of them have a common end, and a  $(0, 1)$ -unit otherwise. A three-vertex unit is a  $(0, 0)$ -unit, disregarding whether it has two or three edges. See Fig. 3 for an illustration of small units.

Note that  $(0, 1)$ -unit and  $(0, 0)$ -unit are special in the sense that they have three core vertices.



**Fig. 3.** Characterizations of small units. The square vertices are the core vertices, while the round vertices make the peripheral. Note that a  $(0, 3)$ -unit cannot contain extra edges, while the other two can.

We need to verify that the types defined above include all the possible units generated. We remark that  $(1, 4)$ -units do not exist in an unreduced unit partition, and they can only be introduced by exchange rules to be discussed in the next section.

**Proposition 1 (\*).**<sup>2</sup> *On an unreduced unit partition, if Exchange Rule 1 is not applicable, then the following hold.*

- (i) *Each unit on five or more vertices is either democratic or despotic; and if it is despotic, its type is not  $(1, 4)$ .*
- (ii) *Each unit on three or four vertices is a small unit in Fig. 3.*
- (iii) *In an  $(a, b)$ -unit, any edge not shown in the graph  $(a, b)$  is incident to one core vertex.*

Recall that when producing the unit partition, we assign each component of  $G - V(\mathcal{P})$  to the same  $P_2$  in  $\mathcal{P}$ . In other words, if there exists an edge between two units, at least one end of this edge is in the their base paths. We however lose this property with the new partition of core vertices and peripheral vertices: There might be edges between peripheral vertices and democratic units, and edges between two peripheral vertices in different units. We now apply the following rules to restore it, whose correctness is straightforward. Since at most three units are involved, the exchange rules can be applied in polynomial time.

**Exchange Rule 2.** *If any of the following holds true, we produce a larger  $P_2$ -packing than  $\mathcal{P}$ .*

- (i) *There is an edge between a vertex  $u_1$  of a democratic unit and a vertex  $u_2$  of another unit where  $u_1$  is not a nose and  $u_2$  is not a core vertex.*
- (ii) *There is an edge between the nose of a bull-unit and a twig of a despotic unit.*
- (iii) *There is an edge between a twig of a despotic unit and a peripheral vertex of another unit.*
- (iv) *There are two edges among three leaves from three despotic/small units.*

<sup>2</sup> Proofs of propositions marked with \* are omitted due to space limit.

In Exchange Rule 2, (i) and (ii) deal with edges incident to a democratic unit, while (iii) and (iv) deal with edges between peripheral vertices. The remaining edges between different units are characterized by the following proposition.

**Proposition 2 (\*)**. *Let  $\mathcal{U}$  be a unit partition on which none of Exchange Rule 1 and 2 is applicable, and let  $u_1u_2$  be an edge between two different units  $U_1$  and  $U_2$ . If neither  $u_1$  nor  $u_2$  is a core vertex, then for both  $i = 1, 2$ , the vertex  $u_i$  is either a leaf or the nose of a bull-unit.*

The following rule deals with edges between leaves or noses of bull-units. It consolidates leaves to make twigs, or transforms bull-units to net-units.

**Exchange Rule 3.** (i) *If there is an edge between a nose  $u_1$  of a bull-unit  $U_1$  and a leaf  $u_2$  of a despotic/small unit  $U_2$ , move  $u_2$  from  $U_2$  to  $U_1$ .*

(ii) *For an edge  $u_1u_2$  between two leaves from two units  $U_1$  and  $U_2$ , if  $U_1$  is a (1,4)-unit, move  $u_1$  from  $U_1$  to  $U_2$ ; and if neither  $U_1$  nor  $U_2$  is a (1,4)-unit and  $|U_1| \geq |U_2|$ , move  $u_1$  from  $U_1$  to  $U_2$ .<sup>3</sup>*

Remark that Exchange Rule 3(ii) creates *insupportable units*, i.e., units not represented by graphs in Figs. 1–3, only if  $u_1$  is added to a (1,4)-unit  $U_2$ , i.e., there is an edge between two leaves from two (1,4)-units. We will prove that this does not happen in our algorithm.

**Proposition 3 (\*)**. *On a unit partition  $\mathcal{U}$  containing only units represented by graphs in Figs. 1–3, if no edge exists between two leaves of two (1,4)-units, then after applications of Exchange Rule 3(ii), the new unit partition still contains only units represented by graphs in Figs. 1–3.*

We conclude the preparation phase by introducing reduced unit partitions. A *reduced unit partition* is a unit partition that consists of units represented by graphs in Figs. 1–3, on which none of above rules (Reduction Rule 1 and Exchange Rule 1–3) are applicable. The following lemma summarizes the properties of a reduced unit partition.

**Lemma 4 (\*)**. *For a reduced unit partition, the following properties hold.*

- (i) *In the subgraph induced on all peripheral vertices, each vertex in a twig has degree one, and each leaf has degree zero.*
- (ii) *The neighborhood of the set of peripheral vertices consists of core vertices.*
- (iii) *Each net-unit is adjacent to some core vertices, and only core vertices.*

## 4 Main Rules

We are ready to present our main exchange rules on the reduced unit partition. They move twigs and leaves among units. If neither a larger packing nor a

<sup>3</sup> Actually, when neither  $U_1$  nor  $U_2$  is a (1,4)-unit, we can move vertices in an arbitrary direction. We choose this way to simplify our proof.

reducible structure emerges, then we are able to eliminate all the units with more than six vertices, and bound the number of units on six vertices.

We start with those units  $U$  with  $d_1(U) \geq 2$ , and the idea is to cut a twig from them and graft it to a small unit. We build an auxiliary bipartite graph  $B_2 = (L \cup R; E)$  as follows:

- for each core vertex, introduce a node into  $L$ ;
- for each twig, introduce a node into  $R$ ;
- add an edge between a node  $x \in L$  and a node  $y \in R$  if the core vertex represented by  $x$  is adjacent to the twig represented by  $y$ .

For a node  $x \in L$ , we use the core vertex of  $x$  to refer to the core vertex represented by  $x$ . For a node  $y \in R$ , the twig of  $y$  refers to the twig represented by  $y$ . We say that  $y_1x_2y_2 \cdots x_\ell$  is a twig-alternating path if (1)  $x_i \in L$  and  $y_j \in R$  for  $2 \leq i \leq \ell$  and  $1 \leq j \leq \ell - 1$ ; and (2)  $x_i$  and  $y_j$  are from the same unit if and only if  $i = j$ . Note that we don't have  $x_1$  and  $y_\ell$ .

**Exchange Rule 4.** *On a reduced unit partition, if there is a twig-alternating path  $y_1x_2y_2 \cdots x_\ell$  where*

- (i) *the unit  $U_1$  containing the twig of  $y_1$  is a  $(2, 2)$ -unit or has  $d_1(U_1) > 2$ ; and*
- (ii) *the unit  $U_2$  containing the core vertex of  $x_\ell$  has  $d_1(U_2) = 0$ ,*

*then for  $i = 1, \dots, \ell - 1$ , move the twig of  $y_i$  to the unit containing the core vertex of  $x_{i+1}$  (Fig. 4).*



**Fig. 4.** An illustration for Exchange Rule 4. After a chain of operations, a twig is “moved” from the first unit to the last one.

**Proposition 4 (\*).** *After applying Exchange Rule 4 for a twig-alternating path  $P = y_1x_2y_2 \cdots x_\ell$ , the following hold.*

- (i) *The unit  $U_1$  containing the twig of  $y_1$  becomes a  $(d_1(U_1) - 1, d_2(U_1))$ -unit.*
- (ii) *The unit containing the twig of  $y_i$  retains its type for  $2 \leq i \leq \ell - 1$ .*
- (iii) *If the unit  $U_2$  containing the core vertex of  $x_\ell$  has  $d_2(U_2) \geq 2$ , then it becomes a  $(d_1(U_1) + 1, d_2(U_1))$ -unit. Otherwise,  $U_2$  is a  $(0, 1)$ - or  $(0, 0)$ -unit and becomes a unit with two vertex-disjoint  $P_2$ 's, or a democratic/despotic unit with six or five vertices.*

Remark that an unreduced unit partition contains no  $(1, 4)$ -units, and Exchange Rule 4 will be the only rule that introduces  $(1, 4)$ -units. If Exchange Rule 4 is not applicable, twig-alternating paths ensure a reducible set.

**Lemma 5.** *On a reduced unit partition, if there exists a unit  $U$  with  $d_1(U) = d_2(U) = 2$  or  $d_1(U) > 2$ , but Exchange Rule 4 is not applicable, then we can find a reducible set in polynomial time.*

*Proof.* Let  $\mathcal{T}$  be the twigs whose representing nodes can be reached by twig-alternating paths from the representing nodes of twigs of  $U$ . Let  $C$  be the set of vertices in the twigs of  $\mathcal{T}$ , and  $A = N(C)$ . By Lemma 4,  $G[C]$  consists of edge components and  $A$  consists of core vertices. Since Exchange Rule 4 is not applicable, for each core vertex in  $A$ , its unit has at least one twig in  $\mathcal{T}$ , i.e., there is a distinct twig attached to it. Hence  $C$  is a reducible set.  $\square$

We next migrate leaves in a similar way as Exchange Rule 4. We build an auxiliary bipartite graph  $B_3 = (L \cup R; E)$  as follows:

- for each core vertex, introduce a node into  $L$ ;
- for each leaf, introduce a node into  $R$ ;
- add an edge between a node  $x \in L$  and a node  $y \in R$  if the core vertex represented by  $x$  is adjacent to the leaf represented by  $y$ .

Again, for a node in  $B_3$ , we use the vertex of it as a shorthand for the vertex represented by it. We say that  $y_1x_2y_2 \cdots x_\ell$  is a leaf-alternating path if (1)  $x_i \in L$  and  $y_j \in R$  for  $2 \leq i \leq \ell$  and  $1 \leq j \leq \ell - 1$ ; and (2)  $x_i$  and  $y_j$  are from the same unit if and only if  $i = j$ .

**Exchange Rule 5.** *On a reduced unit partition without  $(4, 0)$ -,  $(3, 1)$ -, or  $(3, 0)$ -units, if there is a leaf-alternating path  $y_1x_2y_2 \cdots x_\ell$  where*

- (i) *the unit  $U_1$  containing the leaf of  $y_1$  is not a  $(0, 3)$ -unit and has  $d_2(U_1) \geq 3$ ;*  
and
- (ii) *the unit  $U_2$  containing the core vertex of  $x_\ell$  is a unit with  $d_2(U_2) \leq 1$ .*

*then for  $i = 1, \dots, \ell - 1$ , move the leaf of  $y_i$  to the unit containing the core vertex of  $x_{i+1}$ .*

**Proposition 5 (\*).** *After applying Exchange Rule 5 for a leaf-alternating path  $P = y_1x_2y_2 \cdots x_\ell$ , the following hold.*

- (i) *The unit  $U_1$  containing the leaf of  $y_1$  becomes a  $(d_1(U_1), d_2(U_1) - 1)$ -unit.*
- (ii) *The unit containing the leaf of  $y_i$  retains its type for  $2 \leq i \leq \ell - 1$ .*
- (iii) *If the unit  $U_2$  containing the core vertex of  $x_\ell$  has  $d_2(U_2) \geq 2$ , then it becomes a  $(d_1(U_1), d_2(U_2) + 1)$ -unit. Otherwise,  $U_2$  is a  $(0, 1)$ - or  $(0, 0)$ -unit and it becomes a small unit with four vertices or a democratic/despotic unit with five vertices.*

A lemma similar to Lemma 5 holds for Exchange Rule 5.

**Lemma 6 (\*).** *On a reduced unit partition without  $(4, 0)$ -,  $(3, 1)$ -, or  $(3, 0)$ -units, if there exists a unit  $U$  (not  $(0, 3)$ -unit) with  $d_2(U) \geq 3$ , but Exchange Rule 5 is not applicable, then we can find a reducible set in polynomial time.*



After an application of Exchange Rule 4 or 5, core vertices of a  $(0, 1)$ -unit or  $(0, 0)$ -unit may become peripheral, and hence may turn a reduced unit partition into unreduced. Therefore we may need to reapply Exchange Rules 1–3. Precisely we use Exchange Rule 4 to eliminate  $(4, 0)$ -,  $(3, 1)$ -,  $(3, 0)$ - and  $(2, 2)$ -units; and when Exchange Rule 4 is not applicable, we use Exchange Rule 5 to eliminate  $(1, 4)$ -,  $(1, 3)$ - and  $(0, 4)$ -units. Although Exchange Rule 5 may turn a  $(2, 1)$ -unit into a  $(2, 2)$ -unit that triggers Exchange Rule 4, we will show in the next section Exchange Rule 4–5 can be applied at most  $O(k)$  times.

At this point,  $(2, 1)$ -,  $(2, 0)$ -, and  $(1, 2)$ -units are the only despotic units. Only net-units and  $(2, 1)$ -units have more than five vertices, and it remains to bound the number of them by the number of small units. For democratic units, we use  $s_{\text{net}}$ ,  $s_{\text{bull}}$ ,  $s_{\text{pendant}}$ , and  $s_{C_5}$  to denote the number of, respectively, net-units, bull-units, pendant-units, and  $C_5$ -units. For other units, we use  $s_{a,b}$  to denote the number of  $(a, b)$ -units.

We construct an auxiliary bipartite graph  $B_4$  as follows:

- for each net-unit and each twig, add a node into  $L$ ;
- for each vertex not in any net-unit or twig, add a node into  $R$ ;
- for two nodes  $x \in L$  and  $y \in R$ , add an edge  $xy$  if the vertex represented by  $y$  is adjacent to the net-unit or twig represented by  $x$  in  $G$ .

Here we regard a net-unit as a unit with a removable twig. Note that by Lemma 4, there is no isolated node in  $L$ , and each node in  $N(L)$  represents a core vertex.

**Exchange Rule 6.** *On a reduced unit partition where  $(2, 1)$ -,  $(2, 0)$ -, and  $(1, 2)$ -units are the only despotic units, if  $s_{\text{net}} + s_{2,1} + s_{2,0} > s_{0,3} + s_{0,1} + s_{0,0}$  and there is a matching  $M$  of  $B_4$  that saturates  $L$ , then find a larger  $P_2$ -packing than  $\mathcal{P}$  as following.*

- (i) *For each net-unit  $U$ , find two  $P_2$ 's in  $G[U \cup \{v\}]$ , where  $v$  is the vertex whose representing node is matched to  $U$  by  $M$ . Delete  $P_2$ 's involving vertices of  $U \cup \{v\}$ , and add these two new  $P_2$ 's.*
- (ii) *Each twig  $T$  makes a  $P_2$  together with the vertex  $v$  whose representing node is matched to  $T$  by  $M$ . Delete  $P_2$ 's involving vertices in  $V(T) \cup \{v\}$ , and add the new  $P_2$ .*

**Lemma 7.** *Exchange Rule 6 is correct.*

*Proof.* For each net-,  $(2, 1)$ - or  $(2, 0)$ -unit, we find two new  $P_2$ 's. For each  $(1, 2)$ -unit, we find one new  $P_2$ . The number of new  $P_2$ 's are  $2(s_{\text{net}} + s_{2,1} + s_{2,0}) + s_{1,2}$ . Since nodes in  $N(L)$  represent core vertices, no defining  $P_2$ 's of pendant-,  $C_5$ - or bull-units are deleted. Hence the number of  $P_2$ 's in the new  $P_2$ -packing is at least

$$2(s_{\text{net}} + s_{2,1} + s_{2,0}) + s_{1,2} + s_{\text{pendant}} + s_{C_5} + s_{\text{bull}} >$$

$$(s_{\text{net}} + s_{2,1} + s_{2,0}) + (s_{0,3} + s_{0,1} + s_{0,0}) + s_{1,2} + s_{\text{pendant}} + s_{C_5} + s_{\text{bull}},$$

which is the size of  $\mathcal{P}$  as  $(2, 1)$ -,  $(2, 0)$ -, and  $(1, 2)$ -units are the only despotic units. □

If no matching saturating  $L$ , then we can find in polynomial time, by Lemma 2, a subset  $L' \subseteq L$  and a matching of  $B_4$  between  $N(L')$  and  $L'$  that saturates  $N(L')$ . Now we apply our second non-trivial reduction rule, where the “reducible set” contains net-units and edge components. Each net-unit contributes a  $P_2$  and an extra edge.

**Reduction Rule 3.** For a node set  $L' \subseteq L$  of  $B_4$ , let  $s'_{\text{net}}$  be the number of net-units represented by  $L'$ , and  $X$  be the set of vertices in the net-units or twigs represented by  $L'$ . If there exists a matching of  $B_4$  between  $N(L')$  and  $L'$  that saturates  $N(L')$ , then remove  $X \cup N(X)$  and decrease  $k$  by  $s'_{\text{net}} + |N(X)|$ .

**Lemma 8.** Reduction rule 3 is safe:  $\text{opt}(G) = \text{opt}(G - (X \cup N(X))) + s'_{\text{net}} + |N(X)|$ .

*Proof.* Let  $G' = G - (X \cup N(X))$ . Note that  $N(X)$  is precisely the set of vertices represented by  $N(L')$ . Let  $v$  be a vertex in  $N(X)$ . If its representing node is matched to a twig, then  $v$  and the twig together make a  $P_2$ . Otherwise, it is matched to a net-unit  $U$ ; i.e.,  $U$  is adjacent to  $v$ . We can find two adjacent vertices in  $U$  such that they together with  $v$  make another  $P_2$ , and their removal from  $U$  leaves a  $P_2$ . Thus the number of  $P_2$ 's we can find in  $G[X \cup N(X)]$  is  $|N(X)| + s'_{\text{net}}$ . Therefore,  $\text{opt}(G) \geq \text{opt}(G - (X \cup N(X))) + s'_{\text{net}} + |N(X)|$ .

Any  $P_2$ -packing of  $G$  contains at most  $|N(X)|$  vertex-disjoint  $P_2$ 's involving vertices in  $N(X)$ , hence  $\text{opt}(G) \leq \text{opt}(G - N(X)) + |N(X)|$ . And each component in the subgraph of  $G - N(X)$  induced on  $X$ , is either a net-unit or a twig represented by a node in  $L'$ . Thus  $\text{opt}(G - N(X)) = \text{opt}(G - (X \cup N(X))) + s'_{\text{net}}$ , and  $\text{opt}(G) \leq \text{opt}(G - (X \cup N(X))) + s'_{\text{net}} + |N(X)|$ .  $\square$

**Corollary 1.** On a reduced unit partition where  $(2, 1)$ -,  $(2, 0)$ -, and  $(1, 2)$ -units are the only despotic units, if  $s_{\text{net}} + s_{2,1} + s_{2,0} > s_{0,3} + s_{0,1} + s_{0,0}$ , then at least one of Exchange Rule 6 and Reduction Rule 3 is applicable.

## 5 Kernelization Algorithm

We are now ready to summarize the kernelization algorithm and prove Theorem 1. Throughout our algorithm (see Fig. 5), we maintain a reduced unit partition by Reduction Rule 1 and Exchange Rule 1–3, and restart if a reducible structure or a larger  $P_2$ -packing is found. For units with more than five vertices, we apply Exchange Rule 4–5 to eliminate those with  $d_1(U) \geq 3$  or  $d_1(U) + d_2(U) \geq 4$ . Although some six-vertex units cannot be eliminated, their number is upper bounded by the number of small units after applying Exchange Rule 6 and Reduction Rule 3 exhaustively. Putting all these together, we obtain the kernel.

To prove Theorem 1, the first two steps are to show (1) the correctness of our algorithm, and (2) the polynomial running time of our algorithm. The main difficulty of proving these two lemmas is that applications of Exchange Rule 4–5 may introduce edges between peripheral vertices which triggers Exchange Rule 3 (ii) and creates new units with  $d_1(U) \geq 3$  or  $d_1(U) + d_2(U) \geq 4$ . At first glance,

INPUT: a graph  $G$  and an integer  $k$ .  
 OUTPUT: a graph  $G'$  with  $|V(G')| \leq 5k$  and an integer  $k'$ .

1. **if**  $k \leq 0$  **then return** a trivial yes-instance.
2. **if**  $|V(G)| \leq 5k$  **then return**  $(G, k)$ .
3. Apply Reduction Rule 1 exhaustively.
4. Set  $\mathcal{P}$  to be an arbitrary maximal  $P_2$ -packing.
5. **if**  $|\mathcal{P}| \geq k$  **then return** a trivial yes-instance.
6. **if** (Lemma 3) **then**  
     apply Reduction Rule 2; **goto** 1.
7. **else** set  $\mathcal{U}$  to be an unreduced unit partition.
8. **if** Exchange Rule 1 or 2 is applicable **then**  
     apply it; **goto** 5.
9. **if** Exchange Rule 3 is applicable **then**  
     apply it; **goto** 8.
10. **if** there is a (4, 0)-, (3, 1)-, (3, 0)- or (2, 2)-unit **then**  
     **if** Exchange Rule 4 is applicable **then**  
         apply it; **goto** 8.  
     **else** apply Reduction Rule 2 (Lemma 5); **goto** 1;
11. **if** there is a (1, 4)-, (1, 3)- or (0, 4)-unit **then**  
     **if** Exchange Rule 5 is applicable, **then**  
         apply it; **goto** 8.  
     **else** apply Reduction Rule 2 (Lemma 6); **goto** 1.
12. **if** the number of net-, (2, 1)- and (2, 0)-units is larger than  
     the number of small units **then**  
     **if** Exchange Rule 6 is applicable **then**  
         apply it; **goto** 5.  
     **else** apply Reduction Rule 3 (Corollary 1); **goto** 1.
13. **return** a trivial no-instance.

**Fig. 5.** A summary of our algorithm. A trivial yes-instance can be an empty graph and  $k = 0$ ; while a trivial no-instance can be an empty graph and  $k = 1$ . Note that we execute step 11 only when step 10 has been applied exhaustively and execute step 12 only when step 10 and 11 have been applied exhaustively.

this may (1) create insupportable units, and (2) cause deadlock. By careful study, we will show in the following that both cases will not happen.

**Lemma 9 (\*)**. *The algorithm is correct and terminates in polynomial time.*

By Lemma 9, our algorithm is a valid kernelization algorithm. Now Theorem 1 follows by counting numbers of different units.

*Proof (Proof of Theorem 1)*. Our algorithm returns a reduced unit partition with only democratic units, and units satisfying  $d_1(U) \leq 2$  and  $d_1(U) + d_2(U) \leq 3$ . Moreover, we have  $s_{\text{net}} + s_{2,1} + s_{2,0} \leq s_{0,3} + s_{0,1} + s_{0,0}$ . The number of vertices in the resulting graph is

$$\begin{aligned}
& 6s_{\text{net}} + 5(s_{\text{pendant}} + s_{C_5} + s_{\text{bull}}) + 6s_{2,1} + 5(s_{2,0} + s_{1,2}) + 4(s_{0,3} + s_{0,1}) + 3s_{0,0} \\
& \leq 5(s_{\text{net}} + s_{2,1} + s_{\text{pendant}} + s_{C_5} + s_{\text{bull}} + s_{2,0} + s_{1,2}) + (s_{\text{net}} + s_{2,1}) + 4(s_{0,3} + s_{0,1} + s_{0,0}) \\
& \leq 5(s_{\text{net}} + s_{2,1} + s_{\text{pendant}} + s_{C_5} + s_{\text{bull}} + s_{2,0} + s_{1,2}) + 5(s_{0,3} + s_{0,1} + s_{0,0}) \\
& \leq 5(s_{\text{net}} + s_{2,1} + s_{\text{pendant}} + s_{C_5} + s_{\text{bull}} + s_{2,0} + s_{1,2} + s_{0,3} + s_{0,1} + s_{0,0}) \\
& < 5k,
\end{aligned}$$




which proves the main theorem.  $\square$

## References

1. Chang, M.-S., Chen, L.-H., Hung, L.-J.: A  $5k$  kernel for  $P_2$ -packing in net-free graphs. In: ICSEC 2014, pp. 12–17 (2014)
2. Chen, J., Fernau, H., Shaw, P., Wang, J., Yang, Z.: Kernels for packing and covering problems. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) AAIM/FAW -2012. LNCS, vol. 7285, pp. 199–211. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29700-7\\_19](https://doi.org/10.1007/978-3-642-29700-7_19)
3. Fellows, M.R.: Blow-ups, win/win's, and crown rules: some new directions in *FPT*. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 1–12. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-39890-5\\_1](https://doi.org/10.1007/978-3-540-39890-5_1)
4. Hall, P.: On representatives of subsets. *J. Lond. Math. Soc.* **10**, 26–30 (1935)
5. Hopcroft, J.E., Karp, R.M.: An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* **2**(4), 225–231 (1973)
6. Hurkens, C.A.J., Schrijver, A.: On the size of systems of sets every  $t$  of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discret. Math.* **2**(1), 68–72 (1989)
7. Kaneko, A., Kelmans, A.K., Nishimura, T.: On packing 3-vertex paths in a graph. *J. Graph Theory* **36**(4), 175–197 (2001)
8. Kann, V.: Maximum bounded  $H$ -matching is MAX SNP-complete. *Inf. Process. Lett.* **49**(6), 309–318 (1994)
9. Kelmans, A.K.: Packing 3-vertex paths in claw-free graphs and related topics. *Discret. Appl. Math.* **159**(2–3), 112–127 (2011)
10. Kelmans, A.K., Mubayi, D.: How many disjoint 2-edge paths must a cubic graph have? *J. Graph Theory* **45**(1), 57–79 (2004)
11. Kirkpatrick, D.G., Hell, P.: On the complexity of general graph factor problems. *SIAM J. Comput.* **12**(3), 601–609 (1983)
12. Monnot, J., Toulouse, S.: The path partition problem and related problems in bipartite graphs. *Oper. Res. Lett.* **35**(5), 677–684 (2007)
13. Prieto, E., Sloper, C.: Looking at the stars. *Theor. Comput. Sci.* **351**(3), 437–445 (2006)
14. Wang, J., Ning, D., Feng, Q., Chen, J.: An improved kernelization for  $P_2$ -packing. *Inf. Process. Lett.* **110**(5), 188–192 (2010)
15. Xiao, M., Kou, S.: Kernelization and parameterized algorithms for 3-path vertex cover. In: Gopal, T.V., Jäger, G., Steila, S. (eds.) TAMC 2017. LNCS, vol. 10185, pp. 654–668. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-55911-7\\_47](https://doi.org/10.1007/978-3-319-55911-7_47)



# Classical Complexity and Fixed-Parameter Tractability of Simultaneous Consecutive Ones Submatrix & Editing Problems

M. R. Rani<sup>(✉)</sup> , Mohith Jagalmohanam , and R. Subashini<sup>(✉)</sup> 

Department of Computer Science and Engineering,  
National Institute of Technology, Calicut, India  
{rani\_p150067cs,mohith\_b140145cs,suba}@nitc.ac.in

**Abstract.** A binary matrix  $M$  has the consecutive ones property ( $C1P$ ) for rows (resp. columns) if there is a permutation of its columns (resp. rows) that arranges the ones consecutively in all the rows (resp. columns). If  $M$  has the  $C1P$  for rows and the  $C1P$  for columns, then  $M$  is said to have the simultaneous consecutive ones property ( $SC1P$ ). We focus on the classical complexity and fixed parameter tractability of Simultaneous Consecutive Ones Submatrix ( $SC1S$ ) and Simultaneous Consecutive Ones Editing ( $SC1E$ ) [1] problems here.  $SC1S$  problems focus on deleting a minimum number of rows, columns and rows as well as columns to establish the  $SC1P$  whereas  $SC1E$  problems deal with flipping a minimum number of 1-entries, 0-entries and 0-entries as well as 1-entries to obtain the  $SC1P$ . We show that the decision versions of the  $SC1S$  and  $SC1E$  problems are NP-complete. We consider the parameterized versions of the  $SC1S$  and  $SC1E$  problems with  $d$ , being the solution size, as the parameter and are defined as follows. Given a binary matrix  $M$  and a positive integer  $d$ ,  $d$ - $SC1S$ - $R$  ( $d$ - $SC1S$ - $C$ ) problem decides whether there exists a set of rows (columns) of size at most  $d$  whose deletion results in a matrix with the  $SC1P$ . The  $d$ - $SC1S$ - $RC$  problem decides whether there exists a set of rows as well as columns of size at most  $d$  whose deletion results in a matrix with the  $SC1P$ . The  $d$ - $SC1P$ - $0E$  ( $d$ - $SC1P$ - $1E$ ) problem decides whether there exists a set of 0-entries (1-entries) of size at most  $d$  whose flipping results in a matrix with the  $SC1P$ . The  $d$ - $SC1P$ - $01E$  problem decides whether there exists a set of 0-entries as well as 1-entries of size at most  $d$  whose flipping results in a matrix with the  $SC1P$ . Using a related result from the literature [2], we show that  $d$ - $SC1P$ - $0E$  on general binary matrices is fixed-parameter tractable with a run time of  $O^*(45.5625^d)$ . We also give FPT algorithms for  $SC1S$  and  $SC1E$  problems on certain restricted binary matrices.

**Keywords:** Simultaneous consecutive ones property  
Parameterized complexity

## 1 Introduction

Binary matrices having simultaneous consecutive ones property are fundamental in recognizing biconvex graphs [3], recognizing proper interval graphs [4], identifying blocks of matrices (in applications arising from integer linear programming) [5] and finding clusters from metabolic networks [6]. A binary matrix has the *consecutive ones property (C1P) for rows (resp. columns)* [7], if there is a permutation of its columns (resp. rows) that arranges the ones consecutively in all the rows (resp. columns). A binary matrix has the *simultaneous consecutive ones property (SC1P)* [1], if we can permute the rows and columns in such a way that the ones in every column and in every row occur consecutively. That is, a binary matrix has the *SC1P* if it satisfies the *C1P* for both rows and columns. Matrices with the *C1P* and the *SC1P* are related to interval graphs and proper interval graphs respectively. They can be recognized efficiently and have forbidden submatrix characterizations. There exist several linear-time and polynomial-time algorithms for testing the *C1P* for columns [8–13]. These algorithms can also be used for testing the *C1P* for rows. The column permutation (if one exists) to obtain the *C1P* for rows will not affect the consecutive ones property of the columns (if one exists) and vice versa. Thus, testing the *SC1P* can also be done in linear time.

Not all binary matrices have the *SC1P*. We consider the Simultaneous Consecutive Ones Submatrix (*SC1S*) and Simultaneous Consecutive Ones Editing (*SC1E*) [1] problems to deal with matrices that do not have the *SC1P*. *SC1S* problems focus on deleting a minimum number of rows (columns) and rows as well as columns to establish the *SC1P* whereas *SC1E* problems deal with flipping a minimum number of 1-entries (0-entries) and 1-entries as well as 0-entries to obtain the *SC1P*. We pose the following optimization problems: *SC1S*-row deletion, *SC1S*-column deletion and *SC1S*-row-column deletion in the *SC1S* category, and, *SC1P*-1-Flipping, *SC1P*-0-Flipping and *SC1P*-01-Flipping in the *SC1E* category. Given a binary matrix  $M$ , the *SC1S* row, column, and row-column deletion finds a minimum number of rows, columns, and rows as well as columns respectively, which on deletion results in a matrix satisfying the *SC1P*. On the other hand, the *SC1P* 1-flipping, 0-flipping and 01-flipping finds a minimum number of entries containing ones, zeroes, and both zeroes and ones respectively, to be flipped to satisfy the *SC1P*. We show that the decision versions of the above defined problems are NP-complete. We refer to the parameterized versions of the above problems parameterized by  $d$  as *d-SC1S-R*, *d-SC1S-C*, *d-SC1S-RC*, *d-SC1P-1E*, *d-SC1P-0E* and *d-SC1P-01E* respectively, with  $d$  being the number of rows that can be deleted, columns that can be deleted, rows as well as columns that can be deleted, 1-entries that can be flipped, 0-entries that can be flipped, and 0-entries as well as 1-entries that can be flipped respectively.

**Parameterized Complexity:** *Fixed-parameter tractability* is one of the ways to deal with NP-hard problems. In parameterized complexity, the run-time of an algorithm is measured not only in terms of the input size, but also in terms of a parameter. A parameter is an integer associated with an instance of a problem. It is a measure of some property of the input instance. A problem is

fixed-parameter tractable with respect to parameter  $d$ , if there exists an algorithm that solves the problem in  $f(d).n^{O(1)}$  time, where  $f$  is a computable function depending only on  $d$  and  $n$  is the size of the input instance. The time complexity of such algorithms can be expressed as  $O^*(f(d))$ . We recommend the interested reader to [14] for a more comprehensive overview of the topic.

A *matrix* can be considered as a set of rows (columns) together with an order on this set [15]. Throughout this paper, the term matrix refers to a *binary matrix*, and for a given matrix  $M$ ,  $m_{ij}$  refers to the entry corresponding to  $i^{th}$  row and  $j^{th}$  column of the matrix. Matrix having at most  $x$  ones in each column and at most  $y$  ones in each row is denoted as  $(x, y)$ -matrix. A  $(2, *)$ -matrix can contain at most two ones per column and there is no bound on the number of ones per row. A  $(*, 2)$ -matrix has no restriction on the number of ones per column and have at most two ones per row. Given an  $m \times n$  matrix  $M$ , let  $R(M) = \{r_1, r_2, \dots, r_m\}$  and  $C(M) = \{c_1, c_2, \dots, c_n\}$  denote the sets of rows and columns, respectively. Here,  $r_i$  and  $c_j$  denote the binary vectors corresponding to row  $r_i$  and column  $c_j$  of  $M$ , respectively. For a subset  $R' \subseteq R(M)$  of rows,  $M[R']$  and  $M \setminus R'$  denote the submatrix induced on  $R'$  and  $R(M) \setminus R'$  respectively. Similarly, for a subset  $C' \subseteq C(M)$  of columns, the submatrix induced on  $C'$  and  $C(M) \setminus C'$  are denoted by  $M[C']$  and  $M \setminus C'$  respectively. Let  $A = \{ij \mid m_{ij} = 1\}$  and  $B = \{ij \mid m_{ij} = 0\}$  be the set of indices of all 1-entries and 0-entries respectively in  $M$ . For the sake of completeness, the formal definitions of the problems  $d$ -SC1S-R,  $d$ -SC1S-C,  $d$ -SC1S-RC,  $d$ -SC1P-0E and  $d$ -SC1P-01E are given as follows.

**Simultaneous Consecutive Ones Submatrix Problems**

**Instance:**  $\langle M, d \rangle$ - An  $m \times n$  matrix  $M$  and an integer  $d \geq 0$ .

**Parameter:**  $d$ .

**$d$ -SC1S-R:** Does there exist a set of rows  $R' \subseteq R(M)$ , such that  $|R'| \leq d$  and  $M \setminus R'$  satisfies the SC1P?

**$d$ -SC1S-C:** Does there exist a set of columns  $C' \subseteq C(M)$ , such that  $|C'| \leq d$  and  $M \setminus C'$  satisfies the SC1P?

**$d$ -SC1S-RC:** Does there exist a set of rows  $R' \subseteq R(M)$ , and a set of columns  $C' \subseteq C(M)$ , with  $|R'| + |C'| \leq d$  such that  $((M \setminus R') \setminus C')$  satisfies the SC1P?

**Simultaneous Consecutive Ones Editing Problems**

**Instance:**  $\langle M, d \rangle$ - An  $m \times n$  matrix  $M$  and an integer  $d \geq 0$ .

**Parameter:**  $d$ .

**$d$ -SC1P-1E:** Does there exist a set  $A' \subseteq A$ , with  $|A'| \leq d$  such that the resultant matrix obtained by flipping the entries of  $A'$  in  $M$  satisfies the SC1P?

**$d$ -SC1P-0E:** Does there exist a set  $B' \subseteq B$ , with  $|B'| \leq d$  such that the resultant matrix obtained by flipping the entries of  $B'$  in  $M$  satisfies the SC1P?

**$d$ -SC1P-01E:** Does there exist a set  $I \subseteq A \cup B$ , with  $|I| \leq d$  such that the resultant matrix obtained by flipping the entries of  $I$  in  $M$  satisfies the SC1P?

**Complexity Status:** Oswald and Reinelt [1] posed the  $d$ - $SC1P$ - $1E$  problem as  $k$ -augmented simultaneous consecutive ones property and showed that it is NP-complete. To the best of our knowledge, the parameterized problems posed under the  $SC1S$  and  $SC1E$  category except  $d$ - $SC1P$ - $1E$  are not explicitly mentioned in the literature. Also the parameterized complexity of  $SC1S$  and  $SC1E$  problems are not known prior to this work.

**Our Results:** We investigate the classical complexity and the fixed-parameter tractability of  $SC1S$  and  $SC1E$  problems (defined above). We show the NP-completeness result for all the  $SC1S$  and  $SC1E$  problems except  $d$ - $SC1P$ - $1E$ , which was shown to be NP-complete in [1]. Using a related result from the literature [2], we present a fixed-parameter tractable algorithm for  $d$ - $SC1P$ - $0E$  problem on general binary matrices (where there is no restriction on the number of ones in rows and columns) with a run-time of  $O^*(45.5625^d)$ . We also obtain an improved run-time of  $O^*(6^d)$  for  $d$ - $SC1P$ - $0E$  on  $(2, *)$  and  $(*, 2)$  matrices.

For  $(2, 2)$ -matrices, we show that  $SC1S$  and the  $SC1E$  problems are solvable in polynomial-time. For other problems, the FPT results obtained on restricted matrices are listed in the following table:

Problem	$(2, *)$ -matrix	$(*, 2)$ -matrix
$d$ - $SC1S$ - $R/C/RC$	$O^*(4^d/3^d/7^d)$	$O^*(3^d/4^d/7^d)$
$d$ - $SC1P$ - $1E/0E/01E$	$O^*(6^d/6^d/12^d)$	$O^*(6^d/6^d/12^d)$

The fixed-parameter tractable algorithm for  $d$ - $SC1S$ - $R$  on  $(2, *)$ -matrices can be used to show that proper interval vertex deletion [16] on *triangle-free* graphs is FPT (using Lemma 7) with a run-time of  $O^*(4^d)$ , where  $d$  denotes the number of possible vertex deletions. We also observe that the  $SC1S$  and the  $SC1E$  problems admit constant factor polynomial-time approximation algorithms on  $(2, *)$ -matrices and  $(*, 2)$ -matrices.

**Techniques Used:** Our results rely on the following forbidden submatrix characterization of the  $SC1P$  (see Fig. 1) by Tucker [3].

**Theorem 1** ([3, Theorem 11]). *A matrix  $M$  has the  $SC1P$  if and only if no submatrix of  $M$ , or of the transpose of  $M$ , is a member of the configuration (see Sect. 2) of  $M_{I_k}$  ( $k \geq 1$ ),  $M_{2_1}$ ,  $M_{2_2}$ ,  $M_{3_1}$ ,  $M_{3_2}$  and  $M_{3_3}$ .*

That is, a matrix  $M$  has the  $SC1P$  if and only if no submatrix of  $M$  is a member of the configuration of  $M_{I_k}$  ( $k \geq 1$ ),  $M_{2_1}$ ,  $M_{2_2}$ ,  $M_{3_1}$ ,  $M_{3_2}$ ,  $M_{3_3}$  or their transposes. For ease of reference, we refer to the fixed-size forbidden matrices in the forbidden submatrix characterization of  $SC1P$  as  $X$ . i.e.

$$X = \{M_{2_1}, M_{2_2}, M_{3_1}, M_{3_2}, M_{3_3}, M_{2_1}^T, M_{2_2}^T, M_{3_1}^T, M_{3_2}^T, M_{3_3}^T\}.$$

We used the following results to search and destroy the forbidden matrices of  $X$  and  $M_{I_k}/M_{I_k}^T$  (where  $k \geq 1$ ) from  $M$ . Lemmas 1 and 2 state the run-time to find a forbidden matrix of  $X$  and  $M_{I_k}/M_{I_k}^T$  in  $M$ . Lemma 1 is obtained



from ([15, Proposition 3.2]), by considering the maximum possible size of the forbidden submatrix in  $X$  as  $6 \times 5$  (shown in Fig. 1).

**Lemma 1.** *Let  $M$  be a matrix of size  $m \times n$ . Then a minimum size submatrix in  $M$  that is isomorphic to one of the forbidden matrices of  $X$  can be found in  $O(m^6n)$  time.*

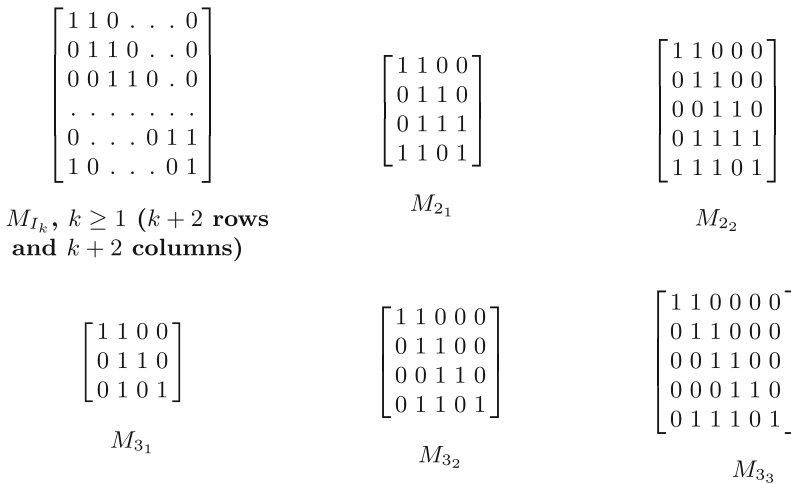
We obtain the following lemma from ([15, Proposition 3.4]), by considering the maximum number of ones in each row of  $M$  as  $n$ .

**Lemma 2.** *Let  $M$  be a matrix of size  $m \times n$ . Then a minimum size submatrix of type  $M_{I_k}$  or  $M_{I_k}^T$  ( $k \geq 1$ ) in  $M$  can be found in  $O(n^3m^3)$  time.*

To destroy  $M_{I_k}/M_{I_k}^T$  (where  $k \geq 1$ ), we consider the representing graph  $G_{M_{I_k}}/G_{M_{I_k}^T}$  (Definition 2). Following result shows that  $G_{M_{I_k}}/G_{M_{I_k}^T}$  is a chordless cycle.

**Lemma 3** ([15, Observation 3.1]). *The representing graph of  $M_{I_k}/M_{I_k}^T$ , i.e.,  $(G_{M_{I_k}}/G_{M_{I_k}^T})$ , is a chordless cycle of length  $2k + 4$ .*

It is clear from Lemma 3, that searching for both  $M_{I_k}$  and its transpose is equivalent to searching for  $M_{I_k}$  alone.



**Fig. 1.** A subset of the forbidden submatrices for the  $SC1P$  [3].

**Organization of the paper:** In Sect. 2, we give necessary preliminaries and observations. In Sect. 3, we first present hardness results for the problems  $d-SC1S-R$ ,  $d-SC1S-C$ ,  $d-SC1S-RC$ ,  $d-SC1P-0E$  and  $d-SC1S-01E$ . Then we present an FPT algorithm for the problem  $d-SC1P-0E$  on general binary matrices. We also give FPT algorithms for the problems posed under  $SC1S$  and  $SC1E$  category on certain restricted classes of matrices. Last section draws conclusions and gives an insight to further work.

## 2 Preliminaries

In this section, we present definitions and notations related to binary matrix and graphs associated with binary matrices.

*Graphs:* All graphs discussed in this paper shall always be *undirected* and *simple*. We refer the reader to [17] for the standard definitions and notations related to graphs. A *Hamiltonian path* is a path that visits every vertex exactly once. A *chord* in a cycle is an edge that is not part of the cycle but connects two non-consecutive vertices in the cycle. A *hole* or *chordless cycle* is a cycle of length at least four, where no chords exist. A graph is *chordal* if it contains no hole. That is, in a chordal graph, every cycle of length at least 4 contains a chord.

**Lemma 4** ([18, Theorem 2]). *In a graph  $G = (V, E)$ , a chordless cycle can be detected in  $O(|V| + |E|)$ -time, where  $|V|$  and  $|E|$  are the number of vertices and edges in  $G$  respectively.*

A graph  $G = (V, E)$  is called a *triangle-free* ( $C_3$ -free) graph if it does not contain  $C_3$  as a subgraph. A *connected component* of  $G$  is a maximal connected subgraph of  $G$ . Deletion of a vertex  $v \in V$  means, deleting  $v$  and all edges incident on  $v$ . A graph is a *proper interval* graph if it is an interval graph that has an intersection model in which no interval properly contains another. Given a graph, the problem of deciding whether there exists a set of vertices of size at most  $k$  whose deletion results in a proper interval graph is known as *proper interval vertex deletion* [16] problem.

A graph  $G = (V, E)$  is *bipartite* if  $V$  can be partitioned into two disjoint vertex sets  $V_1$  and  $V_2$  such that every edge in  $E$  has one endpoint in  $V_1$  and the other endpoint in  $V_2$ . A bipartite graph is denoted as  $G = (V_1, V_2, E)$ , where  $V_1$  and  $V_2$  are the two partitions of  $V$ . A bipartite graph is *chordal bipartite* if each cycle of length at least 6 has a chord. A bipartite graph  $H$ , which is an even chordless cycle of length  $2n$  can be converted to a chordal bipartite graph by adding  $n - 2$  edges. This observation is also mentioned in a different form in ([2, Lemma 4.2]). The number of ways to achieve this is given in the following lemma.

**Lemma 5** ([2, Lemma 4.3]). *Given a bipartite graph  $H = (V_1, V_2, E)$  which is an even chordless cycle of length  $2n$  (where  $n \geq 3$ ), then the number of ways to make  $H$  a chordal bipartite graph by adding  $n - 2$  edges is equal to the number of ternary trees with  $n - 1$  internal nodes and is no greater than  $8^{n-1}$ .*

We used the following Lemma to get a tighter upper bound of  $6.75^{n-1}$ .

**Lemma 6** [19].  $\lim_{n \rightarrow \infty} n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ .

A bipartite graph  $G = (V_1, V_2, E)$  is *biconvex* if both  $V_1$  and  $V_2$  can be ordered so that for every vertex  $v$  in  $V_1 \cup V_2$ , neighbours of  $v$  occur consecutively in the ordering. Given a bipartite graph, deciding if there exists a set of vertices of

size at most  $k$ , whose deletion results in a biconvex graph is known as *biconvex deletion problem*. The results in [20,21] shows that biconvex deletion problem is NP-complete. A bipartite graph  $G = (V_1, V_2, E)$  is called a *chain graph* [22] if there exists an ordering  $\pi$  of the vertices in  $V_1, \pi : \{1, 2, \dots, |V_1|\} \rightarrow V_1$  such that  $N(\pi(1)) \subseteq N(\pi(2)) \subseteq \dots \subseteq N(\pi(|V_1|))$ , where  $N(\pi(i))$  denotes the set of neighbours of  $\pi(i)$  in  $G$ . Given a bipartite graph  $G = (V_1, V_2, E)$ , the problem of deciding whether  $k$  number of edges can be added to  $G$  to make it a chain graph is known as *Minimum Chain Completion* problem. Yannakakis showed that Minimum Chain Completion problem is NP-complete [23]. He also developed finite forbidden induced subgraph characterization for chain graphs. Accordingly a bipartite graph  $G = (V_1, V_2, E)$  is a chain graph if and only if it does not contain  $2K_2$  as an induced subgraph. Given a bipartite graph, the problem of deciding whether  $k$  number of edges can be added and removed to the graph to make it a chain graph is called *k-chain editing* problem. *k-chain editing* is known to be NP-Complete [24].

*Matrices:* Given an  $m \times n$  matrix  $M$ , the  $n \times m$  matrix  $M'$  with  $m'_{ji} = m_{ij}$  is called the *transpose* of  $M$ , and is denoted by  $M^T$ . Two matrices  $M$  and  $M'$  are *isomorphic*, if  $M$  is a permutation of the rows and columns of  $M'$ . We say, a matrix  $M$  *contains*  $M'$ , if  $M$  contains a submatrix that is isomorphic to  $M'$ . The *configuration* of an  $m \times n$  matrix  $M$  is defined to be the set of all  $m \times n$  matrices which can be obtained from  $M$  by row and column permutations. The following lemma shows a characterization of the proper interval graph.

**Lemma 7** [15]. *A graph is a proper interval graph if and only if its maximal-clique matrix (vertex-clique incidence matrix) has the SC1P.*

**Definition 1.** *The half adjacency matrix [15] of a bipartite graph  $G = (V_1, V_2, E)$  with  $V_1 = \{u_1, \dots, u_{n_1}\}$  and  $V_2 = \{v_1, \dots, v_{n_2}\}$  is an  $n_1 \times n_2$  matrix  $M_G$  with  $m_{ij} = 1$  if and only if  $\{u_i, v_j\} \in E$ , where  $1 \leq i \leq n_1$  and  $1 \leq j \leq n_2$ .*

Every matrix  $M$  can be viewed as the half adjacency matrix of a bipartite graph, referred to as the *representing graph*  $G_M$  of  $M$ . The *representing graph*  $G_M$  [15] of a matrix  $M_{m \times n}$  is obtained as follows:

**Definition 2.** *For a matrix  $M$ ,  $G_M$  contains a vertex corresponding to every row and every column of  $M$ , and there is an edge between two vertices corresponding to  $i$ th row and  $j$ th column of  $M$  if and only if the corresponding entry  $m_{ij} = 1$ , where  $1 \leq i \leq m$  and  $1 \leq j \leq n$ .*

Following lemma shows a characterization of the half adjacency matrices of biconvex graphs.

**Lemma 8** [3]. *A bipartite graph is biconvex if and only if its half adjacency matrix has the SC1P.*

Following lemma is based on the finite forbidden induced subgraph characterization of chain graphs.

**Lemma 9** [23]. *A bipartite graph  $G = (V_1, V_2, E)$  is a chain graph if and only if its half adjacency matrix  $M_G$  does not contain  $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  as a submatrix.*

The half-adjacency matrix of a chain graph satisfies the *SC1P*, however the converse is not true. A graph  $G$  can also be represented using *edge-vertex incidence matrix* denoted by  $M(G)$  and is defined as follows.

**Definition 3.** *For a graph  $G = (V, E)$ , the rows and columns of  $M(G)$  correspond to edges and vertices of  $G$  respectively. The entries of  $M(G)$  is defined by  $m_{ij} = 1$ , if edge  $e_i$  is incident on vertex  $v_j$ , and  $m_{ij} = 0$  otherwise, where  $1 \leq i \leq m$  and  $1 \leq j \leq n$ .*

Following lemma shows that  $G$  is a path if  $M(G)$  has the *C1P* for rows.

**Lemma 10** ([15, Theorem 2.2]). *If  $G$  is a connected graph and the edge-vertex incidence matrix  $M(G)$  of  $G$  has the *C1P* for rows, then  $G$  is a path.*

### 3 Our Results

#### 3.1 Hardness Results

Even though the number of forbidden submatrices to establish the *SC1P* is less than the number of forbidden submatrices for the *C1P*, the problems posed in this paper, to obtain the *SC1P* turn out to be NP-complete. We refer to the decision versions of the optimization problems *SC1S-row deletion*, *SC1S-column deletion*, *SC1S-row-column deletion*, *SC1P-0-Flipping* and *SC1P-01-Flipping* defined in Sect. 1 as *k-SC1S-R*, *k-SC1S-C*, *k-SC1S-RC*, *k-SC1P-0E* and *k-SC1P-01E*, where  $k$  denotes the number of allowed operations. Here, we show that these problems are NP-complete.

The following theorem proves the NP-completeness of the *k-SC1S-R* problem using Hamiltonian path as a candidate problem.

**Theorem 2.** *Given an  $m \times n$  matrix  $M$ , deciding if there exists a set of rows  $R' \subseteq R(M)$ , such that  $|R'| \leq k$  and  $M \setminus R'$  has the *SC1P* is NP-complete.*

*Proof.* We first show that *k-SC1S-R*  $\in$  NP. Given a matrix  $M$  and an integer  $k$ , the certificate chosen is the given set of rows  $R' \subseteq R(M)$ . The verification algorithm affirms that  $|R'| \leq k$ , and then it checks whether deleting these  $k$  rows from  $M$  yields a matrix with the *SC1P*. This certificate can be verified in polynomial-time.

We prove that *k-SC1S-R* problem is NP-hard by showing that Hamiltonian-Path  $\leq_p$  *k-SC1S-R*. Let  $G = (V, E)$  be a graph with  $|V| = n$  and  $|E| = m$ , and  $M(G)_{m \times n}$  be the edge-vertex incidence matrix (see Definition 3) obtained from  $G$ . Without loss of generality, assume that  $G$  is connected and let  $k$  be  $m - n + 1$ . We show that  $G$  has a Hamiltonian path if and only if there exists a set of rows of size  $k$  in  $M(G)$  whose deletion results in a matrix  $M'(G)$ , that satisfy the *SC1P*.

Assume that  $G$  contains a Hamiltonian path. In  $M(G)$ , delete the rows that correspond to edges which are not part of the Hamiltonian path in  $G$ . Since Hamiltonian path contains  $n - 1$  edges, the number of rows remaining in  $M(G)$  will be  $n - 1$  which is equal to  $m - k$  and hence the number of rows deleted will be  $k$ . Now order the columns and rows of  $M(G)$  with respect to the sequence of vertices and edges respectively in the Hamiltonian path. Clearly, the resulting matrix has the  $SC1P$ .

To prove the other direction, let  $M'(G)$  be the matrix obtained by deleting  $k$  rows from  $M(G)$  and assume that  $M'(G)$  has the  $SC1P$ . Now, the number of rows in  $M'(G)$  is  $m - k$  which is equal to  $n - 1$ . Let  $G'$  be the subgraph obtained from  $M'(G)$ , by considering  $M'(G)$  as an edge-vertex incidence matrix of  $G'$ . Since  $M'(G)$  has the  $SC1P$ , it has the  $C1P$  for rows. Also, note that  $M'(G)$  has  $n - 1$  rows. We claim that the subgraph  $G'$  is connected, otherwise one of the connected components of  $G'$  must contain a cycle which contradicts the fact that  $M'(G)$  has the  $C1P$  for rows. This implies that  $G'$  is a path (see Lemma 10) of length  $n - 1$ , which clearly indicates that  $G$  has a Hamiltonian path. The column permutation needed to convert  $M'(G)$  into a matrix that has the  $C1P$  for rows gives the relative order of vertices of  $G$ 's Hamiltonian path. This proves the NP-completeness of  $k$ - $SC1S$ - $R$ .  $\square$

**Corollary 1.** *The problem  $k$ - $SC1S$ - $C$  is NP-complete.*

*Proof.* The NP-completeness of  $k$ - $SC1S$ - $C$  can be proved similar to Theorem 2 (NP-completeness of  $k$ - $SC1S$ - $R$ ) by considering  $M$  as the vertex-edge incidence matrix and  $k$  as the number of columns to be deleted.  $\square$

To prove the NP-completeness of the  $k$ - $SC1S$ - $RC$  problem, we use the biconvex deletion problem (Sect. 2) as a candidate problem. The following theorem proves the NP-completeness of  $k$ - $SC1S$ - $RC$ .

**Theorem 3.** *The  $k$ - $SC1S$ - $RC$  problem is NP-complete.*

*Proof.* It is easy to show that  $k$ - $SC1S$ - $RC \in NP$ . We prove that  $k$ - $SC1S$ - $RC$  problem is NP-hard by showing that biconvex deletion problem  $\leq_p k$ - $SC1S$ - $RC$ . Let  $G = (V_1, V_2, E)$  be a bipartite graph and  $M$  be a half adjacency matrix (see Definition 2) of  $G$ . Using Lemma 8, it can be shown that  $G$  has a set of vertices  $V'_1 \subseteq V_1$  and  $V'_2 \subseteq V_2$  with  $|V'_1| + |V'_2| \leq k$ , whose deletion results in a biconvex graph if and only if there exists a set of rows  $R' \subseteq R(M)$  and columns  $C' \subseteq C(M)$ , with  $|R'| + |C'| \leq k$  in  $M$  whose deletion results in a matrix  $M'$ , that satisfy the  $SC1P$ . Therefore  $k$ - $SC1S$ - $RC$  is NP-complete.  $\square$

The following theorem proves the NP-completeness of the  $k$ - $SC1P$ - $0E$  problem using the chain completion problem (Sect. 2) on bipartite graphs as a candidate problem.

**Theorem 4.** *The  $k$ - $SC1P$ - $0E$  problem is NP-complete.*

*Proof.* We first show that  $k\text{-SC1P-0E} \in NP$ . Given a matrix  $M$  and an integer  $k$ , the certificate is a set  $A'$  of indices corresponding to 0-entries in  $M$ . The verification algorithm checks that  $|A'| \leq k$  and whether flipping these 0-entries in  $M$  yields a matrix with the  $SC1P$ . This verification can be done in polynomial time.

We prove that  $k\text{-SC1P-0E}$  problem is NP-hard by showing that *chain completion problem*  $\leq_p k\text{-SC1P-0E}$ . The half-adjacency matrix of any chain graph can be observed to satisfy the  $SC1P$ , however the converse is not true. Given a bipartite graph  $G = (V_1, V_2, E)$  with  $V_1 = m$  and  $V_2 = n$ , we create a  $2m \times 2n$  binary matrix  $M_{G_{new}}$  as follows.  $M_{G_{new}} = \begin{bmatrix} J_{m,n} & M_G \\ 0_{m,n} & J_{m,n} \end{bmatrix} = \begin{bmatrix} A & B \\ D & C \end{bmatrix}$ , where  $M_G$  is the half adjacency matrix of  $G$ ,  $J_{m,n}$  is an  $m \times n$  matrix with all entries as one and  $0_{m,n}$  is an  $m \times n$  matrix with all entries as zero. It can be noted that adding an edge in  $G$  corresponds to flipping a 0-entry in  $B$ . We show that  $G$  can be converted to a chain graph  $G'$  by adding at most  $k$  edges if and only if there are at most  $k$  number of 0-flippings in  $M_{G_{new}}$ , such that the resultant matrix  $M_{G'_{new}}$  satisfies the  $SC1P$ .

Suppose  $G'$  is a chain graph, then  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  cannot occur exclusively in  $B$  (from Lemma 9). By construction of  $M_{G_{new}}$ , it can be observed that  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$  and  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$  cannot occur as submatrices in  $M_{G'_{new}}$ . From Fig. 1, it is clear that one of the configurations of these two matrices occur as a submatrix in all the forbidden submatrices of the  $SC1P$ , except  $M_{I_1}$ . Hence  $M_{I_1}$  is the only forbidden submatrix of the  $SC1P$  that could appear in  $M_{G'_{new}}$ . However, if  $M_{G'_{new}}$  contains  $M_{I_1}$ , then it would further imply that  $B'$ (matrix obtained after flipping the 0-entries of  $B$ ) contains  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  as a submatrix, which contradicts the assumption that  $G'$  is a chain graph. Therefore, if  $k$  edges can be added to  $G$  to make it a chain graph, then  $k$  0-entries can be flipped in  $M_{G_{new}}$  to make it satisfy the  $SC1P$ .

Conversely, suppose that  $k = k_1 + k_2$  0-flippings are performed on  $M_{G_{new}}$  to make it satisfy the  $SC1P$ , where  $k_1$  and  $k_2$  refer to the number of 0-flippings performed in  $B$  and  $D$  respectively. Let us assume that the corresponding bipartite graph  $G'$ , obtained after the flipping of zeroes in  $B$  is not a chain graph. Since  $G'$  is not a chain graph, it contains  $2K_2$  as an induced subgraph, which further means that  $B'$  contains  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  as a submatrix. The construction of  $M_{G_{new}}$  implies that  $M_{G'_{new}}$  has  $M_{I_1}$  as a submatrix (considering the remaining 3 quadrants of  $M_{G'_{new}}$ ), which leads to a contradiction. Hence  $G'$  is a chain graph. Therefore,  $k\text{-SC1P-0E}$  is NP-complete.  $\square$

The following theorem proves the NP-completeness of the  $k\text{-SC1P-01E}$  problem using the  $k$ -chain editing problem (Sect. 2) on bipartite graphs as a candidate problem.

**Theorem 5.** *The  $k\text{-SC1P-01E}$  problem is NP-complete.*

---

**Algorithm 1.** Algorithm  $d$ -SC1P-0-Flipping( $M, d$ )

---

**Input:** An instance  $\langle M_{m \times n}, d \rangle$  where  $M$  is a binary matrix and  $d \geq 0$ .

**Output:** Return Yes, if there exists a set of indices,  $B' \subseteq B$  with  $|B'| \leq d$  such that the resultant matrix obtained by flipping the indices of  $B'$  in  $M$  has the SC1P, otherwise return No.

- 1: **if**  $M$  has the SC1P and  $d \geq 0$  **then** return Yes.
  - 2: **if**  $d < 0$  **then** return No. **Branching Step:**
  - 3: **if**  $M$  contains a forbidden submatrix  $M'$  from  $X$  **then**,  
     Branch into at most 18 instances  $I_i = \langle M_i, d_i \rangle$  where  $i \in \{1, 2, \dots, 18\}$   
     Set  $M_i \leftarrow M$  with  $i$ th 0-entry of  $M'$  flipped (where 0-entries of  $M'$  are named in row-major order).  
     Update  $d_i \leftarrow d - 1$  // Decrement parameter by 1.  
     For some  $i \in \{1, 2, \dots, 18\}$ , if  $d$ -SC1P-0-Flipping( $M_i, d_i$ ) returns Yes, then return Yes, else if all instances return No, then return No.
  - 4: **else**
  - 5:     **if**  $M$  contains either  $M_{I_k}$  or  $M_{I_k}^T$  **then**,
  - 6:         Find a minimum size  $M_{I_k}$  or  $M_{I_k}^T$  in  $M$ , (say  $M'$ )
  - 7:         **if**  $k > d$  **then**, return No.
  - 8:         **else**
  - 9:             Branch into at most  $O(46^d)$  (number of ways to destroy  $M'$ ) instances  $I_i = \langle M_i, d_i \rangle$  where  $i \in \{1, 2, \dots, 46^d\}$ .  
            Set  $M_i \leftarrow M$  with  $k$  appropriate 0-entries of  $M'$  flipped.  
            Update  $d_i \leftarrow d - k$  // Decrement parameter by  $k$ .
  - 10:         **end if**
  - 11:     **end if**
  - 12: **end if**  
     For some  $i \in \{1, 2, \dots, O(46^d)\}$ , if  $d$ -SC1P-0-Flipping( $M_i, d_i$ ) returns Yes, then return Yes, else if all instances return No, then return No.
- 

*Proof.* The NP-completeness of  $k$ -SC1P-01E can be proved similar to the NP-completeness of  $k$ -SC1P-0E (Theorem 4) by considering  $M_{G_{new}}$  as follows:

$$M_{G_{new}} = \begin{bmatrix} J_{m,mn} & M_G \\ 0_{mn,mn} & J_{mn,n} \end{bmatrix} \text{ where } G = (P, Q, E) \text{ is a bipartite graph, with } |P|=m \text{ and } |Q|=n \text{ and } M_G \text{ being the half adjacency matrix of } G.$$

### 3.2 An FPT Algorithm for $d$ -SC1P-0E Problem

Here, we present an FPT algorithm  $d$ -SC1P-0-Flipping (Algorithm 1) for the problem  $d$ -SC1P-0E on general binary matrices. Given a binary matrix  $M$  and a nonnegative integer  $d$ , Algorithm 1 destroys forbidden submatrices in  $M$  using a simple search tree based branching algorithm. The algorithm recursively branches if  $M$  contains a forbidden matrix from  $X$  (see Sect. 1) as well as  $M_{I_k}$  or  $M_{I_k}^T$  (where  $k \geq 1$ ). If  $M$  contains a forbidden matrix from  $X$ , then the algorithm branches into at most eighteen subcases, since the largest forbidden matrix of  $X$  has eighteen 0-entries. In each subcase, flip one of the 0-entry of the forbidden submatrix found in  $M$  and decrement the parameter  $d$  by one. Otherwise, if  $M$

contains a forbidden submatrix of type  $M_{I_k}$  or  $M_{I_k}^T$ , then the algorithm finds a minimum size forbidden matrix  $M'$  of type  $M_{I_k}$  or  $M_{I_k}^T$  in  $M$ . If the value of  $k$  is greater than  $d$ , then the algorithm returns No (using Corollary 2), otherwise the algorithm branches into at most  $O(46^d)$ -subcases (using Lemma 11). In each subcase, flip  $k$  0-entries of the forbidden submatrix  $M'$  found in  $M$  and decrement the parameter  $d$  by  $k$ . This process is continued in each subcase until its  $d$  value becomes zero or until it satisfies the *SC1P*. Algorithm 1 returns Yes if any of the subcases returns Yes, otherwise it returns No. Flipping a 0-entry in  $M$  is equivalent to adding an edge in the representing graph  $G_M$  of  $M$ . From this fact and Lemma 3, it follows that to destroy  $M_{I_k}$  and  $M_{I_k}^T$  in  $M$ , it is sufficient to destroy chordless cycles of length greater than four in  $G_M$  (i.e. make  $G_M$  a chordal bipartite graph (Sect. 2) by addition of edges).

**Corollary 2.** *The minimum number of 0-flippings required to destroy an  $M_{I_k}$  or  $M_{I_k}^T$ , where  $(k \geq 1)$  is  $k$ .*

*Proof.* It follows from Lemma 3 and 5.

**Observation 1.** *The number of 0-entries in an  $M_{I_k}$  or  $M_{I_k}^T$ , where  $(k \geq 1)$  is  $O(k^2)$ .*

The above observation leads to a  $O^*(d^{2d})$  algorithm for  $d$ -*SC1P-0E*. But, using the result of the following lemma, we get a  $O^*(45.5625^d)$  algorithm for  $d$ -*SC1P-0E*.

**Lemma 11.** *Given a bipartite graph  $H = (V_1, V_2, E)$  which is an even chordless cycle of length  $2n$  (where  $n \geq 3$ ), then the number of ways to make  $H$  a chordal bipartite graph by adding  $n - 2$  edges is at most  $6.75^{n-1}$ .*

*Proof.* Number of ways to make  $H$  a chordal bipartite graph = Number of ternary trees with  $n - 1$  internal nodes (using Lemma 5).

$$\begin{aligned} \text{Number of ternary trees with } n \text{ internal nodes} &= \frac{\binom{3n+1}{n}}{3n+1} \\ &= \frac{\binom{3n}{n}}{2n+1} = \frac{(3n)!}{(2n+1)(2n)!n!} \\ \lim_{n \rightarrow \infty} n! &= \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \text{ (using Lemma 6).} \\ \lim_{n \rightarrow \infty} \frac{\binom{3n}{n}}{2n+1} &= \frac{\sqrt{2\pi(3n)} \left(\frac{3n}{e}\right)^{3n}}{\sqrt{2\pi(2n)} \left(\frac{2n}{e}\right)^{2n} \times \sqrt{2\pi(n)} \left(\frac{n}{e}\right)^n \times (2n+1)} \\ &= \frac{\sqrt{3} \times 3^{3n}}{\sqrt{4\pi n} \times 2^{2n} \times (2n+1)} \\ \frac{\binom{3n}{n}}{2n+1} &= O\left(\frac{3^{3n}}{\sqrt{n} \times 2^{2n} \times (2n+1)}\right) \sim O\left(\frac{3^{3n}}{2^{2n}}\right) = O(6.75^n) \end{aligned}$$



Therefore, number of ternary trees with  $n$  internal nodes =  $O(6.75^n)$ .

Hence, the number of ways to make  $H$  a chordal bipartite graph is same as the number of ternary trees with  $n - 1$  internal nodes and is  $O(6.75^{n-1})$ .  $\square$

**Theorem 6.**  *$d$ -SC1P-0E problem on a matrix  $M_{m \times n}$ , can be solved in  $O^*(45.5625^d)$ -time, where  $d$  denotes the number of 0-entries that can be flipped. Consequently it is FPT.*

*Proof.* The technique used in Algorithm 1 employs a search tree. Each node in the search tree has at most  $18^d$  or  $O(46^d)$  subproblems, depending on whether we are destroying the fixed size forbidden matrices or  $M_{I_k}$  respectively. In the worst case, the parameter  $d$  is decremented by one at each level. This occurs when the chordless cycle found is of length six. So, the total number of branches will be  $6.75^{\frac{6}{2}-1}$  (using Lemma 11). Therefore the tree has at most  $O(46^d)$  leaves. The size of the search tree is  $O(46^d)$ . A submatrix  $M'$  of  $M$ , that is isomorphic to one of the forbidden matrices in  $X$  and  $M_{I_k}$  or  $M_{I_k}^T$  can be found in  $O(m^6n)$ -time (using Lemma 1) and  $O(m^3n^3)$ -time (using Lemma 2) respectively. Therefore the total time complexity of Algorithm 1 is  $O^*(45.5625^d)$ .  $\square$

For  $(2, 2)$ -matrices, the problems  $d$ -SC1S-R,  $d$ -SC1S-C,  $d$ -SC1S-RC,  $d$ -SC1P-1E,  $d$ -SC1P-0E and  $d$ -SC1P-01E defined in Sect. 1 turn out to be polynomial time solvable, because these problems correspond to removing chordless cycles in a certain graph representation of the matrix by deleting vertices/edges depending on the variant considered. Since the chordless cycles in graphs associated with  $(2, 2)$ -matrices are disjoint, it is easy to determine an optimal solution. For  $(2, *)$  and  $(*, 2)$ -matrices, the SC1S and SC1E problems are FPT and admit constant factor polynomial-time approximation algorithms. The details are omitted due to space constraint.

## 4 Conclusion

We showed that the decision versions of the SC1S and SC1E problems are NP-complete. We showed that  $d$ -SC1P-0E problem is fixed-parameter tractable on general binary matrices. We also show that the parameterized versions of SC1S and SC1E problems on  $(2, *)$ -matrices and  $(*, 2)$ -matrices are FPT. We also observe that the fixed-parameter tractability of  $d$ -SC1S-R problem on  $(2, *)$ -matrices shows that proper interval vertex deletion problem is FPT on triangle free graphs with a run-time of  $O^*(4^d)$ .

## References

1. Oswald, M., Reinelt, G.: The simultaneous consecutive ones problem. Theoret. Comput. Sci. **410**(21–23), 1986–1992 (2009)
2. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. SIAM J. Comput. **28**(5), 1906–1922 (1999)

3. Tucker, A.: A structure theorem for the consecutive 1's property. *J. Comb. Theory Ser. B* **12**(2), 153–162 (1972)
4. Fishburn, P.C.: Interval orders and interval graphs. *Discret. Math.* **55**(2), 135–149 (1985)
5. Oswald, M.: Weighted consecutive ones problems. Ph.D. thesis (2003)
6. König, R., Schramm, G., Oswald, M., Seitz, H., Sager, S., Zapatka, M., Reinelt, G., Eils, R.: Discovering functional gene expression patterns in the metabolic network of *escherichia coli* with wavelets transforms. *BMC Bioinform.* **7**(1), 119 (2006)
7. Fulkerson, D., Gross, O.: Incidence matrices and interval graphs. *Pac. J. Math.* **15**(3), 835–855 (1965)
8. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.* **13**(3), 335–379 (1976)
9. Hsu, W.L.: A simple test for the consecutive ones property. *J. Algorithms* **43**(1), 1–16 (2002)
10. Hsu, W.L., McConnell, R.M.: PC trees and circular-ones arrangements. *Theoret. Comput. Sci.* **296**(1), 99–116 (2003)
11. McConnell, R.M.: A certifying algorithm for the consecutive-ones property. In: *Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 768–777. Society for Industrial and Applied Mathematics (2004)
12. Meidanis, J., Porto, O., Telles, G.P.: On the consecutive ones property. *Discret. Appl. Math.* **88**(1–3), 325–354 (1998)
13. Raffinot, M.: Consecutive ones property testing: cut or swap. In: Löwe, B., Normann, D., Soskov, I., Soskova, A. (eds.) *CiE 2011. LNCS*, vol. 6735, pp. 239–249. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-21875-0\\_25](https://doi.org/10.1007/978-3-642-21875-0_25)
14. Downey, R.G., Fellows, M.R.: *Fundamentals of Parameterized Complexity*, vol. 4. Springer, London (2013). <https://doi.org/10.1007/978-1-4471-5559-1>
15. Dom, M.: *Recognition, Generation, and Application of Binary Matrices with the Consecutive Ones Property*. Cuvillier, Gottingen (2009)
16. van't Hof, P., Villanger, Y.: Proper interval vertex deletion. *Algorithmica* **65**(4), 845–867 (2013)
17. West, D.B.: *Introduction to Graph Theory*, vol. 2. Prentice Hall, Upper Saddle River (2009)
18. Uno, T., Satoh, H.: An efficient algorithm for enumerating chordless cycles and chordless paths. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) *DS 2014. LNCS (LNAI)*, vol. 8777, pp. 313–324. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-11812-3\\_27](https://doi.org/10.1007/978-3-319-11812-3_27)
19. Stirling, J.: *Methodus differentialis, sive tractatus de summation et interpolation serierum infinitarum*, London. *The Differential Method: A Treatise of the Summation and Interpolation of Infinite Series* (1730). (Trans. by, J. Holliday)[1749](1730)
20. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: *Proceedings of 10th Annual ACM Symposium on Theory of Computing*, pp. 253–264. ACM (1978)
21. Yannakakis, M.: Node-deletion problems on bipartite graphs. *SIAM J. Comput.* **10**(2), 310–327 (1981)
22. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. *Discret. Appl. Math.* **113**(1), 109–128 (2001)

23. Yannakakis, M.: Computing the minimum fill-in is NP-complete. *SIAM J. Algebr. Discret. Methods* **2**(1), 77–79 (1981)
24. Drange, P.G., Dregi, M.S., Lokshantov, D., Sullivan, B.D.: On the threshold of intractability. In: Bansal, N., Finocchi, I. (eds.) *ESA 2015*. LNCS, vol. 9294, pp. 411–423. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48350-3\\_35](https://doi.org/10.1007/978-3-662-48350-3_35)



# Improved Kernels for Several Problems on Planar Graphs

Qilong Feng, Beilin Zhuo, Guanlan Tan, Neng Huang, and Jianxin Wang<sup>(✉)</sup>

School of Information Science and Engineering,  
Central South University, Changsha 410083, People's Republic of China  
jxwang@mail.csu.edu.cn

**Abstract.** In this paper, we study the kernelization of the Induced Matching problem on planar graphs, the Parameterized Planar 4-Cycle Transversal problem and the Parameterized Planar Edge-Disjoint 4-Cycle Packing problem. For the Induced Matching problem on planar graphs, based on Gallai-Edmonds structure, a kernel of size  $26k$  is presented, which improves the current best result  $28k$ . For the Parameterized Planar 4-Cycle Transversal problem, by partitioning the vertices in given instance into four parts and analyzing the size of each part independently, a kernel with at most  $51k - 22$  vertices is obtained, which improves the current best result  $74k$ . Based on the kernelization process of the Parameterized Planar 4-Cycle Transversal problem, a kernel of size  $51k - 22$  can also be obtained for the Parameterized Planar Edge-Disjoint 4-Cycle Packing problem, which improves the current best result  $96k$ .

## 1 Introduction

Given an instance  $(I, k)$  of a parameterized problem  $Q$ , the kernelization process is to transform  $(I, k)$  into a new instance  $(I', k')$  in polynomial time such that  $(I, k)$  is a yes-instance of  $Q$  if and only if  $(I', k')$  is a yes-instance of  $Q$ , where  $k' \leq k$ , and  $|I'| \leq f(k)$  for some computable function  $f$ . In this paper, we study the kernelization of the Induced Matching problem on planar graphs, the Parameterized Planar 4-Cycle Transversal problem, and the Parameterized Planar Edge-Disjoint 4-Cycle Packing problem.

### Induced matching

In graph theory, a matching in a graph  $G = (V, E)$  is a set of edges without common vertices. A matching  $M$  of  $G$  is an induced matching of  $G$  if no edge in  $E - M$  has both endpoints contained in  $V(M)$  ( $V(M)$  is the set of vertices contained in  $M$ ). The Induced Matching problem is to decide whether a given graph  $G$  has an induced matching of size at least  $k$ . The Induced Matching problem was introduced by Stockmeyer and Vazirani [29], and has attracted lots of attention. Duckworth et al. [9] proved that the Induced Matching problem

---

This work is supported by the National Natural Science Foundation of China under Grants (61420106009, 61232001, 61472449, 61672536).

on general graphs is NP-complete. The NP-hardness of the problem was also studied on many restricted graph classes, such as, the bipartite graphs with maximum degree three [23], planar bipartite graphs [9], 3-regular planar graphs [9],  $C_4$ -free bipartite graphs [23], chair-free graphs [19], line-graphs [19], and Hamiltonian graphs [19]. The Induced Matching problem is polynomial time solvable in many graph classes, such as, trees [11, 12], interval-filament graphs [16], AT-free graphs [16], circular arc graphs [16], chordal graphs [5], weakly chordal graphs [7], line-graphs of Hamiltonian graphs [19], polygon-circle graphs [6],  $(P_5, D_m)$ -free graphs [19, 24],  $(P_k, K_{1,n})$ -free graphs [19, 24], trapezoid graphs [12], interval-dimension graphs [12], and comparability graphs [12].

Duckworth et al. [9] proved that the Induced Matching problem is APX-complete on  $r$ -regular graphs ( $r \geq 3$ ) and bipartite graphs with maximum degree three. Orlovich et al. [27] gave that in general graphs, the Induced Matching problem cannot be approximated within a factor of  $n^{1/2-\varepsilon}$  for any  $\varepsilon > 0$ . Chlebík and Chlebíková [8] proved that it is NP-hard to approximate the Induced Matching problem within factor of  $r/2^{O(\sqrt{\ln r})}$  for  $r$ -regular graphs. Duckworth et al. [9] gave an approximation algorithm for the problem on  $r$ -regular graphs ( $r \geq 3$ ) with ratio  $r-1$ , and proposed a polynomial-time approximation scheme (PTAS) for the Induced Matching problem on planar graphs of maximum degree three. Gotthilf and Lewenstein [13] gave an approximation algorithm for the Induced Matching problem with ratio  $0.75r + 0.15$ .

In this paper, we study the following problem.

**Induced Matching problem on planar graphs:** Given a planar graph  $G = (V, E)$  and an integer  $k$ , find an induced matching of size at least  $k$  in  $G$ , or report that no such matching exists.

Moser and Thilikos [25] proved that the Induced Matching problem on general graph is W[1]-hard. It was pointed out in [26] that the Induced Matching problem is even W[1]-hard on bipartite graphs. Based on the kernelization methods in [1], Moser and Sikdar [26] gave a linear kernel for the Induced Matching problem on planar graphs. Kanj et al. [18] improved the above kernel result to  $40k$ . Erman et al. [10] gave that every  $n$ -vertex twinless planar graph contains an induced matching of size  $(n+9)/28$ , and a kernel of size  $28k$  was obtained, which is the current best result. A kernel of size  $2k(1+d+d^2)$  was presented for the Induced Matching problem on degree-bounded graphs with maximum degree  $d$  by Moser and Sikdar [26].

In this paper, we study the Induced Matching problem on planar graphs. The key point to get the improved kernel is based on the analysis of Gallai-Edmonds decomposition structure. Several new reduction rules are presented, which results in a kernel of size  $26k$  for the Induced Matching problem on planar graphs.

### **$s$ -Cycle Transversal**

The  $s$ -Cycle Transversal problem has been widely studied in extremal graph theory [2], graph coloring [35] and computational biology [28], which is to find a set  $S$  of edges of size at most  $k$  in a given graph  $G$  such that  $S$  intersects every cycle of length  $s$  in  $G$ , where  $s$  is a constant. When  $s$  is small, several related

problems have also been studied, such as the chromatic numbers in graphs without 3-cycles [30] and 5-cycles [31], designing Low-Density Parity-Check (LDPC) codes [15] based on Tanner graphs without 4-cycles.

The  $s$ -Cycle Transversal problem for any fixed  $s \geq 3$  is known to be NP-complete on general graphs [34]. Brüggmann et al. [4] showed that the  $s$ -Cycle Transversal problem remains NP-complete on planar graphs for  $s = 3$ . Xia and Zhang [32] proved that the  $s$ -Cycle Transversal problem is NP-complete on planar graphs for any fixed  $s \geq 3$ . Krivelevich [21] presented a 2-approximation algorithm for the 3-Cycle Transversal problem. Kortsarz et al. [20] showed that a  $(2 - \epsilon)$ -approximation algorithm for 3-Cycle Transversal problem implies a  $(2 - \epsilon)$ -approximation algorithm for Vertex Cover problem. Kortsarz et al. [20] presented a generalized  $(s - 1)$ -approximation algorithm for  $s$ -Cycle Transversal problem for odd number  $s$ .

The  $s$ -Cycle Transversal and related problems have also been studied from parameterized complexity point of view, which are defined as follows.

**Parameterized 4-Cycle Transversal:** Given an undirected graph  $G = (V, E)$  and an integer  $k$ , find a subset  $E' \subseteq E$  with  $|E'| \leq k$  such that each 4-cycle in  $G$  contains at least one edge from  $E'$ , or report that no such subset exists.

**Parameterized ( $\leq s$ )-Cycle Transversal:** Given an undirected graph  $G = (V, E)$ , a constant  $s$  and an integer  $k$ , find a subset  $E' \subseteq E$  with  $|E'| \leq k$  such that each ( $\leq s$ )-cycle in  $G$  contains at least one edge from  $E'$ , or report that no such subset exists.

**Parameterized Planar 4-Cycle Transversal:** Given a planar graph  $G = (V, E)$  and an integer  $k$ , find a subset  $E' \subseteq E$  with  $|E'| \leq k$  such that each 4-cycle in  $G$  contains at least one edge from  $E'$ , or report that no such subset exists.

A kernel with  $6k$  vertices and a kernel with  $11k/3$  vertices in general graphs and planar graphs for 3-Cycle Transversal problem were presented in [4], respectively. Xia and Zhang [32] gave that the Parameterized 4-Cycle Transversal problem and the Parameterized ( $\leq 4$ )-Cycle Transversal problem admit a kernel with  $6k^2$  vertices on general graphs. By applying the region decomposition technique developed by Guo and Niedermeier [14], Xia and Zhang [32] obtained several kernelization results on planar graphs: a kernel with  $74k$  vertices for Parameterized 4-Cycle Transversal problem, a kernel with  $32k$  vertices for Parameterized ( $\leq 4$ )-Cycle Transversal and a kernel with  $266k$  vertices for the Parameterized ( $\leq 5$ )-Cycle Transversal problem. Xia and Zhang [33] studied the kernelization of the Parameterized ( $\leq s$ )-Cycle Transversal problem, and obtained a kernel of size  $36s^3k$  for  $s > 5$ .

In this paper, we study the kernelization of the Parameterized Planar 4-Cycle Transversal problem. We give several reduction rules and partition the vertices in given instance into four parts to bound the size of reduced instance. A kernel with at most  $51k - 22$  vertices is obtained for the Parameterized Planar 4-Cycle Transversal problem, which improves the current best result  $74k$ . The kernelization process for the Parameterized Planar 4-Cycle Transversal problem

can be applied to the kernelization of the Parameterized Planar Edge-Disjoint 4-Cycle Packing problem, which is to decide whether  $k$  edge-disjoint 4-cycles can be found in a given planar graph  $G$ . We can get that the Parameterized Planar Edge-Disjoint 4-Cycle Packing problem admits a kernel of size  $51k - 22$ , which improves the current best result given in [17].

## 2 Preliminaries

Given a graph  $G = (V, E)$ , for two vertices  $u, v$  in  $G$ , let  $uv$  denote the edge between  $u$  and  $v$ . For a vertex  $v \in G$ , let  $N(v) = \{u | vu \in E\}$ . For a vertex  $v$  in  $G$ , let  $deg(v)$  denote the degree of  $v$ . A vertex in  $G$  with degree  $d$  is called a degree- $d$  vertex. For a subset  $V' \subseteq V$ , let  $G - V'$  denote the graph obtained by removing the vertices in  $V'$  and all its incident edges from  $G$ . For a subset  $E' \subseteq E(G)$ , let  $G - E'$  denote the graph obtained by deleting all edges in  $E'$  from  $G$ . Assume that all paths discussed in this paper are simple. For two sets  $A, B$ , let  $A \setminus B$  denote  $A - B$ . A 4-cycle in  $G$  is a cycle in  $G$  with four vertices and four edges. An edge subset  $S$  is called a *4-Cycle Transversal set* of  $G$  if  $G \setminus S$  is 4-cycle free. For any cycle  $C$  in  $G$ , let  $E(C)$  be the set of edges contained in  $C$ .

For two vertices  $u, v$  in  $G$ , if  $u$  and  $v$  have the same neighborhood, i.e.,  $N(u) = N(v)$ , then  $u, v$  are called *twin-vertices*. A graph  $G$  is called a *twinless graph* if no twin-vertices are contained in  $G$ . For a subset  $V'$  of  $V$ , the subgraph induced by  $V'$  is denoted by  $G[V']$ . For a set  $S$  of edges of  $G$ , let  $V(S)$  denote the set of vertices contained in  $S$ . For a set  $M$  of edges of  $G$ , if no two edges in  $M$  have common vertices, then  $M$  is a matching of  $G$ , all the vertices in  $M$  are called *matched* vertices, and the vertices in  $V \setminus V(M)$  are called *unmatched* vertices. The size of a matching  $M$  is the number of edges in  $M$ , denoted by  $|M|$ . A maximum matching is a matching that contains the largest possible number of edges. A matching is a perfect matching if all the vertices in graph are matched vertices. For a set  $S$  of edges of  $G$ ,  $S$  is an induced matching of  $G$  if  $S$  satisfies the following properties: (1)  $S$  is a matching of  $G$ ; (2) no edge in  $E \setminus S$  has both endpoints contained in  $V(S)$ . For an induced matching  $S$  of  $G$ , the size of  $S$  is the number of edges contained in  $S$ , denoted by  $|S|$ . For a graph  $G$ , the *independence number* of  $G$  is the size of the maximum independent set of  $G$ .

Given a graph  $G = (V, E)$ , a 4-cycle packing  $\mathcal{P} = \{C_1, C_2, \dots, C_t\}$  of size  $t$  is a collection of  $t$  edge-disjoint 4-cycles, i.e., each element  $C_i \in \mathcal{P}$  is a 4-cycle and  $E(C_i) \cap E(C_j) = \emptyset$  for any two different 4-cycles  $C_i, C_j \in \mathcal{P}$ . A 4-cycle packing is maximal if it is not properly contained in any strictly larger 4-cycle packing in  $G$ . The set of vertices in 4-cycles in  $\mathcal{P}$  is denoted by  $V(\mathcal{P})$ .

## 3 Improved Kernel for the Induced Matching Problem on Planar Graphs

Given an instance  $(G, k)$  of the Induced Matching problem on planar graphs, we first give several reduction rules for the problem.

**Rule 3.1** [26]. For a vertex  $v$  in  $G$  with degree zero, delete  $v$  from  $G$ .

**Rule 3.2** [26]. For a vertex  $v$  in  $G$ , if  $v$  contains at least two degree-1 neighbors, denoted by  $\{u_1, u_2, \dots, u_i\}$  ( $i \geq 2$ ), then delete arbitrarily  $i - 1$  vertices from  $\{u_1, u_2, \dots, u_i\}$ .

**Rule 3.3** [26]. For two vertices  $u, v$  with  $|N(u) \cap N(v)| \geq 2$ , if  $N(u) \cap N(v)$  contains at least two degree-2 vertices, denoted by  $\{w_1, w_2, \dots, w_j\}$  ( $j \geq 2$ ), then delete arbitrarily  $j - 1$  vertices from  $\{w_1, w_2, \dots, w_j\}$ .

**Rule 3.4.** For any two twin-vertices  $u, v$  in  $G$ , delete one of  $\{u, v\}$ .

It is easy to see that if the induced matching contains vertex from  $\{u, v\}$ , then only one of  $\{u, v\}$  is contained in the induced matching, and any one of  $\{u, v\}$  can be in the induced matching.

**Rule 3.5.** For any two vertices  $u, v$  in  $G$ , if there is a degree-2 vertex  $w$  in  $N(u) \cap N(v)$ ,  $u$  has a degree-1 neighbor  $x$ , and  $v$  has a degree-1 neighbor  $y$ , then vertex  $w$  can be deleted.

**Lemma 1.** *Rule 3.5 is correct and can be applied in  $O(n^3)$  time.*

*Proof.* Assume that  $(G, k)$  is an instance of the Induced Matching problem on planar graphs. We prove this lemma based on the following cases.

(1) no edge from  $\{[u, w], [u, x], [v, w], [v, y]\}$  is contained in any induced matching of size at least  $k$  of  $G$ .

Assume that  $S$  is an induced matching of size  $k$  of  $G$  without containing any edge from  $\{[u, w], [u, x], [v, w], [v, y]\}$ . By deleting vertex  $w$ ,  $S$  is still an induced matching of size  $k$  in  $G[V \setminus \{w\}]$ .

(2) one edge from  $\{[u, w], [v, w]\}$  is contained in an induced matching of size at least  $k$  in  $G$ .

Without loss of generality, assume that edge  $[v, w]$  is contained in an induced matching  $S$  of size  $k$  in  $G$ . Let  $S' = (S \setminus \{[v, w]\}) \cup \{[v, y]\}$ . It is easy to see that  $S'$  is an induced matching of size  $k$  in  $G$ .

This reduction rule can be executed in the following way: for each possible  $w$  in  $G$ , check any two vertices  $u, v$  in  $N(w)$ , and decide whether  $u, v$  have degree-1 vertices in their neighbors, respectively. It is easy to see that Rule 3.5 can be applied in  $O(n^3)$  time.  $\square$

**Rule 3.6.** For any three vertices  $v_1, v_2, v_3$  in  $G$ , if there is degree-3 vertex  $u$  in  $N(v_1) \cap N(v_2) \cap N(v_3)$ ,  $v_1$  has a degree-1 neighbor  $x$ ,  $v_2$  has a degree-1 neighbor  $y$ , and  $v_3$  has a degree-1 neighbor  $z$ , then vertex  $u$  can be deleted.

**Lemma 2.** *Rule 3.6 is correct and can be applied in  $O(n^4)$  time.*

*Proof.* Assume that  $(G, k)$  is an instance of the Induced Matching problem on planar graphs. We prove this lemma based on the following cases.

(1) no edge from  $\{[u, v_1], [u, v_2], [u, v_3]\}$  is contained in any induced matching of size at least  $k$  of  $G$ .



Assume that  $S$  is an induced matching of size  $k$  of  $G$  without containing any edge from  $\{[u, v_1], [u, v_2], [u, v_3]\}$ . By deleting vertex  $u$ ,  $S$  is still an induced matching of size  $k$  in  $G[V \setminus \{u\}]$ .

(2) one edge from  $\{[u, v_1], [u, v_2], [u, v_3]\}$  is contained in an induced matching of size at least  $k$  in  $G$ .

Without loss of generality, assume that edge  $[u, v_1]$  is contained in an induced matching  $S$  of size  $k$  in  $G$ . Let  $S' = (S \setminus \{[u, v_1]\}) \cup \{[v_1, x]\}$ . It is easy to see that  $S'$  is an induced matching of size  $k$  in  $G$ .

This reduction rule can be executed in the following way: for each possible  $u$  in  $G$ , check any three vertices  $v_1, v_2, v_3$  in  $N(u)$ , and decide whether  $v_1, v_2, v_3$  have degree-1 vertices in their neighbors, respectively. It is easy to see that Rule 3.6 can be applied in  $O(n^4)$  time.  $\square$

We first introduce the terminologies related to Gallai-Edmonds structure [22].

Given a graph  $G$ , if for each vertex  $v$  in  $G$ ,  $G \setminus \{v\}$  has a perfect matching, then  $G$  is called a *factor-critical* graph. For a subset  $V'$  of vertices in  $G$ , let  $N(V')$  denote all the vertices of  $G$  which are adjacent to at least one vertex in  $V'$ . For a matching  $M$  in  $G$ ,  $M$  is called a *near-perfect matching* of  $G$  if there is exactly one unmatched vertex in  $G$ .

**Theorem 1** (*The Gallai-Edmonds Structure Theorem*) [22]. *For a given graph  $G$ , let  $D$  be the set of vertices which are not covered by at least one maximum matching of  $G$ , let  $A$  be the set of vertices in  $V \setminus D$  which are adjacent to at least one vertex in  $D$ , and let  $C = V \setminus (A \cup D)$ . Then,*

- (a) *the components of the subgraph induced by  $D$  are factor-critical,*
- (b) *the subgraph induced by  $C$  has a perfect matching,*
- (c) *if  $M$  is any maximum matching of  $G$ , it contains a near-perfect matching of each component of  $G[D]$ , a perfect matching of each component of  $G[C]$  and matches all vertices of  $A$  with vertices in distinct components of  $G[D]$ ,*
- (d) *the size of the maximum matching is  $1/2(|V| - c(G[D]) + |A|)$ , where  $c(G[D])$  is the number of components in  $G[D]$ .*

For simplicity, a Gallai-Edmonds structure of graph  $G$  is denoted by  $(C, A, D)$ .

**Lemma 3** [22]. *For a given graph  $G = (V, E)$ , a Gallai-Edmonds structure  $(C, A, D)$  of  $G$  can be obtained in polynomial time.*

The relationship between maximum matching and induced matching can be obtained as follows.

**Lemma 4** [18]. *Let  $\mathcal{G}$  be a minor-closed family of graphs and let  $c$  be a constant such that any graph in  $\mathcal{G}$  is  $c$ -colorable. Moreover, let  $G$  be a graph from  $\mathcal{G}$  and let  $M$  be a matching in  $G$ . Then  $G$  contains an induced matching of size at least  $|M|/c$ .*

For an instance  $(G, k)$  of the Induced Matching problem on planar graphs, apply Rules 3.1–3.6 whenever possible on  $G$ . Let  $(G' = (V', E'), k')$  be the reduced instance such that no rule is applicable on  $G'$ .

**Theorem 2.** *The Induced Matching problem on planar graphs admits a kernel of size  $26k$ .*

*Proof.* For a Gallai-Edmonds structure  $(C, A, D)$  of  $G'$ , the components in  $G'[D]$  are divided into two parts  $S, T$  such that  $T$  contains the set of components, each of which has at least three vertices, and  $S$  contains the set of isolated vertices in  $G'[D]$ . Let  $T_{2i+1}$  ( $i \geq 1$ ) be a subset of  $T$  such that each component in  $T_{2i+1}$  has  $2i + 1$  vertices. Assume that  $T = \bigcup_{i=1}^h T_{2i+1}$ . Let  $S_1 = \{u|u \in S, \text{deg}(u) = 1\}$ ,  $S_2 = \{u|u \in S, \text{deg}(u) = 2\}$ , and  $S_3 = \{u|u \in S, \text{deg}(u) \geq 3\}$ . Since Rule 3.4 is not applicable on  $G'$ ,  $G'$  contains no twin-vertices. Therefore, in the subgraph induced by the vertices in  $A \cup D$ , by Euler formula,  $|S_2| \leq 3|A| - 6$ , and  $|S_3| \leq 2|A| - 4$ . We discuss the size of  $S$  by the following cases.

(1)  $0 \leq |S_1| < |A|/2$ .

Under this case, we can get that:

$$\begin{aligned} |S| &= |S_1| + |S_2| + |S_3| \\ &\leq |S_1| + 3|A| - 6 + 2|A| - 4 \\ &\leq |S_1| + 5|A| - 10 \\ &< 5.5|A| - 10 \end{aligned}$$

(2)  $|A|/2 \leq |S_1| \leq |A|$ .

By Rule 3.5, if there exists a degree-2 vertex  $w$  in common neighbors of  $u, v$  and both  $u, v$  have degree-1 neighbors, then vertex  $w$  can be deleted. Therefore, if  $|A|/2 \leq |S_1| \leq |A|$ , then the number of degree-2 vertices in  $S_2$  is bounded by  $3|A| - 6 - (|S_1| - |A|/2)$ . Then, we can get that

$$\begin{aligned} |S| &= |S_1| + |S_2| + |S_3| \\ &\leq |S_1| + 3|A| - 6 - (|S_1| - |A|/2) + 2|A| - 4 \\ &\leq 5.5|A| - 10 \end{aligned}$$

By the above two cases, we can get that  $|S| \leq 5.5|A| - 10$ .

For a subset  $T_{2i+1}$  of  $T$  and for a maximum matching  $M$  in  $G'$ , at least  $i$  edges of  $T_{2i+1}$  can be added into  $M$ . Therefore, we can get that

$$\begin{aligned} \frac{|M|}{|V'|} &\geq \frac{|A| + \sum_{i=1}^h i|T_{2i+1}| + 1/2|C|}{|A| + 5.5|A| - 10 + \sum_{i=1}^h (2i + 1)|T_{2i+1}| + |C|} \\ &> \frac{1}{6.5} \end{aligned}$$

Then,  $|V'| < 6.5|M|$ . Let  $I$  be any induced matching of size  $k$  in  $G'$ . By Lemma 4 and the Four-color theorem of planar graphs,  $|M| \leq 4|I|$ . Therefore,  $|V'| < 6.5 \cdot 4|I| \leq 26k$ . □

## 4 Improved Kernel for the Parameterized Planar 4-Cycle Transversal Problem

For a given instance  $(G = (V, E), k)$  of the Parameterized Planar 4-Cycle Transversal problem, we firstly find a maximal 4-cycle packing  $\mathcal{P}$  in  $G$ , and let  $Q = V - V(\mathcal{P})$ . We can get that the size of  $V(\mathcal{P})$  is at most  $4k$ , and  $Q$  contains no 4-cycle. The remaining task is to bound the size of  $Q$ . We first give several reduction rules.

**Rule 4.1.** If there exists an edge  $e \in E$  which is not contained in any 4-cycle, then delete  $e$  from  $G$ ; if there exists a vertex  $v$  in  $G$  not contained in any cycle, then delete  $v$  from  $G$ .

It is easy to see that Rule 4.1 is safe and can be executed in polynomial time. For any vertex  $v$  in  $G$  and any cycle  $C$  in  $\mathcal{P}$ , if  $v$  is connected to at least one vertex in  $C$ , then we call  $v$  is adjacent to  $C$ . For each cycle  $C \in \mathcal{P}$ , let  $Q(C)$  denote the set of vertices in  $Q$  that are adjacent to  $C$ .

**Rule 4.2.** If there is a 4-cycle  $C \in \mathcal{P}$  with  $V' = Q(C) \cup V(C)$  such that  $G[V']$  contains at least two edge-disjoint 3-cycles, then replace  $C$  by these 4-cycles in  $\mathcal{P}$ .

**Rule 4.3.** If there are two 4-cycles  $C_1, C_2 \in \mathcal{P}$  with  $V'' = Q(C_1) \cup V(C_1) \cup Q(C_2) \cup V(C_2)$  such that  $G[V'']$  contains at least three edge-disjoint 4-cycles, then replace  $C_1, C_2$  in  $\mathcal{P}$  by these 4-cycles in  $\mathcal{P}$ .

Each execution of Rule 4.2 and Rule 4.3 can be done in polynomial time, and increases the number of 4-cycles in  $\mathcal{P}$  by at least 1.

For a given instance  $(G, k)$  of the Parameterized Planar 4-Cycle Transversal problem, Rule 4.3 is applied when Rule 4.2 is not applicable on graph  $G$ . Note that after each application of Rule 4.3, the updated 4-cycle packing  $\mathcal{P}$  is still maximal. For simplicity, a maximal 4-cycle packing  $\mathcal{P}$  is called a *proper* 4-cycle packing if neither Rule 4.2 nor Rule 4.3 is applicable to update  $\mathcal{P}$ .

In the following, we assume that  $\mathcal{P}$  is a proper 4-cycle packing obtained by applying Rules 4.1–4.3 exhaustively, and let  $Q = V - V(\mathcal{P})$ . We now discuss the properties of the edges in  $G[Q]$ . For an edge  $e = uv$  in  $G[Q]$ , if there exists a cycle  $C$  in  $\mathcal{P}$  such that  $u, v$  with two adjacent vertices in  $C$  form a 4-cycle, then edge  $uv$  is called a *single edge* in  $G[Q]$ , and we say  $e$  is adjacent to cycle  $C$ .

**Lemma 5.** *Let  $C$  be an arbitrary 4-cycle in  $\mathcal{P}$ , and let  $R$  be the set of vertex-disjoint single edges in  $G[Q]$  adjacent to  $C$ . If  $|R| \geq 2$ , then all the single edges in  $R$  must be adjacent to a unique edge in  $C$ .*

For a 4-cycle  $C$  in  $G$ , if only one edge of  $C$  is shared with other 4-cycles in  $G$ , then  $C$  is called a *dangling* cycle in  $G$ .

**Rule 4.4.** For a dangling 4-cycle  $C$  in  $G$ , all the edges in  $C$  can be deleted from  $G$ , and  $k = k - 1$ .

Let  $(G, k)$  be the reduced instance of the Parameterized Planar 4-Cycle Transversal problem by exhaustively applying reduction Rules 4.1–4.4. We now analyze the size of  $G$ . Assume that  $\mathcal{P}$  is a proper 4-cycle packing in  $G$ , and let  $Q = V - V(\mathcal{P})$ . We divide the vertices in  $Q$  into the following parts:  $Q_1 = \{v \in Q \mid |N(v) \cap V(\mathcal{P})| = 1\}$ ,  $Q_2 = \{v \in Q \mid |N(v) \cap V(\mathcal{P})| = 2\}$ ,  $Q_3 = \{v \in Q \mid |N(v) \cap V(\mathcal{P})| \geq 3\}$ ,  $Q_0 = Q \setminus (Q_1 \cup Q_2 \cup Q_3)$ .

Since  $\mathcal{P}$  is a proper 4-cycle packing in  $G$  and reduction Rule 4.1 is not applicable on  $G$ , we can get that  $Q_0 = \emptyset$ . We now bound the size of  $Q_3$ .

**Lemma 6.**  $|Q_3| \leq \max\{0, 2|V(\mathcal{P})| - 4\}$ .

In the following, we will bound the size of  $Q_1$  and  $Q_2$ . For any two distinct vertices  $u, v$  in  $\mathcal{P}$ , let  $Q_2(uv) = \{w \in Q_2 \mid N(w) \cap V(\mathcal{P}) = \{u, v\}\}$ . Assume that  $T$  is the set of all non-empty subsets  $Q_2(uv)$  for each distinct vertices  $u, v$  in  $\mathcal{P}$ .

**Lemma 7.** *Each subset in  $T$  has size one.*

**Lemma 8.**  $|T| \leq 3|V(\mathcal{P})| - 6$ .

In graph  $G$ , a path with three vertices and two edges is called a 3-path. For any two 3-paths  $p_1 = (x_1, x_2, x_3)$  and  $p_2 = (y_1, y_2, y_3)$ ,  $p_1$  is called connected to  $p_2$  if  $p_1, p_2$  satisfies the following properties: for any vertex  $x_i$  ( $1 \leq i \leq 3$ ), there exists a unique vertex  $y_j$  in  $p_2$  ( $1 \leq j \leq 3$ ) such that  $x_i y_j$  is an edge in  $G$ .

**Lemma 9.** *For any arbitrary 4-cycle  $C \in \mathcal{P}$ , there exists at most one 3-path in  $G[Q]$  connected to a 3-path of  $C$ .*

For a single edge  $e$  in  $G[Q]$ , we first claim that the two endpoints of edge  $e = uv$  are not both from  $Q_1$ . Assume that  $u, v$  are both from  $Q_1$ , and are connected to an edge  $e' = u'v'$  of 4-cycle  $C$  in  $\mathcal{P}$ . It is easy to see that 4-cycle constructed by vertices  $u, v, u', v'$  is a dangling 4-cycle, which can be handled by Rule 4.4, a contradiction. Thus, we can get that for each single edge  $e$  with one vertex from  $Q_1$  in  $G[Q]$ ,  $e$  is adjacent to the unique edge in a 4-cycle of  $\mathcal{P}$ , and the other endpoints of  $e$  must be from  $Q_2 \cup Q_3$ .

Suppose that  $e_1 = \{a, b\}$  and  $e_2 = \{c, d\}$  are two single edges in  $G[Q]$  which are adjacent to the same edge  $e = \{u, v\}$  of a 4-cycle in  $\mathcal{P}$ , where  $a, c$  are the vertices in  $Q_2 \cup Q_3$ ,  $b, d$  are the vertices in  $Q_1$ . Assume that  $a, c$  are adjacent to  $u$ , and  $b, d$  are adjacent to  $v$ . We first claim that  $e_1$  and  $e_2$  cannot share a vertex. Assume that  $e_1$  and  $e_2$  share a vertex. If  $a = c$ , then a 4-cycle  $\{a, b, v, d\}$  can be found, which is edge-disjoint with the 4-cycles in  $\mathcal{P}$ , contradicting with the maximality of  $\mathcal{P}$ . Other cases of sharing vertices of  $e_1$  and  $e_2$  can be similarly discussed.

Assume that  $e$  is a single edge in  $G[Q]$  which is adjacent to an edge  $e'$  of a 4-cycle in  $\mathcal{P}$ . Let  $u$  be one vertex in  $e$  from  $Q_1$ , and let  $v$  be the other endpoint of  $e$  which is from  $Q_2 \cup Q_3$ . It is not hard to see that vertex  $u$  can be adjacent to exactly one vertex in  $Q_2 \cup Q_3$ , and vertex  $v$  can be adjacent to exactly one vertex of  $Q_1$ . Otherwise, one of reduction rules can be applied again. We now bound the number of vertices in  $Q_1$ .

**Lemma 10.** *The number of vertices in  $Q_1$  is at most  $6|V(\mathcal{P})| + 3k - 12$ .*

*Proof.* It is not hard to get that the vertices in  $Q_1$  might form single edges and 3-paths. By Lemma 9, we get that for any arbitrary 4-cycle  $C \in \mathcal{P}$ , there exists at most one 3-path in  $G[Q]$  that is connected to 3-path of  $C$ . Assume that  $P'$  is a 3-path in  $G[Q]$  that is connected to 3-path of  $C$ . The vertices in  $P'$  might be from  $Q_1$  and  $Q_2 \cup Q_3$ . The vertices in the 3-paths whose vertices are all from  $Q_1$  are bounded by  $3k$ . Thus, the remaining task is to consider the vertices in 3-paths that contain vertices from  $Q_2 \cup Q_3$ , and the vertices in the single edges that contain vertices from  $Q_2 \cup Q_3$ .

Let  $Q'_2$  be a subset of  $Q_2$  such that each vertex in  $Q'_2$  has at least one neighbor in  $Q_1$ , and let  $Q'_3$  be a subset of  $Q_3$  such that each vertex in  $Q'_3$  has at least one neighbor in  $Q_1$ . In order to bound the number of vertices in  $Q_1$ , we first construct an auxiliary graph  $H$  as follows: (1) add vertices  $Q_1 \cup Q'_2 \cup Q'_3 \cup V(\mathcal{P})$  into  $H$ ; (2) for a vertex  $u$  in  $Q_1$  and a vertex  $v$  in  $Q'_2 \cup Q'_3 \cup V(\mathcal{P})$ , if there exists an edge between  $u$  and  $v$  in  $G$ , then add edge  $uv$  into  $H$ . Based on the auxiliary graph  $H$ , another auxiliary graph  $H'$  can be constructed in the following way: (1) add vertices  $Q'_2 \cup Q'_3 \cup V(\mathcal{P})$  into  $H'$ ; (2) for any vertex  $u \in Q'_2 \cup Q'_3$  and any vertex  $v \in V(\mathcal{P})$ , if there exists a vertex  $w$  in  $Q_1$  such that  $uw$  and  $wv$  are the edges in  $H$ , then add edge  $uv$  into  $H'$ . It is easy to see that  $H'$  is a bipartite planar graph and triangle-free, and each vertex in  $Q'_2 \cup Q'_3$  has degree at least three. By the above discussion, for any vertex  $u$  in  $Q'_2 \cup Q'_3$  and any vertex  $v$  in  $V(\mathcal{P})$ ,  $u$  and  $v$  can have at most one common neighbor from  $Q_1$  in  $G$ . Thus, no two vertices in  $H'$  have multiple edges. The number of vertices in  $Q_1$  is exactly the number of edges in  $H'$ . The number of vertices contained in  $Q'_2 \cup Q'_3$  is bounded by  $2|V(\mathcal{P})| - 4$ . Therefore, the number of edges in  $H'$  is bounded by  $2((2|V(\mathcal{P})| - 4) + |V(\mathcal{P})|) - 4 = 6|V(\mathcal{P})| - 12$ . Thus, the total number of vertices in  $Q_1$  is at most  $6|V(\mathcal{P})| + 3k - 12$ .  $\square$

For an isolated vertex  $v$  in  $G[Q]$ , if  $v$  is connected to the vertices of  $C$ , such as  $a, c$  or  $b, d$ , then it is called vertex  $v$  is connected to 4-cycle  $C$ .

**Lemma 11.** *For any arbitrary 4-cycle  $C \in \mathcal{P}$ , if an isolated vertex  $v$  in  $G[Q]$  is connected to  $C$ , then no single edge or 3-path in  $G[Q]$  can be connected to  $C$ . Similarly, if a single edge in  $G[Q]$  is connected to  $C$ , then no isolated vertex or 3-path in  $G[Q]$  can be connected to  $C$ ; if a 3-path in  $G[Q]$  is connected to  $C$ , then no isolated vertex or single edge can be connected to  $C$ .*

**Theorem 3.** *The Parameterized Planar 4-Cycle Transversal problem admits a kernel of at most  $51k - 22$  vertices.*

*Proof.* For the reduced instance  $(G, k)$  of the Parameterized Planar 4-Cycle Transversal problem, the size of  $G$  is bounded by  $|V(\mathcal{P})| + |Q_1| + |Q_2| + |Q_3|$ . The size of  $V(\mathcal{P})$  is bounded by  $4k$ . By Lemma 8, the number of vertices in  $Q_2$  is bounded by  $3|V(\mathcal{P})| - 6$ , and by Lemma 6, the number of vertices in  $Q_3$  is bounded by  $2|V(\mathcal{P})| - 4$ . By Lemma 10, the number of vertices in  $Q_1$  is bounded by  $6|V(\mathcal{P})| + 3k - 12$ . Thus, the total number of vertices in the reduced graph

$G$  is  $|V(G)| = |V(\mathcal{P})| + |Q_1| + |Q_2| + |Q_3| \leq |V(\mathcal{P})| + (6|V(\mathcal{P})| + 3k - 12) + (3|V(\mathcal{P})| - 6) + (2|V(\mathcal{P})| - 4) \leq 51k - 22$ .  $\square$

The kernelization process for the Parameterized Planar 4-Cycle Transversal problem can be applied to the kernelization of the Parameterized Planar Edge-Disjoint 4-Cycle Packing.

**Corollary 1.** *The Parameterized Planar Edge-Disjoint 4-Cycle Packing problem admits a kernel of at most  $51k - 22$  vertices.*

## References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating set. *J. ACM* **51**(3), 363–384 (2004)
2. Alon, N., Bollobás, B., Krivelevich, M., Sudakov, B.: Maximum cuts and judicious partitions in graphs without short cycles. *J. Comb. Theory, Ser. B* **88**(2), 329–346 (2003)
3. Borodin, O.V., Kostochka, A.V., Sheikh, N.N., Yu, G.: M-degrees of quadrangle-free planar graphs. *J. Graph Theory* **60**(1), 80–85 (2009)
4. Brüggmann, D., Komusiewicz, C., Moser, H.: On generating triangle-free graphs. *Electron. Notes Discret. Math.* **32**, 51–58 (2009)
5. Cameron, K.: Induced matchings. *Discret. Appl. Math.* **24**, 97–102 (1989)
6. Cameron, K.: Induced matchings in intersection graphs. *Discret. Math.* **278**(1–3), 1–9 (2004)
7. Cameron, K., Sritharan, R., Tang, Y.: Finding a maximum induced matching in weakly chordal graphs. *Discret. Math.* **266**(1–3), 133–142 (2003)
8. Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems. In: Albers, S., Radzik, T. (eds.) *ESA 2004*. LNCS, vol. 3221, pp. 192–203. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30140-0\\_19](https://doi.org/10.1007/978-3-540-30140-0_19)
9. Duckworth, W., Manlove, D.F., Zito, M.: On the approximability of the maximum induced matching problem. *J. Discret. Algorithms* **3**(1), 79–91 (2005)
10. Erman, R., Kowalik, L., Krnc, M., Waleń, T.: Improved induced matchings in sparse graphs. *Discret. Appl. Math.* **158**, 1994–2003 (2010)
11. Fricke, G., Laskar, R.: String matching in trees. *Congr. Numer.* **89**, 239–243 (1992)
12. Golubic, M.C., Lewenstein, M.: New results on induced matchings. *Discret. Appl. Math.* **101**(1–3), 157–165 (2000)
13. Gotthilf, Z., Lewenstein, M.: Tighter approximations for maximum induced matchings in regular graphs. In: Erlebach, T., Persinao, G. (eds.) *WAOA 2005*. LNCS, vol. 3879, pp. 270–281. Springer, Heidelberg (2006). [https://doi.org/10.1007/11671411\\_21](https://doi.org/10.1007/11671411_21)
14. Guo, J., Niedermeier, R.: Linear problem kernels for NP-Hard problems on planar graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) *ICALP 2007*. LNCS, vol. 4596, pp. 375–386. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73420-8\\_34](https://doi.org/10.1007/978-3-540-73420-8_34)
15. Halford, T.R., Grant, A.J., Chugg, K.M.: Which codes have 4-cycle-free tanner graphs. *IEEE Trans. Inf. Theory* **52**(9), 4219–4223 (2006)
16. Heggenes, P., Hof, P.V., Lokshtanov, D., Paul, C.: Irredundancy in circular arc graphs. *Discret. Appl. Math.* **44**(1–3), 79–89 (1993)

17. Jiang, M., Xia, G., Zhang, Y.: Edge-disjoint packing of stars and cycles. *Theor. Comput. Sci.* **640**, 61–69 (2016)
18. Kanj, I., Pelsmayer, M.J., Schaefer, M., Xia, G.: On the induced matching problem. *J. Comput. Syst. Sci.* **77**, 1058–1070 (2011)
19. Kobler, D., Rotics, U.: Finding maximum induced matchings in subclasses of claw-free and  $P_5$ -free graphs, and in graphs with matching and induced matching of equal maximum size. *Algorithmica* **37**(4), 327–346 (2003)
20. Kortsarz, G., Langberg, M., Nutov, Z.: Approximating maximum subgraphs without short cycles. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX/RANDOM-2008. LNCS, vol. 5171, pp. 118–131. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85363-3\\_10](https://doi.org/10.1007/978-3-540-85363-3_10)
21. Krivelevich, M.: On a conjecture of Tuza about packing and covering of triangles. *Discret. Math.* **142**(1–3), 281–286 (1995)
22. Lovász, L., Plummer, M.D.: *Matching Theory*. North Holland, Amsterdam (1986)
23. Lozin, V.V.: On maximum induced matchings in bipartite graphs. *Inf. Process. Lett.* **81**(1), 7–11 (2002)
24. Lozin, V.V., Rautenbach, D.: Some results on graphs without long induced paths. *Inf. Process. Lett.* **88**(4), 167–171 (2003)
25. Moser, H., Thilikos, D.M.: Parameterized complexity of finding regular induced subgraphs. In: *Proceedings of the Second Workshop on Algorithms and Complexity in Durham*, pp. 107–118 (2006)
26. Moser, H., Sikdar, S.: The parameterized complexity of the induced matching problem. *Discret. Appl. Math.* **157**, 715–727 (2009)
27. Orlovich, Y., Finke, G., Gordon, V., Zverovich, I.: Approximability results for the maximum and minimum maximal induced matching problems. *Discret. Optim.* **5**, 584–593 (2008)
28. Pevzner, P., Tang, H., Tesler, G.: De novo repeat classification and fragment assembly. *Genome Res.* **14**(9), 1786–1796 (2004)
29. Stockmeyer, L.J., Vazirani, V.V.: NP-completeness of some generalizations of the maximum matching problem. *Inf. Process. Lett.* **15**(1), 14–19 (1982)
30. Thomassen, C.: On the chromatic number of triangle-free graphs of large minimum degree. *Combinatorica* **22**(4), 591–596 (2002)
31. Thomassen, C.: On the chromatic number of pentagon-free graphs of large minimum degree. *Combinatorica* **27**(2), 241–243 (2007)
32. Xia, G., Zhang, Y.: Kernelization for cycle transversal problems. In: *Proceedings of AAIM*, pp. 293–303 (2010)
33. Xia, G., Zhang, Y.: On the small cycle transversal of planar graphs. *Theor. Comput. Sci.* **412**(29), 3501–3509 (2011)
34. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: *Proceedings of STOC*, pp. 253–264 (1978)
35. Zhu, J., Bu, Y.: Equitable list colorings of planar graphs without short cycles. *Theor. Comput. Sci.* **407**(1–3), 21–28 (2008)

## **Other Algorithms**





# On Bayesian Epistemology of Myerson Auction

Xiaotie Deng<sup>1(✉)</sup> and Keyu Zhu<sup>2(✉)</sup>

<sup>1</sup> School of Electronics Engineering and Computer Science, Peking University, Science Building, No. 5 Yiheyuan Lu, Haidian District, Beijing 100871, China  
xiaotie@pku.edu.cn

<sup>2</sup> Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, China  
HyperSpaceX@sjtu.edu.cn

**Abstract.** Bayesian Epistemology bases its analysis of the objects under study on a prior, a probability distribution, which is in turn the subject matter in statistical learning, and that of machine learning at least implicitly. We are interested in a game setting where the agents to be learned may shift in accordance with the data collector's strategies. We focus on this issue of learning and exploiting for Myerson auction where a seller wants to gain information on bidders' value distributions to achieve the maximum revenue. We show that a world of the power-law distribution would enable the auctioneer to achieve both but the bidders can consistently lie about their probability distribution to improve utility under the other distributions.

**Keywords:** Statistical learning · Bayesian Epistemology  
Probability distribution cheating · Myerson auction  
Bidding game Nash equilibrium

## 1 Introduction

The Bayesian Epistemology has been widely applied to keep our logic thinking in terms of probability consistent, and hence has been adopted to deal with data in presence of random perturbations of various kinds. It has become one of the most useful data analytic tools, and expectably will play a much more important role in today's era of big data. On the other hand, the coming of a huge amount of data creates the conditions for the law of large numbers to hold, and hence derives increasingly accurate characterizations of the world for statistical learning, and more generally machine learning, to perform better and better. The acclaimed “the end of theory” [1] advocated a future for petabyte data and its technologies to advance sciences “even without coherent models, unified theories, or really any mechanistic explanation at all”, with a new empiricism where data may speak on their own, free of theory [8].

---

Research results reported in this work are partially supported by the National Natural Science Foundation of China (Grant Nos. 61632017, 61173011).

In guarding the tradition of mathematical and logic rigor in economics, Debreu suggested that “Being denied a sufficiently secure experimental base, economic theory has to adhere to the rules of logical discourse and must renounce the facility of internal inconsistency” [2]. The coming of the big data era, with information and communication technology (ICT) enabled economical activities, has brought in the possibility to connect theoretical understanding with reality especially for electronically conducted transactions and services. Economics theories, at the level of microeconomics, are under re-examination to be applied to the highly electronic market based environment. As an example, the generalized second price auction designed for the sponsored search market to sell billions of “clicks” has received intensively studies in recent years [5]. Here transactions on selling each click can be completely recorded and fully analyzed to test the exact level of matching between theory and economic data in this and other similar cases.

Bayesian statistics has been the key link in reconciling the gap between the probability calculus as a reasoning methodology and the data foundation for empirical relevance to reality. It uses parameter modeling to create a prior of the uncertain world based on empirical data, then applies the model to make predictions about future, turning new data to potential outcomes. This methodology has received warm welcome by both side of the methodology. Practicians designed models, parameter fitted on training data, verified on testing data, applied their models to make predictions and to decide operations in the real world. Theorists have derived new concepts and developed new branches of scientific investigations. One of the most interesting work is that of revelation principle and maximum revenue auction, commonly known as the Myerson auction [9]. The Myerson auction assures that, given that bidders’ value (or prior) distributions are common knowledge [6], the seller achieves the optimal expected revenue among all truthful auction protocols. Here the priors of the bidders are assumed to be known to all and this theorem has focused on the second half of the methodology of Bayesian Statistics. In an effort to implement Myerson auction, it would be an immediate task to complement the theorem to find the priors for the whole process to function.

We are interested to take this example of Myerson auction to develop an understanding whether there can be a consistent view to merge the two stages of Bayesian statistical learning process to have a comprehensive Bayesian epistemic methodology under this setting. We find out interesting probability distributions for which this is possible for full implementation of Myerson auction, and others for which deviation from the seller’s theoretical optimum is inevitable. Our discovery establishes a rather limited scenario where learning can be done at the same time of optimal pricing but more broadly opens up an issue whether and under what scenario we are able learn and exploit the learning outcome.

**Our Model and Results:** We consider a Myerson auction among  $n$  bidders with independent true prior distributions. They have the right to make bids following distributions not necessarily identical to their true prior distributions. Assume that their true prior and bidding distributions belong to a specific class

of distributions. Then we figure out that for the class of power-law distributions, bidders stick to following their true prior distributions. This leads to the best scenario for the auctioneer to implement the Myerson auction: the bidders will follow their true prior distributions and bid true values of the item simultaneously. For half-normal distribution class, bidders have the incentive to deviate from their true prior distributions and follow distributions with smaller variance. Similar results are found under the exponential distribution assumption.

**Related Works:** The misreporting distribution game under Myerson auction has received interests from different points of views recently. For independent and identical prior distributions, the misreporting game's equilibrium computation issue was studied in [3]. [10] derived a classic type revenue equivalence property of the fake distribution game which allows players' best response strategies to be selected arbitrary distributions using a nice technical tool to use the quantile space in the bidding language. In this work our interest is to question the potential of a complete Bayesian epistemic approach in the practicality of Myerson auction, and show so for the power-law distribution. As the power-law distribution is the most common in economic value distributions [7], the results for Myerson auction's validity in this case bring in a possibility to include the prior finding component to the Bayesian Epistemology.

**Presentation Structure:** We present essential preliminary work in Sect. 2. In Sect. 3, we formally introduce our main model and discuss three classes of distributions in the three subsections.

## 2 Preliminaries

We will use the terms “bidder” and “agent” interchangeably throughout the paper. Consider the following auction environment: there is one auctioneer, who has one unit of item for sale. Let  $N$  be the set of agents participating in the auction and  $|N| = n$ . Each agent has a value (which is also referred to by the use of type in the literatures) to this item, generated from some prior distribution. Let vector  $t = (t_1, \dots, t_n)$  denote their bids in the auction, and let  $f_i(\cdot)$  represent the density function of agent  $i$ 's prior value distribution (or rather behaved distribution by consistent biddings) while  $F_i(\cdot)$  represents the corresponding cumulative density function of agent  $i$ . To distinguish the true information from the reported information, we apply a bar notation  $\bar{\cdot}$  to highlight the true value (or prior). In this paper, we always assume that the true prior distributions across the agents are independent. The auctioneer then proposes an auction consisting of an allocation rule and a payment rule to sell this item. Basically, a mechanism will take each bidder's bid ( $t_i$ ) as the input, to produce the output assignment probability for each bidder to be the winner,  $Q(t) = (q_1(t), q_2(t), \dots, q_n(t))$ , the payment charged on all the bidders,  $p(t) = (p_1(t), p_2(t), \dots, p_n(t))$ .

Myerson auction [9] specifies allocation rule  $Q(t)$  and payment rule  $p(t)$ , based on a crucial concept called the virtual value.

**Definition 1.** For any probability density function  $f_i(\cdot)$  for agent  $i$ ,  $\forall i \in N$ , its **virtual valuation function** [9] is defined as  $\phi_i(t_i) = t_i - \frac{1-F_i(t_i)}{f_i(t_i)}$ , where  $F_i(\cdot)$  is the cumulative distribution function with respect to  $f_i(\cdot)$ .

Myerson auction basically offers the item to the agent with the highest virtual value and charges a price that the minimum possible type the winner would bid to win this item. More precisely, let  $c_i(t_{-i})$  denote infimum of all winning bids for each agent  $i$  against the others' bids  $t_{-i}$ , given as follows

$$c_i(t_{-i}) = \inf\{t_i : \phi_i(t_i) \geq 0, \phi_i(t_i) \geq \phi_j(t_j), \forall j \neq i\}.$$

Thus the allocation and pricing rules,  $Q_i(t)$  and  $p_i(t)$  are shown below

$$Q_i(t_{-i}, t_i) = \begin{cases} 1 & t_i > c_i(t_{-i}), \\ 0 & t_i < c_i(t_{-i}). \end{cases}$$

$$p_i(t) = \begin{cases} c_i(t_{-i}) & Q_i(t) = 1, \\ 0 & Q_i(t) = 0. \end{cases}$$

Ties are broken randomly by assigning each bidder with the equal winning probability and its associated payment (which may be different from one to another, dependent on their distributions).

## 2.1 Myerson Auction Against Strategic Bidding Distributions

In the repeated statistical learning process for repeated auctions, the auctioneer strives to achieve its double goals of learning bidders' value distributions and making use of them to achieve Myerson optimal auction. Our bidding language requires the submission of two terms from the agents in  $N$ , probability distributions and the agents' bids for the item. As in statistical learning, the classes of probability distributions under consideration are characterized by their corresponding parameters. We focus on several classes of parameter distributions, including power-law, half-normal and exponential distributions. An agent is assumed to have the same probability distribution in the repeated auction process but may have different true values at each round.

Upon receiving the parameter distributions and the bids from all bidders, the auctioneer consistently applies the allocation and payment rule according to the Myerson auction with respect to the submitted distributions. Arguably, the consistency in reporting the underlying value distribution can be guaranteed by the learning effort through the repeated auctions. We study how well the Myerson auction can be launched under this setting, namely, study its profit guarantees under the equilibrium of bidders' strategies, as defined subsequently.

**Definition 2.** For a class of priors with parameters, the **strategy set** can be denoted by  $S = \times_{i=1}^n S_i$  which consists of individual strategy set

$$S_i = \{f_i(\cdot) \mid f_i(\cdot) \text{ is a probability density function}\}$$

for any  $i \in [n]$ , where  $f_i$  is encoded by parameters.

This strategy set is all we know about partial information for everyone’s true prior, and we are interested in understanding the power of Myerson auction in this case.

We quantify agent strategic behavior via the standard definition of utility as follows:

**Definition 3.** For each agent  $i$ , the **expected utility**  $U_i(f_1, \dots, f_n)$  with respect to the strategy  $(f_1(\cdot), \dots, f_n(\cdot)) \in S$  is defined as

$$\int_{\mathbb{R}^n} \left( \prod_{i=1}^n f_i(t_i) \right) (Q(t)\bar{t}(t) - p(t)) dt,$$

where  $\bar{t}(t)$  is bidder  $i$ ’s true value of the good for sale correspondent to his/her bid for this good. (In the latter sections, we introduce mapping criteria to quantify this relationship between bids and true values.)

**Definition 4.** A particular choice of strategies is a **Nash equilibrium** if and only if for each agent  $i$ , there is no incentive for him or her changing the strategy for a higher expected utility  $U_i$  supposing that the others’ strategies remain unchanged. That is to say,  $(f_1(\cdot), \dots, f_n(\cdot)) \in S$  is a Nash equilibrium if and only if  $\forall g_i \in S_i$  and  $i \in [n]$ ,

$$U_i(f_1, \dots, f_n) \geq U_i(f_1, \dots, f_{i-1}, g_i, f_{i+1}, \dots, f_n).$$

**Definition 5.** A mechanism is **dominant strategy incentive compatible (DSIC)** if and only if each bidder’s expected utility is optimized by making bids according to his or her true prior distribution characterized by  $f_i$  no matter what others’ value distributions their bids follow. In other words,  $\forall i \in [n]$  and  $(f_1(\cdot), \dots, f_n(\cdot)) \in S$ ,

$$U_i(f_1, \dots, f_{i-1}, \bar{f}_i, f_{i+1}, \dots, f_n) \geq U_i(f_1, \dots, f_n).$$

**Definition 6.** A distribution is **regular** if and only if its virtual valuation function is monotonically increasing.

**Definition 7.** For a regular distribution, the **reserve price** is  $\phi^{-1}(0)$ , where  $\phi$  is the corresponding virtual valuation function.

### 3 Main Model

#### 3.1 Power-Law Distributions

To begin with, we constrain individual strategy sets by presuming that prior distributions of all of the bidders, including their true prior distributions, belong to a subordinate category of power-law distributions. In other words,  $S_i(\alpha, \beta) = \{g_i(\cdot) \mid g_i(t) = \mathbb{1}_{\left[\left(\frac{\beta}{\alpha-1}\right)^{\frac{1}{\alpha-1}}, +\infty\right)} \beta t^{-\alpha}, \forall \alpha > 1, \beta > 0\}$ . Here  $\alpha$  is observed to be a constant number in the interval  $(2, 3)$  in many application problems such as wages, city size and key parameters in complex network [7]. Since  $\alpha$  is a fixed value, individual strategy set  $S_i(\alpha, \beta)$  consisting of functions with the single parameter  $\beta > 0$  and can be simplified as  $S_i(\beta)$  without any ambiguity.

**Theorem 1.** *Our mechanism is DSIC if each bidder  $i$  bids according to the power-law distribution characterized by  $\beta_i$  with true prior distribution characterized by  $\bar{\beta}_i$ . It implies that the unique Nash equilibrium happens at  $(\bar{\beta}_1, \dots, \bar{\beta}_n)$ .*

Detailed proofs can be found in [3] (Sect. 3). The scheme of the proof is that the auctioneer implements Myerson auction with respect to reported prior distributions from agents and each agent aims to maximize their expected utility by selecting a strategy, *i.e.*, a prior distribution to report. Then we analyze each agent’s behavior according to its incentive. One thing to notice is that calculating expected utility is not straightforward since we need to map winning bid from reported prior distribution to true value of true prior distribution with a technique, *Mapping Criterion*. In the following two sections, we consider two different distribution classes under this framework of the proof.

### 3.2 Half-Normal Distributions

In this section, we constrain individual strategy sets by assuming that prior distributions of all of the bidders, including their true prior distributions, belong to the class of half-normal distributions. In other words,  $S(\sigma) = \{g(\cdot) \mid g(t) = \mathbb{1}_{[0,+\infty)} \frac{\sqrt{2}}{\sqrt{\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}}, \forall \sigma > 0\}$ . Naturally, the aforementioned virtual valuation function of  $S(\sigma)$  can be given by the following formula

$$\phi_\sigma(t) = t - \frac{1 - \int_0^t \frac{\sqrt{2}}{\sqrt{\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} dx}{\frac{\sqrt{2}}{\sqrt{\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}}}, \quad \forall t \geq 0.$$

Then we move on to prove two lemmas in order to explore several desirable properties of the formula above.

**Lemma 1 (Regularity).** *Every half-normal distribution characterized by  $\sigma > 0$  is regular.*

*Proof.* With Definition 6, we merely need to verify that  $\phi_\sigma(t)$  is an increasing function when  $t$  is non-negative. Take the derivative and we have

$$\begin{aligned} \phi'_\sigma(t) &= 2 - \frac{\left(1 - \int_0^t \frac{\sqrt{2}}{\sqrt{\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} dx\right) \frac{t}{\sigma^2}}{\frac{\sqrt{2}}{\sqrt{\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}}} = 2 - \frac{t}{\sigma^2} \int_t^{+\infty} e^{-\frac{t^2-x^2}{2\sigma^2}} dx \\ &\geq 2 - \frac{t}{\sigma^2} \int_t^{+\infty} e^{-\frac{t}{\sigma^2}(t-x)} dx = 2 - 1 = 1 > 0. \end{aligned}$$

Since the first derivative of the virtual valuation function  $\phi_\sigma(t)$  is strictly positive for any  $t \geq 0$ , we claim that every half-normal distribution is regular.

Moreover, we figure out that  $e^{\frac{t^2}{2\sigma^2}} \int_t^{+\infty} e^{-\frac{x^2}{2\sigma^2}} dx < \frac{\sigma^2}{t}$ , for any  $\sigma > 0$  and  $t > 0$ . This inequality offers an upper bound which we employ to our proof below. In addition to monotonicity, we introduce another lemma to explore an underlying property of graphs of virtual valuation functions with different  $\sigma > 0$ .

**Lemma 2.**  $\phi_{\sigma_1}(t) > \phi_{\sigma_2}(t), \forall \sigma_2 > \sigma_1 > 0, t \geq 0$ .

*Proof.* Let

$$m(t) = \frac{t}{\sqrt{2\pi}} + e^{\frac{t^2}{2}}(1 - \Phi(t))(1 - t^2), \quad \forall t \geq 0,$$

where  $\Phi(t) = \int_{-\infty}^t \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ . We continue to prove the function  $m(x)$  is positive for any  $x \geq 0$  and its limit is 0 as  $x$  approaches plus infinity. Upper and lower tail bounds of standard normal distributions enable us to complete a relatively elegant proof.

$$\frac{t}{t^2 + 1} < \int_t^{+\infty} e^{\frac{t^2 - x^2}{2}} dx = \sqrt{2\pi} e^{\frac{t^2}{2}}(1 - \Phi(t)) < \frac{1}{t}, \quad \forall t > 0.$$

The upper bound is not difficult to obtain since we recall the previous inequality  $e^{\frac{t^2}{2}} \int_t^{+\infty} e^{-\frac{x^2}{2}} dx < \frac{\sigma^2}{t}$  and take  $\sigma = 1$ . Let

$$\Delta(t) = \int_t^{+\infty} e^{\frac{t^2 - x^2}{2}} dx - \frac{t}{t^2 + 1}.$$

Since  $0 \leq \lim_{t \rightarrow +\infty} \int_t^{+\infty} e^{\frac{t^2 - x^2}{2}} dx \leq \lim_{t \rightarrow +\infty} \frac{1}{t} = 0$ , we claim that  $\lim_{t \rightarrow +\infty} \Delta(t) = 0$ . Additionally, the first derivative  $\Delta'(t) = -e^{-\frac{t^2}{2}} \frac{2\sqrt{2\pi}}{(t^2 + 1)^2}$  is strictly negative for any  $t > 0$ . Due to the fact that  $\Delta(0) = \frac{\sqrt{2\pi}}{2} > 0$ , the function  $\Delta(t)$  is positive, which actually completes our proof of the lower bound.

Therefore,  $m'(t) = \frac{t^2}{\sqrt{2\pi}} - e^{\frac{t^2}{2}}(1 - \Phi(t))(t^3 + t)$  is strictly negative because of the newly established lower bound. Also, we can estimate the limit of  $m(t)$  as  $t$  approaches plus infinity.

$$\frac{2t}{\sqrt{2\pi}(t^2 + 1)} < m(t) < \frac{1}{\sqrt{2\pi}t}, \forall t > 0.$$

With the squeeze theorem, the limit of  $m(t)$  exists and is equal to 0. Note that the function  $m(t)$  is strictly decreasing with its limit 0 and  $m(0) = \frac{1}{2} > 0$ . Thus,  $m(t)$  is strictly positive, whatever the non-negative  $t$ . Let

$$M(t, \sigma) = \sigma e^{\frac{t^2}{2\sigma^2}} \left( 1 - \Phi\left(\frac{t}{\sigma}\right) \right).$$

For any fixed  $t \geq 0$ , we are supposed to verify that the function  $M(t, \sigma)$  is strictly increasing with respect to the variable  $\sigma$ . Take the partial derivative and we obtain

$$\frac{\partial M(t, \sigma)}{\partial \sigma} = m\left(\frac{t}{\sigma}\right) > 0, \forall \sigma > 0.$$

And

$$\phi_{\sigma_1}(t) - \phi_{\sigma_2}(t) = \sqrt{2\pi}(M(t, \sigma_2) - M(t, \sigma_1)) > 0,$$

whenever  $\sigma_2 > \sigma_1 > 0$  and  $t \geq 0$  because of the property of the monotonic function  $M(t, \sigma)$ .

**Corollary 1.**  $\frac{\partial \phi_\sigma^{-1}(t)}{\partial \sigma} \geq 0, \forall \sigma > 0, t > \phi_\sigma(0).$

*Proof.* Let  $\tau > \sigma$ . The inequality  $\phi_\sigma(t) > \phi_\tau(t)$  holds due to Lemma 2 and there exists a point  $T > t$  such that  $\phi_\tau(T) = \phi_\sigma(t)$  because of Lemma 1. Therefore, we obtain

$$\frac{\partial \phi_\sigma^{-1}(t)}{\partial \sigma} = \lim_{\tau \rightarrow \sigma^+} \frac{T - t}{\tau - \sigma} \geq 0.$$

The bidder is unlikely to win if the virtual value of his or her bid is less than the reserve price. Without loss of generality, let  $r(\sigma)$  denote the reserve price with respect to the half-normal distribution characterized by the single parameter  $\sigma$ .

**Proposition 1.**  $\sigma \phi_1^{-1}(\frac{t}{\sigma}) = \phi_\sigma^{-1}(t), \forall t \geq 0, \sigma > 0.$

$$t = \phi_\sigma(s) = s - \frac{1 - \int_0^s \frac{\sqrt{2}}{\sqrt{\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}} dx}{\frac{\sqrt{2}}{\sqrt{\pi\sigma}} e^{-\frac{s^2}{2\sigma^2}}},$$

$$\frac{t}{\sigma} = \phi_1\left(\frac{s}{\sigma}\right) = \frac{s}{\sigma} - \frac{1 - \int_0^{\frac{s}{\sigma}} \frac{\sqrt{2}}{\sqrt{\pi}} e^{-\frac{x^2}{2}} dx}{\frac{\sqrt{2}}{\sqrt{\pi}} e^{-\frac{s^2}{2\sigma^2}}}.$$

Thus,  $\sigma \phi_1^{-1}(\frac{t}{\sigma}) = \phi_\sigma^{-1}(t)$  because of Lemma 1.

**Corollary 2 (Linearity of Reserve Price).**  $r(\sigma)$  is a linear function. In other words,  $r(\sigma) = \sigma r(1), \forall \sigma > 0.$

*Proof.* It is clear that  $r(\sigma) = \phi_\sigma^{-1}(0)$ . According to Proposition 1, we have  $r(\sigma) = \sigma r(1)$  for any  $\sigma > 0$ .

**Proposition 2 (Mapping Criterion).** *There exists a unique mapping between two half-normal distributions from  $S(\sigma)$ , characterized by  $\sigma_1, \sigma_2$  respectively, preserving  $p$ -fractile,  $\forall p \in [0, 1]$ . To put it differently,  $\pi : t_1 \rightarrow t_2 = \frac{\sigma_2}{\sigma_1} t_1$ , subject to*

$$\int_{t_1}^{+\infty} \frac{\sqrt{2}}{\sqrt{\pi\sigma_1}} e^{-\frac{t^2}{2\sigma_1^2}} = \int_{t_2}^{+\infty} \frac{\sqrt{2}}{\sqrt{\pi\sigma_2}} e^{-\frac{t^2}{2\sigma_2^2}}.$$

*Remark 1.* We can calculate bidders' true values of the item for sale with this criterion as long as we obtain private information on their true prior distributions on which we base our work on expected utility of each bidder.

Then we start our analysis of bidders' strategies by discussing a relatively simple scenario where only one bidder gets involved in the auction.

**Warm-Up: Single Bidder**

**Theorem 2.** *Suppose that the bidder's true value follows a true prior distribution  $f(\bar{t}) = \mathbb{1}_{[0,+\infty)} \frac{2}{\sqrt{2\pi\sigma}} e^{-\frac{\bar{t}^2}{2\sigma^2}}$ . When he or she is allowed to make bids according to another half-normal distribution characterized by  $\sigma$ , it is the dominant strategy for the bidder to reduce  $\sigma$  to 0 in terms of the bidder's expected utility, whatever the true prior distribution.*



*Proof.* The expected utility of the bidder can be given by the following formula

$$\begin{aligned}
 U(\sigma, \bar{\sigma}) &= \int_{r(\sigma)}^{+\infty} \left( \frac{\bar{\sigma}}{\sigma} t - r(\sigma) \right) \frac{2}{\sqrt{2\pi\sigma}} e^{-\frac{t^2}{2\sigma^2}} dt \\
 &\stackrel{x=\frac{t}{\sigma}}{=} \int_{r(1)}^{+\infty} (\bar{\sigma}x - \sigma r(1)) \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx.
 \end{aligned}$$

For any fixed  $\bar{\sigma} > 0$ , the expected utility function is a linear function with respect to the parameter  $\sigma$  with a constant negative slope  $-\int_{r(1)}^{+\infty} r(1) \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ . Thus, the bidder is motivated to select a half-normal distribution with a sufficient small parameter  $\sigma$ . Note that the optimal strategy here is exactly a Dirac delta function

$$\delta_\sigma(t) = \lim_{\sigma \rightarrow 0^+} \mathbb{1}_{[0, +\infty)} \frac{2}{\sqrt{2\pi\sigma}} e^{-\frac{t^2}{2\sigma^2}}.$$

### Multiple Bidders

**Theorem 3.** *Suppose that  $n$  bidders participate in the auction ( $n \geq 2$ ). For bidder  $i$ , his or her true value follows a true prior distribution  $f(\bar{t}) = \mathbb{1}_{[0, +\infty)} \frac{2}{\sqrt{2\pi\bar{\sigma}_i}} e^{-\frac{\bar{t}^2}{2\bar{\sigma}_i^2}}$ . When he or she is allowed to make bids according to another half-normal distribution characterized by  $\sigma_i$ , bidder  $i$  always has an incentive to lower the value of  $\sigma_i$  if  $\sigma_i \geq \bar{\sigma}_i$ , whatever the true prior distribution and bidder  $j$ 's strategy,  $\forall j \neq i$ .*

*Proof.* Before establishing main results in this section, we move on to introduce several notations for the convenience of our proof. Let  $f_j(t_j) = \mathbb{1}_{[0, +\infty)} \frac{2}{\sqrt{2\pi\sigma_j}} e^{-\frac{t_j^2}{2\sigma_j^2}}$  denote the distribution which bidder  $j$ 's bids follow,  $\forall j \in [n]$ . Consider two separate regions of integration,

$$\begin{aligned}
 D_1(\sigma_i, \sigma_{-i}) &= \{t \mid t_i \geq r(\sigma_i), 0 \leq \max_{j \neq i} \phi_{\sigma_j}(t_j) \leq \phi_{\sigma_i}(t_i)\}, \\
 D_2(\sigma_i, \sigma_{-i}) &= [r(\sigma_i), +\infty) \times \prod_{j \neq i} [0, r(\sigma_j)].
 \end{aligned}$$

We leverage the notation  $X_{(n-1), -i}$  slightly different from previous sections. Here,  $X_{(n-1), -i}$  is considered as the largest observation among all these  $n$  random variables except  $X_i$ , where  $X_i$  is a random variable of each bidder  $i$ 's virtual values with respect to bids. Consider the expected utility function

$$U_i(\sigma, \bar{\sigma}_i) = \int_{t: \phi_{\sigma_i}(t_i) \geq \max\{\phi_{\sigma_j}(t_j), 0\}, \forall j \neq i} (\bar{t}_i(t_i) - p_i(t)) f_i(t) dt,$$

where  $\sigma = (\sigma_1, \dots, \sigma_n)$ . The region of integration is actually the combination of  $D_1(\sigma_i, \sigma_{-i})$  and  $D_2(\sigma_i, \sigma_{-i})$ . With order statistics, we can further simplify the integration over the region  $D_1(\sigma_i, \sigma_{-i})$  as follows

$$\int_0^{+\infty} f_{X_{(n-1), -i}}(t_{-i}) dt_{-i} \int_{\phi_{\sigma_i}^{-1}(t_{-i})}^{+\infty} \left( \frac{\bar{\sigma}_i t_i}{\sigma_i} - \phi_{\sigma_i}^{-1}(t_{-i}) \right) f_i(t_i) dt_i,$$

where

$$f_{X_{(n-1),-i}}(t_{-i})dt_{-i} = d \left( \prod_{j \neq i} \int_0^{\phi_{\sigma_j}^{-1}(t_{-i})} f_j(t_j)dt_j \right).$$

Similarly, the integration over the the region  $D_2(\sigma_i, \sigma_{-i})$  can be given by the following formula

$$\prod_{j \neq i} \left( \int_0^{r(\sigma_j)} f_j(t_j)dt_j \right) \int_{r(\sigma_i)}^{+\infty} \left( \frac{\bar{\sigma}_i t_i}{\sigma_i} - r(\sigma_i) \right) f_i(t_i)dt_i.$$

We split our discussion into two parts with respect to regions,  $D_1(\sigma_i, \sigma_{-i})$  and  $D_2(\sigma_i, \sigma_{-i})$ . First, we calculate the expected utility  $U_i(\sigma_i, \sigma_{-i})|_{D_1}$  over the fixed region  $D_1(\sigma_i, \sigma_{-i})$  as follows

$$\int_0^{+\infty} f_{X_{(n-1),-i}}(t_{-i})dt_{-i} \int_{\phi_{\sigma_i}^{-1}(t_{-i})}^{+\infty} \left( \frac{\bar{\sigma}_i t_i}{\sigma_i} - \phi_{\sigma_i}^{-1}(t_{-i}) \right) f_i(t_i)dt_i.$$

For the convenience of our proof, we normalize the formula above into the following one

$$\int_0^{+\infty} f_{X_{(n-1),-i}}(t_{-i})dt_{-i} \int_{\phi_1^{-1}(\frac{t_{-i}}{\sigma_i})}^{+\infty} (\bar{\sigma}_i x - \phi_{\sigma_i}^{-1}(t_{-i})) f(x)dx,$$

where  $f(x)$  is the probability density function of the standard half-normal distribution. We wonder what difference the deviation from  $\sigma_i$  makes to the partial expected utility of bidder  $i$ . Thus, we take the derivative of the parameter  $\sigma_i$  and have

$$\begin{aligned} \frac{\partial U_i(\sigma_i, \sigma_{-i})|_{D_1}}{\partial \sigma_i} &= - \int_0^{+\infty} f_{X_{(n-1),-i}}(t_{-i})dt_{-i} \left[ \int_{\phi_1^{-1}(\frac{t_{-i}}{\sigma_i})}^{+\infty} \frac{\partial \phi_{\sigma_i}^{-1}(t_{-i})}{\partial \sigma_i} f(x)dx \right. \\ &\quad \left. + \frac{\partial \phi_1^{-1}(\frac{t_{-i}}{\sigma_i})}{\partial \sigma_i} \frac{\bar{\sigma}_i - \sigma_i}{\sigma_i} \phi_{\sigma_i}^{-1}(t_{-i}) f \left( \phi_1^{-1}(\frac{t_{-i}}{\sigma_i}) \right) \right] \\ &\leq 0, \quad \forall \sigma_i \geq \bar{\sigma}_i > 0, \end{aligned}$$

where we apply the inequality in Corollary 1. Then we take a look at the second region  $D_2(\sigma_i, \sigma_{-i})$  and obtain similar results.

$$\begin{aligned} U_i(\sigma_i, \sigma_{-i})|_{D_2} &= \prod_{j \neq i} \left( \int_0^{r(\sigma_j)} f_j(t_j)dt_j \right) \int_{r(\sigma_i)}^{+\infty} \left( \frac{\bar{\sigma}_i t_i}{\sigma_i} - r(\sigma_i) \right) f_i(t_i)dt_i \\ &= \prod_{j \neq i} \left( \int_0^{r(\sigma_j)} f_j(t_j)dt_j \right) \int_{r(1)}^{+\infty} (\bar{\sigma}_i x - r(\sigma_i)) f(x)dx. \end{aligned}$$

Take the partial derivative of  $\sigma_i$  and we figure out that

$$\frac{\partial U_i(\sigma_i, \sigma_{-i})|_{D_2}}{\partial \sigma_i} = - \prod_{j \neq i} \left( \int_0^{r(\sigma_j)} f_j(t_j)dt_j \right) \int_{r(1)}^{+\infty} r(1) f(x)dx < 0, \quad \forall \sigma_i \geq \bar{\sigma}_i > 0.$$

Combining these two parts, we have

$$\frac{\partial U_i(\sigma_i, \sigma_{-i})}{\partial \sigma_i} = \frac{\partial U_i(\sigma_i, \sigma_{-i}) |_{D_1}}{\partial \sigma_i} + \frac{\partial U_i(\sigma_i, \sigma_{-i}) |_{D_2}}{\partial \sigma_i} < 0, \quad \forall \sigma_i \geq \bar{\sigma}_i > 0.$$

It means that for each bidder  $i$ , he or she is motivated to make bids following a half-normal distribution characterized by  $\sigma_i$  which is strictly less than  $\bar{\sigma}_i$ .

**Theorem 4.** *Nash equilibrium happens at  $(\sigma_1, \dots, \sigma_n)$  where  $\sigma_i < \bar{\sigma}_i$  for any  $i \leq n$  if Nash equilibrium exists under this setting.*

*Proof.* According to Theorem 3, each bidder always has an incentive to deviate from his or her true prior distribution by lowering the value of  $\sigma_i$  whenever  $\sigma_i \geq \bar{\sigma}_i$ . This result indicates that Nash equilibrium can only be achieved where  $\sigma_i$  is strictly less than  $\bar{\sigma}_i$ .

### 3.3 Exponential Distribution

In this section, we consider the class of exponential distributions and obtain results similar to the previous section of half-normal distributions. We specify individual strategy sets as  $S(\lambda) = \{g(\cdot) \mid g(t) = \mathbb{1}_{[0,+\infty)}\lambda e^{-\lambda t}\}$  and express the virtual valuation function characterized by the single parameter  $\lambda$  using the following formula

$$\phi_\lambda(t) = t - \frac{1 - \int_0^t \lambda e^{-\lambda x} dx}{\lambda e^{-\lambda t}} = t - \frac{1}{\lambda}.$$

Apparently, the inequalities in Lemmas 1 and 2 still hold with  $\lambda$  substituted for  $\sigma$  since the function  $\phi_\lambda$  is a linear function with a constant slope being 1, whatever the value of the parameter  $\lambda$ . Furthermore, the mapping criterion here is specified as  $\pi : t_1 \rightarrow t_2 = \frac{\lambda_1}{\lambda_2}t_1$  between two exponential distributions characterized by  $\lambda_1$  and  $\lambda_2$  respectively. Then we move on to consider the case of multiple bidders involved in the auction just as we analyze the last case.

### Multiple Bidders

**Theorem 5.** *Suppose that  $n$  bidders participate in the auction ( $n \geq 2$ ). For bidder  $i$ , his or her true value follows a true prior distribution  $f(\bar{t}) = \mathbb{1}_{[0,+\infty)}\bar{\lambda}_i e^{-\bar{\lambda}_i \bar{t}}$ . When he or she is allowed to make bids according to another exponential distribution characterized by  $\lambda_i$ , bidder  $i$  always has an incentive to increase the value of  $\lambda_i$  if  $\lambda_i \leq \bar{\lambda}_i$ , whatever the true prior distribution and bidder  $j$ 's strategy,  $\forall j \neq i$ .*

*Proof.* Let  $f_j(t_j) = \mathbb{1}_{[0,+\infty)}\lambda_j e^{-\lambda_j t_j}$  denote the distribution bidder  $j$ 's bids follow for any  $j \in [n]$ . Consider the expected utility function in this case

$$U_i(\lambda, \bar{\lambda}_i) = \int_{t: \phi_{\lambda_i}(t_i) \geq \max\{\phi_{\lambda_j}(t_j), 0\}, \forall j \neq i} (\bar{t}_i(t_i) - p_i(t)) f(t) dt.$$

Note that the region of integration of the formula above consists of two almost disjoint areas

$$D_1(\lambda_i, \lambda_{-i}) = \left\{ t \mid t_i \geq \frac{1}{\lambda_i}, 0 \leq \max_{j \neq i} \phi_{\lambda_j}(t_j) \leq \phi_{\lambda_i}(t_i) \right\},$$

$$D_2(\lambda_i, \lambda_{-i}) = \left[ \frac{1}{\lambda_i}, +\infty \right) \times \prod_{j \neq i} \left[ 0, \frac{1}{\lambda_j} \right].$$

The integration over the region  $D_1$  can be transformed into the following one

$$\int_0^{+\infty} f_{X_{(n-1),-i}}(t_{-i}) dt_{-i} \int_{t_{-i} + \frac{1}{\lambda_i}}^{+\infty} \left( \frac{\lambda_i}{\lambda_i} t_i - t_{-i} - \frac{1}{\lambda_i} \right) f_i(t_i) dt_i,$$

where we introduce the concept of  $f_{X_{(n-1),-i}}$  defined before. Likewise, we obtain the integration over the region  $D_2(\lambda_i, \lambda_{-i})$  as follows

$$\prod_{j \neq i} \int_0^{\frac{1}{\lambda_j}} f_j(t_j) dt_j \int_{\frac{1}{\lambda_i}}^{+\infty} \left( \frac{\lambda_i}{\lambda_i} t_i - \frac{1}{\lambda_i} \right) f_i(t_i) dt_i = \prod_{j \neq i} \int_0^{\frac{1}{\lambda_j}} f_j(t_j) dt_j \int_1^{+\infty} \left( \frac{x}{\lambda_i} - \frac{1}{\lambda_i} \right) f(x) dx,$$

where  $f(x) = \mathbb{1}_{[0,+\infty)} e^{-x}$ . Take the partial derivative of the parameter  $\lambda_i$  respectively, we obtain

$$\begin{aligned} & \frac{\partial U_i(\lambda_i, \lambda_{-i})}{\partial \lambda_i} \Big|_{D_1} \\ &= \int_0^{+\infty} f_{X_{(n-1),-i}}(t_{-i}) dt_{-i} \left[ \int_{\lambda_i t_{-i} + 1}^{+\infty} \frac{1}{\lambda_i^2} f(x) dx - t_{-i} \left( \frac{\lambda_i}{\lambda_i} - 1 \right) \left( t_{-i} + \frac{1}{\lambda_i} \right) f(\lambda_i t_{-i} + 1) \right] \\ &\geq 0, \quad \forall \lambda_i \in (0, \bar{\lambda}_i], \end{aligned}$$

$$\frac{\partial U_i(\lambda_i, \lambda_{-i})}{\partial \lambda_i} \Big|_{D_2} = \prod_{j \neq i} \int_0^{\frac{1}{\lambda_j}} f_j(t_j) dt_j \int_1^{+\infty} \frac{1}{\lambda_i^2} f(x) dx > 0, \quad \forall \lambda_i \in (0, \bar{\lambda}_i].$$

Thus, combining these two parts, we complete our proof of Theorem 5

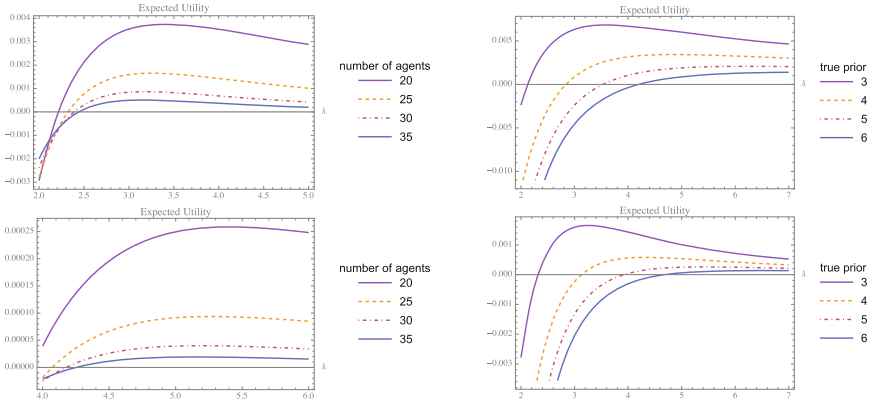
$$\frac{\partial U_i(\lambda_i, \lambda_{-i})}{\partial \lambda_i} = \frac{\partial U_i(\lambda_i, \lambda_{-i})}{\partial \lambda_i} \Big|_{D_1} + \frac{\partial U_i(\lambda_i, \lambda_{-i})}{\partial \lambda_i} \Big|_{D_2} > 0, \quad \forall \lambda_i \in (0, \bar{\lambda}_i].$$

**Theorem 6.** *Nash equilibrium happens at  $(\lambda_1, \dots, \lambda_n)$  where  $\lambda_i > \bar{\lambda}_i$  for any  $i \in [n]$  if Nash equilibrium exists under this setting.*

*Proof.* According to Theorem 5, each bidder always has an incentive to deviate from his or her true prior distribution by increasing the value of  $\lambda_i$  whenever  $\lambda_i \leq \bar{\lambda}_i$ . This result indicates that Nash equilibrium can only be achieved where  $\lambda_i$  is strictly greater than  $\bar{\lambda}_i$ .

### 3.4 Pictorial Representation

We present our results pictorially in order to show that bidders do have an incentive to misreport their prior distributions in Fig. 1.



**Fig. 1.** (Left) Suppose that the others bid according to the negative exponential distribution characterized by 1. We draw two different plots of individual expected utility on the condition that true prior distributions of the selected bidder are characterized by 3 and 5 respectively. It turns out that this bidder’s optimal strategy is to bid according to a distribution with the parameter  $\lambda$  strictly larger than 3 (in the first plot) or 5 (in the second plot) when total numbers of agents range from 20 to 35. (Right) suppose that the others bid according to the negative exponential distribution characterized by 1. We draw two different plots of individual expected utility on the condition that total numbers of agents are 12 and 20 respectively. It turns out that the selected bidder’s optimal strategy is to bid according to a distribution with the parameter  $\lambda$  strictly larger than that of its true prior when its true prior distributions are characterized by 3, 4, 5 and 6.

### 4 Conclusions

Our work shows a possibility where exploration and exploitation can be achieved against agents of values following the power-law distribution, but not the half-normal distributions nor negative exponential distribution. In general, the normal distribution is the most adopted distribution because of law of large numbers [4]. It is not difficult to see that other distributions share the same property of being able to cheat to benefit. This does not show a contradiction to Myerson auction’s maximality property. The difference is that our study starts at a lower knowledge hierarchical level, where agent’s value distribution is exhibited through their bids, and common knowledge is assumed as an ex post outcome. The power-law distribution has been observed to present itself often in economics [7]. The fact it becomes a key factor in our positive results is interesting. We would like to explore the matter further to investigate a possibility this may lead to a better understanding of the data driven scientific research paradigm more extensively.

## References

1. Anderson, C.: The end of theory: the data deluge makes the scientific method obsolete. *Wired Mag.* **16**(7) (2008)
2. Debreu, G.: The mathematization of economic theory. *Am. Econ. Rev.* **81**(1), 1–7 (1991)
3. Deng, X., Xiao, T., Zhu, K.: Learn to play maximum revenue auction. *IEEE Trans. Cloud Comput.* **PP**, 1 (2017)
4. Durrett, R.: *Probability: Theory and Examples*. Cambridge University Press, Cambridge (2010)
5. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second-price auction: selling billions of dollars worth of keywords. *Am. Econ. Rev.* **97**(1), 242–259 (2007)
6. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: *Reasoning About Knowledge*. The MIT Press, Cambridge (1995)
7. Gabaix, X.: Power laws in economics and finance. *Annu. Rev. Econ.* **1**, 255–293 (2009)
8. Kitchin, R.: Big data, new epistemologies and paradigm shifts. *Big Data Soc.* **1**, 1–12 (2014)
9. Myerson, R.B.: Optimal auction design. *Math. Oper. Res.* **6**(1), 58–73 (1981)
10. Tang, P., Zeng, Y.: How to manipulate truthful prior-dependent mechanisms? *CoRR*, abs/1606.02409 (2016)



# Low-Weight Superimposed Codes and Their Applications

Luisa Gargano<sup>✉</sup>, Adele A. Rescigno<sup>✉</sup>, and Ugo Vaccaro<sup>(✉)</sup>

Dipartimento di Informatica, Università di Salerno, 84084 Fisciano, SA, Italy  
uvaccaro@unisa.it

**Abstract.** A  $(k, n)$ -superimposed code is a well known and widely used combinatorial structure that can be represented by a  $t \times n$  binary matrix such that for any  $k$  columns of the matrix and for any column  $\mathbf{c}$  chosen among these  $k$  columns, there exists a row in correspondence of which column  $\mathbf{c}$  has an entry equal to 1 and the remaining  $k - 1$  columns have entries equal to 0. Due to the many situations in which superimposed codes find applications, there is an abundant literature that studies the problem of constructing  $(k, n)$ -superimposed codes with a small number  $t$  of rows. Motivated by applications to conflict-free communication in multiple-access networks, group testing, and data security, we study the problem of constructing superimposed codes that have the additional constraints that the number of 1's in each column of the matrix is constant, and equal to an input parameter  $w$ . Our results improve on the known literature in the area. We also extend our findings to other important combinatorial structures, like selectors.

## 1 Superimposed Codes

A  $(k, n)$ -superimposed code consists of a  $t \times n$  binary matrix such that for any  $k$  columns of the matrix and for any column  $\mathbf{c}$  chosen among these  $k$  columns, there exists a row in correspondence of which column  $\mathbf{c}$  has an entry equal to 1 and the remaining  $k - 1$  columns have entries equal to 0.

Superimposed codes (also known as cover-free families [23], strongly selective families [13], disjoint matrices [20]) find applications in a surprising variety of areas: Compressed sensing [5, 16, 31], cryptography and data security [37, 47], computational biology [2, 18], multi-access communication [17, 46], database theory [33], pattern matching [15, 41], distributed coloring [38], secure distributed computation [6], and circuit complexity [11], among the others. Due to the importance of the many different problems where superimposed codes find applications, considerable effort has been devoted to the design of algorithms for the construction of superimposed codes with good parameters.

Typically, the parameter to optimize is the number  $t$  of rows of the matrix representing the superimposed code. It is impossible to summarize here the long list of important contributions to this area and we address the reader to the survey paper [21] and Chap. 7 of [20] for a (partial) account of the literature.

Recent important results are contained in the papers by Porat and Rotschild [42] and by Bshouty [8], who were the first to give polynomial time algorithms for the construction of superimposed codes with  $t = O(k^2 \log_2 n)$ . Recall that for any  $(k, n)$ -superimposed code it holds the fundamental lower bound  $t = \Omega(k^2 \log_k n)$  [1, 22, 27, 43], therefore the constructions by Porat and Rotschild [42] and by Bshouty [8] are not too far from being optimal.

Although in many situations the most relevant parameter is indeed the number  $t$  of rows of the superimposed code (oftentimes referred to as the *length* of the code), there are many other circumstances where other parameters of the code have a greater importance. We illustrate a few of them in the following subsections, in order put our successive investigations in the proper context.

### 1.1 Conflict Resolution in Wireless Networks with Bounded Number of Transmissions

One of the early applications of superimposed codes was for conflict resolution in multiaccess channels [46]. In this scenario one has a multiaccess channel,  $n$  stations that have access to it, and at most a  $k \ll n$  of them might be *active* at the same time, i.e., might want to transmit over the channel. The goal is to let all active station to successfully transmit over the channel. The basic constraint is that an active station successfully transmits if and only if it transmits singly on the channel. Following the model and assumptions laid out in the seminal paper by Massey and Mathys [39], we assume that time is divided into time slots and that transmissions occur during these time slots. All stations have a global clock and active stations start transmitting at the same time slot. In a distributed model, a scheduling algorithm can be represented by a set of  $n$  Boolean vectors of length  $t$ , identified by integers from 1 through  $n$ , each of which corresponds to a distinct station, with the meaning that station  $j$  is scheduled to transmit at step  $i$  if and only if the  $i$ -th entry of its associated Boolean vector  $j$  is 1. A *conflict resolution* algorithm for the above described multiaccess system is a scheduling protocol that allows active stations to transmit successfully. A *non adaptive* conflict resolution algorithm is a protocol that for each step  $i = 1, \dots, t$  establishes which stations should transmit at step  $i$  without looking at what happened over the channel at the previous steps. A non adaptive conflict resolution algorithm can be represented by the Boolean matrix having as columns the  $n$  Boolean vectors associated with the scheduling of the transmissions of the  $n$  stations. Entry  $(i, j)$  of such a matrix is 1 if and only is station  $j$  is scheduled to transmit at step  $i$ . A crucial parameter is the number of rows of the matrix, since it represents the number of time slots the conflict resolution algorithm needs to let all  $k$  active stations transmit with success.

It is clear that there is a perfect equivalence between non adaptive conflict resolution algorithms in the above described scenario and  $(k, n)$ -superimposed codes. Indeed, superimposed codes ensures that each station, out of  $k$  conflicting ones, has a time slot in which it transmits singly (and therefore successfully) over the multi access channel. It is equally clear why it is important to construct



$(k, n)$ -superimposed codes with a small number  $t$  of rows, since  $t$  naturally corresponds to the worst case number of time slots required by the associated conflict resolution protocol, that is, the time-efficiency of the conflict resolution protocol. However, there can be other parameters of conflict resolution protocols one wants to optimize. Indeed, in order to save energy, it would be desirable that the protocol requires each station to transmit a *bounded number*  $w$  of time. This aspect is particularly important when stations are autonomous sensors, that have severe energy constraints [7]. Motivated by these energy constraints, several authors have started to investigate communications protocols that require a *bounded* number of transmissions for each node in the network [30, 32, 34]. In our scenario, energy constraints suggest the following natural algorithmic problem: *find the minimum integer  $t(k, n, w)$  such there exists a  $(k, n)$ -superimposed code of length  $t(k, n, w)$ , whose columns have exactly  $w$  ones.*<sup>1</sup> We notice that the property that each column has exactly  $w$  ones is not much stronger than requiring that each column has at most  $w$  ones, since a code satisfying the latter property can be converted to a code satisfying the former property by adding at most  $w - 1$  rows. Adding rows cannot destroy the code property.

Finally, we remark that the number of attempted transmission has been used in other contexts, as well (e.g., [4, 9, 10]).

## 1.2 Group Testing

The problem of finding superimposed codes, where the number of 1's in each column is limited, naturally arises also in the area of Group Testing. In Group Testing one has a population  $[n] = \{1, \dots, n\}$  of  $n$  individuals, an unknown set  $P \subseteq [n]$  of positive individuals,  $|P| \leq k$ , and one wants to determine  $P$  by performing tests of the form “is  $A \cap P \neq \emptyset$ ?”, where  $A$  is a chosen subset of  $[n]$ . This is an important problem, with many applications in different areas [19, 20].

It is well known that  $(k, n)$ -superimposed codes *are equivalent* to non adaptive algorithms to solve the Group Testing problem [20], where tests corresponds to subsets of  $[n]$  whose characteristic vectors are the rows of the  $(k, n)$ -superimposed code. Non adaptive algorithms are procedures in which the tests can be performed in parallel, since no test executed by the algorithm requires the knowledge of the outcomes of previous executed tests. In practice, non adaptive tests are by far the most preferable ones. The usual parameter one wants to optimize is the number of tests, that is, the number of rows of the  $(k, n)$ -superimposed code. However, a moment of thought reveals that the number of 1's in each column is also a crucial parameter. To see why this is so, let us remind that the original motivation for which Group Testing was introduced were to economize the blood testing of troops during WWII (see Chap. 1 of [19]). Here, blood samples of individual soldiers are pooled together and tested to see whether at least one member of the group is infected. If one employs

---

<sup>1</sup> From now on, the number of 1's in a binary vector will be denoted as the *weight* of that vector.

the tests given by a given  $(k, n)$ -superimposed code, then its number of rows represents the total number of performed tests, and the number  $w$  of 1's in the generic column  $i$  represents the number of times person  $i$  is tested, that is, *the number of unit of bloods of person  $i$  the algorithm needs*. It is clear that  $w$  should be kept limited, since individual blood samples cannot be recycled, once pooled together.

There are other practical situations in which it is desirable to limit the number of times individual items are tested in a group testing procedure. For instance, Sobell and Groll [45] describe several scenarios in the area of product testing in which groups of items are tested together, and the outcome of each test tell us whether or not the group of items contains at least a defective one. The final goal is to discover all the malfunctioning items. Usually, those tests are kind of “stress tests” (for instance, to discover whether gas containers might leak, they are filled up to a pressure higher than the operating one), therefore it is important to limit the number of times each item is tested since too many stress tests could by themselves break things. Again, this limitation corresponds to bounding the weight of each column in the employed superimposed code.

It is time to recall that the almost optimal-length superimposed codes of [8, 42] (with number of rows  $t = O(k^2 \log_2 n)$ ) have, in each column, a number  $w$  of 1's that is  $w = \Theta(k \log n)$ . This means that in a group testing procedure that employs such superimposed codes, one needs  $\Theta(k \log n)$  samples from each individual, to remain in the blood testing scenario. In view of the recent resurgence of the blood testing motivations of Group Testing in distressed area (see [24, 26, 48] and references therein quoted) classical constructions of  $(k, n)$ -superimposed codes *are not* appropriate. The same conclusion holds for other applications of Group Testing. Hence, the study of above introduced parameter  $t(k, n, w)$  has its importance.

### 1.3 Additional Applications

There are several situations in Data Security where  $(k, n)$ -superimposed codes find applications. In some of them [37, 47], superimposed codes are used to distribute to parties some pieces of information that is not possessed by *any other* coalitions of at most  $k - 1$  other participants. The method is obvious: give participant  $j$  a “key”  $\alpha_i$  if and only if in the  $i$ -th row of the  $j$ -th column of the superimposed code there is a 1. The number of 1's in each column represents the amount of information that the corresponding participants receive. In many practical scenarios, one wants to keep bounded that amount and the total number of keys to generate. Therefore, the parameter  $t(k, n, w)$  again comes into play. Finally, superimposed codes in which the number of 1's in each column is limited find applications also in the scenario considered in [3, 29].

## 2 Our Results in Perspective

Motivated by the scenarios described in the previous section, in which it is important to keep the weight  $w$  bounded, in this paper we intend to study the

tradeoff between the length of superimposed codes and the weight  $w$  of the columns in the code. We first formally define the main object of our studies.

**Definition 1.** *Let  $k, w, n$  be positive integers,  $k \leq n$ . A  $(k, w, n)$ -superimposed code is a  $t$ -rows  $n$ -columns binary matrix  $M$  such that each column has weight  $w$  and for any  $k$ -tuple of columns of  $M$  the following property holds: For any column  $\mathbf{c}$  of the given  $k$ -tuple, there is a row  $i \in \{1, \dots, t\}$  such that column  $\mathbf{c}$  has 1 in row  $i$  and all remaining  $k - 1$  columns of the  $k$ -tuple have all 0 in row  $i$ . The number of rows  $t$  of the matrix  $M$  is called the length of the  $(k, w, n)$ -superimposed code.*

The property that characterizes superimposed codes can be equivalently stated as follows: *For any column  $\mathbf{c}$  of  $M$  and for any other  $k - 1$  columns (all different from  $\mathbf{c}$ ) there is a row  $i \in \{1, \dots, t\}$  such that column  $\mathbf{c}$  has 1 in row  $i$  and all remaining columns of the  $(k - 1)$ -tuple have all 0 in row  $i$ .*

For given  $k, w, n$ , let us denote by  $t(k, w, n)$  the minimum  $t$  such that a  $(k, w, n)$ -superimposed code of length  $t$  exists. The classical bounds on the length of *unrestricted* superimposed codes are, clearly, bounds on  $\min_w t(k, w, n)$ . Here we are interested in finding good upper and lower bound on  $t(k, w, n)$ , for any value of the parameter  $w$ . The following theorem is one of the main results of our paper.

**Theorem 1.** *There exists a  $(k, w, n)$ -superimposed code of length  $t$ , where  $t$  is the minimum integer such that the following inequality holds:*

$$e \left( \frac{w(k-1) - \frac{w-1}{2}}{t - \frac{w-1}{2}} \right)^w k \left[ \binom{n}{k} - \binom{n-k+1}{k} \right] \leq 1. \tag{1}$$

Quite surprisingly (in view of the practical importance of weight-limited superimposed codes, as highlighted in the previous section) there are not many results in the literature to which compare our Theorem 1. Erdős, Frankl, and Füredi proved a results on extremal set theory [23] that, translated in our language, says that a  $(k, w, n)$ -superimposed code of length  $t$  always exists, provided that

$$n \geq \binom{t}{\lceil \frac{w}{k} \rceil} \times \binom{w}{\lceil \frac{w}{k} \rceil}^{-2}. \tag{2}$$

We can show that our result is much better, in the sense that it produces  $(k, w, n)$ -superimposed codes of shorter length. Recently, Gandikota *et al.* [28] proved that a  $(k, n)$ -superimposed code of length  $t$  in which all columns have weight *at most*  $w$  always exists, provided that

$$\left( \frac{w(k-1)}{t} \right)^w \binom{n}{k-1} (n-k+1) < 1. \tag{3}$$

One can see that the bounds  $t(k, w, n) \leq t$  one gets from our Theorem 1 is always better than the bound  $t(k, w, n) \leq t + w - 1$  one would get from (3). A way to see this is to notice that

$$\left(\frac{w(k-1) - \frac{w-1}{2}}{t - \frac{w-1}{2}}\right)^w < \left(\frac{w(k-1)}{t}\right)^w,$$

(and in general the first term of the inequality is much smaller than the second), moreover

$$ek \left[ \binom{n}{k} - \binom{n-k+1}{k} \right] < k \binom{n}{k} = \binom{n}{k-1} (n-k+1)$$

for  $k < \sqrt{n}$  (see below for a justification of the importance of this interval of values). The computations that leads to above inequality are somewhat cumbersome and they are deferred to the extended version of this paper. However, it is clear that our improvement is much more significant, mainly in the important regime where the parameter  $k$  is much smaller with respect to  $n$ . It is indeed customarily, in Group Testing for instance, to assume that  $k \ll n$ . This assumption is dictated by the practical scenarios where Group Testing finds real applications. Moreover, there is a well known result [44] proving that for  $k = \Omega(\sqrt{n})$  optimal superimposed codes have length equal to  $n$ , so there is nothing to optimize in that regime of  $k$ . Our improvement, although significant, does not implies an asymptotic improvement in the order of magnitude. We point out, however, that having success in doing so would imply beating the  $O(k^2 \log n)$  upper bound on weight-unrestricted superimposed codes, a problem that is open since the 60's.

We can also extend our findings to combinatorial objects strictly related to superimposed codes and that, likewise, find applications in different areas. In this version of the paper, we limit ourselves to the constructions of *selectors* [14,18], satisfying the same additional constraints on the weight of their columns discussed before. Selectors find applications in several areas, most notably in optimal two-stage Group Testing and in the efficient construction of optimal protocols for conflict resolution in multiple access channel with feedback [36]. This extension will be briefly analyzed in Sect. 4.

### 3 Proof of Theorem 1

In order to prove Theorem 1, we need the following technical result, whose proof is given in Appendix A.

**Fact 1.** *Given integers  $a, b, c$  with  $c \leq a \leq b$ , it holds that*

$$\binom{a}{c} \times \binom{b}{c}^{-1} \leq \left(\frac{a - \frac{c-1}{2}}{b - \frac{c-1}{2}}\right)^c \tag{4}$$

Furthermore, we need to recall the celebrated Lovász Local Lemma for the symmetric case (see Knuth's exercises 317 and 319 in [35]), as stated below.

**Lemma 1.** *Let  $A_1, A_2, \dots, A_b$  be events in an arbitrary probability space. Suppose that each event  $A_i$  is mutually independent of the set of all the other events  $A_j$  except for at most  $d$ , and that  $\Pr(A_i) \leq Q$  for all  $1 \leq i \leq b$ . If  $eQd \leq 1$ , then  $\Pr(\bigcap_{i=1}^b \bar{A}_i) > 0$ , where  $e = 2.71828\dots$  is the base of the natural logarithm.*

We are now ready to prove Theorem 1.

**Proof of Theorem 1.** Let  $M$  be a  $t \times n$  binary matrix, where each column  $\mathbf{c}$  is a  $t \times 1$  binary vector of weight  $w$ , and it is generated independently with probability

$$\Pr(\mathbf{c}) = \binom{t}{w}^{-1}.$$

Let  $A$  be any subset of indices of columns of  $M$ . Let us denote by  $M(A)$  the submatrix of  $M$  consisting of the  $|A|$ -tuple of columns of  $M$  whose indices belong to  $A$ . For a *given* column  $\mathbf{c}_i$  of  $M$  and set of column-indices  $B$ ,  $|B| = k - 1$ ,  $i \notin B$ , let us consider the “bad” event  $E_{\mathbf{c}_i, B}$  such that for each row in which  $\mathbf{c}_i$  has 1, it occurs that at least one of the remaining  $k - 1$  columns of  $M(B)$  has 1 in that same row. There are  $n \binom{n-1}{k-1}$  such an events. We want to compute the probability  $\Pr(E_{\mathbf{c}_i, B})$ . We know that the number of different 1’s that can be in the  $k - 1$  columns of  $M(B)$  is at most  $w(k - 1)$ . Therefore, the number of columns  $\mathbf{c}_i$  having *each* of its  $w$  1’s in the same rows where some of the columns of  $M(B)$  have 1, is upper bounded by  $\binom{w(k-1)}{w}$ . Therefore, for each  $\mathbf{c}_i$  and  $B$  as above, we have that

$$\Pr(E_{\mathbf{c}_i, B}) \leq \frac{\binom{w(k-1)}{w}}{\binom{t}{w}}. \tag{5}$$

By using Fact 1, we can further bound  $\Pr(E_{\mathbf{c}_i, B})$  as follows:

$$\Pr(E_{\mathbf{c}_i, B}) \leq \frac{\binom{w(k-1)}{w}}{\binom{t}{w}} \leq \left( \frac{w(k-1) - \frac{w-1}{2}}{t - \frac{w-1}{2}} \right)^w = Q \tag{6}$$

Let  $\{1, 2, \dots, n\} = [n]$  be the set of column indices of matrix  $M$ . Fixed an event  $E_{\mathbf{c}_i, X}$ , all events  $E_{\mathbf{c}_j, Y}$  are independent from  $E_{\mathbf{c}_i, X}$  if  $Y \subseteq [n] \setminus (X \cup \{i\})$  and  $j \notin Y$ . The number of such events is

$$\binom{n-k}{k-1} (n-k+1) = k \binom{n-k+1}{k}.$$

Therefore, for any given  $E_{\mathbf{c}_i, X}$  the number of events  $E_{\mathbf{c}_j, Y}$  that are dependent from  $E_{\mathbf{c}_i, X}$  is upper bounded by  $d$ , where

$$d = n \binom{n-1}{k-1} - k \binom{n-k+1}{k} = k \left[ \binom{n}{k} - \binom{n-k+1}{k} \right]. \tag{7}$$

According to Lemma 1, if parameters  $Q$  and  $d$ , as defined in (6) and (7) respectively, satisfy  $eQd \leq 1$  then the probability that *none* of the “bad” events  $E_{\mathbf{c}_i, B}$  occurs is strictly positive. That is, there is a strictly positive probability that *for each* column  $\mathbf{c}_i$  of matrix  $M$  and *for any* subset  $B$  of  $k - 1$  indices of columns of  $M$ , there exists an index row  $r \in \{1, \dots, t\}$  such that column  $\mathbf{c}_i$  has 1 in the row indexed by  $r$  and all remaining  $k - 1$  columns with indices in  $B$  have all 0 in the row indexed by  $r$ . In other terms, it is positive the probability that matrix  $M$  is a  $(k, w, n)$ -superimposed code.  $\square$

**Remark.** We would like to remark that we could have used a simple union bound to prove an result analogous to our Theorem 1, but weaker. If we had proceeded in this way, we would have proved that there exists a  $(k, w, n)$ -superimposed code of length  $t$ , provided that  $t$  satisfies the inequality

$$\frac{\binom{w(k-1)}{w}}{\binom{t}{w}} \binom{n}{k-1} (n-k+1) < 1. \tag{8}$$

or, in a weaker form, provided that  $t$  is such that

$$\left( \frac{w(k-1) - \frac{w-1}{2}}{t - \frac{w-1}{2}} \right)^w \binom{n}{k-1} (n-k+1) < 1. \tag{9}$$

It is immediate to see that also this simpler way to proceed gives a bound (9) that is better than the bound (3) by Gandikota *et al.* [28].

One can de-randomize the obvious algorithm deriving from Theorem 1 by using the important result of [40]. The algorithm one would get has complexity  $O(n^k)$ , that might look prohibitive. However, please notice that superimposed codes need to be constructed *once and for all*, and not every time one has to use them in a given application. One can also see that in order to satisfy inequality (1) in Theorem 1 it is sufficient to choose the parameter  $t$  in such a way that

$$t \geq \frac{w-1}{2} + \left( w(k-1) - \frac{w-1}{2} \right) \left( ek \left[ \binom{n}{k} - \binom{n-k+1}{k} \right] \right)^{\frac{1}{w}}.$$

Hence, we have the following corollary to Theorem 1.

**Corollary 1.** *There exists a  $(k, w, n)$ -superimposed code of length  $t$ , where*

$$t \leq \left\lceil \frac{w-1}{2} + \left( w(k-1) - \frac{w-1}{2} \right) \left( ek \left[ \binom{n}{k} - \binom{n-k+1}{k} \right] \right)^{\frac{1}{w}} \right\rceil.$$

One can get a more “readable” (but much weaker) expression of Corollary 1 by using the well known inequality  $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$  to obtain

$$t \leq \left\lceil w \left( k - \frac{3}{2} \right) + \frac{1}{2} \right\rceil \left( \frac{e^{1+\frac{1}{k}}}{k^{1-1/k}} \right)^{\frac{k}{w}} n^{\frac{k}{w}} + \frac{w-1}{2}. \tag{10}$$

If we parametrize the weight  $w$  in terms of the other parameter  $k$ , as  $w = kj$  for integer  $j$ , then (10) tell us how the length of the superimposed codes decreases, as the allowed weight of the columns of the superimposed code increases, in the following nice way:

**Corollary 2.** *There exists a  $(k, kj, n)$ -superimposed code of length at most*

$$\left\lceil \left( k - \frac{3}{2} \right) kj + \frac{1}{2} \right\rceil \left( \frac{e^{1+\frac{1}{k}}}{k^{1-1/k}} \right)^{\frac{1}{j}} \sqrt[j]{n} + O(kj).$$

We can obtain another interesting consequence of (10) by plugging into it the value  $w = k \log n$ , in order to get the following result.

**Corollary 3.** *There exists a  $(k, k \log n, n)$ -superimposed code of length at most*

$$2 \left( \frac{e^{1+\frac{1}{k}}}{k^{1-1/k}} \right)^{\frac{1}{\log n}} k \left( k - \frac{3}{2} \right) \log n + O(k \log n).$$

Considering that  $\frac{e^{1+\frac{1}{k}}}{k^{1-1/k}} < 1$  for  $k \geq 5$ , Corollary 3 implies the existence of  $(k, k \log n, n)$ -superimposed codes of length  $O(k^2 \log n)$  (and this is well known), where the hidden constant in the Big-Oh notation is *strictly smaller* than 2. For all the practical values of the parameters, our derived bound is better than the best known upper bound on the length of *weight-unconstrained* superimposed codes given in [12], that reads

$$\min_w t(k, w, n) \leq \frac{k+1}{B_k} \log n, \quad \text{where } B_k = \max_{q>1} \frac{-\log[1 - (1 - 1/q)^k]}{q}.$$

We have already noticed that results from [23] allow to say that there exists a  $(k, w, n)$ -superimposed code with length  $t$  such that

$$n \geq \binom{t}{\lceil \frac{w}{k} \rceil} \times \binom{w}{\lceil \frac{w}{k} \rceil}^{-2}. \tag{11}$$

The upper bound on  $t(k, w, n)$  one can extract from (11) is

$$t(k, w, n) \leq \frac{e}{2} [w(2k - 1) + k] (nek)^{\frac{k}{w}} + \frac{w}{2k}. \tag{12}$$

It is evident that our upper bound (10) is better. In the full version of the paper we will provide numerical comparison between the bound obtained from (10) and the exact value obtainable from (11) showing that our improvement is significant.

## 4 Selectors

Selectors can be considered as generalization of superimposed codes and they were introduced in [18]. Since then, they have been found applications in many different areas, most notably to obtain optimal two-stage group testing algorithm [18, 25], optimal protocols in the multiple access scenario of [36] (see also [14]). In this section, we intend to study selectors in which the weight of each columns is forced to be equal to an input parameter  $w$ . The motivations for this constrained are similar to the ones illustrated in the first section of this paper.

**Definition 2.** *Let  $k, w, n, p$  be positive integers,  $1 \leq p \leq k \leq n$ . A  $(p, k, w, n)$ -selector is a  $w$ -weighted  $t \times n$  binary matrix  $M$  (i.e., in which all columns have weight  $w$ ) such that for any  $k$ -tuple of columns of  $M$  the following property holds: At least  $p$  rows of the identity matrix  $I_k$  appear among the selected  $k$  columns of  $M$ . The number of rows  $t$  of the matrix  $M$  is called the length of the  $(p, k, w, n)$ -selector.*

It is clear that for  $p = k$  we get the usual definition of  $(k, w, n)$ -superimposed codes. In the following we introduce a property of a binary matrix and then we prove that a matrix enjoying such a property is a selector. We will use it to bound the length of a selector.

**Definition 3.** Let  $M$  be a  $t$ -rows  $n$ -columns  $w$ -weighted binary matrix. We say that  $M$  enjoys the  $(p, k)$ -property, for  $1 \leq p \leq k \leq n$ , if for any  $k$ -tuple of columns of  $M$  and any  $k - p + 1$  columns chosen among the selected  $k$ , there exists a row of the identity matrix  $I_k$  whose 1 appears among the selected  $k - p + 1$  columns of  $M$ .

**Lemma 2.** Let  $M$  be a  $t$ -rows  $n$ -columns  $w$ -weighted binary matrix. Let  $1 \leq p \leq k \leq n$ . If  $M$  enjoys the  $(p, k)$ -property then  $M$  is a  $(p, k, w, n)$ -selector.

*Proof.* By contradiction, assume that  $M$  enjoys the  $(p, k)$ -property but  $M$  is not a  $(p, k, w, n)$ -selector. Since  $M$  is not a  $(p, k, w, n)$ -selector there exists a  $k$ -tuple of columns of  $M$ , whose set of indices is  $K$ ,  $|K| = k$ , for which no more than  $\alpha \leq p - 1$  rows of the identity matrix  $I_k$  appear in  $M(K)$ . Let us write  $K = K' \cup K''$ ,  $K' \cap K'' = \emptyset$ , where  $K'$  contains the column indices where such  $\alpha \leq p - 1$  rows of  $I_k$  have a 1. Clearly,  $|K''| \geq k - p + 1$  and in the rows of  $M(K)$  does not appear any row of  $I_k$  whose 1 appears in the  $k - \alpha \geq k - p + 1$  column indices belonging to  $K''$ . This is in contradiction with the  $(p, k)$ -property of  $M$ . □

Lemma 2 allows to reduce the problem of finding a  $(p, k, w, n)$ -selector of small length to the problem of finding a  $t$ -rows  $n$ -columns  $w$ -weighted binary matrix enjoying the  $(p, k)$ -property whose number of rows  $t$  is small. We use this trick to prove the main result of this section.

**Theorem 2.** Let  $1 \leq p \leq k \leq n$ . There exists a  $(p, k, w, n)$ -selector of length  $t$ , where  $t$  is such that the following inequality holds

$$e^{\left(\frac{w(k-1) - \frac{w-1}{2}}{t - \frac{w-1}{2}}\right)^{w(k-p+1)}} \binom{k}{p-1} \left[ \binom{n}{k} - \binom{n-k}{k} \right] \leq 1. \tag{13}$$

*Proof.* Let  $M$  be a  $t$ -rows  $n$ -columns  $w$ -weighted random binary matrix, where each column  $\mathbf{c}$  is generated independently with probability

$$\Pr(\mathbf{c}) = \binom{t}{w}^{-1}.$$

We want to prove that it is positive the probability that matrix  $M$  is a  $(p, k, w, n)$ -selector; that is, there is a strictly positive probability that for each subset  $K$  of  $k$  indices of columns of  $M$ , at least  $p$  rows of the identity matrix  $I_k$  appear in  $M(K)$ .

Let  $A, B \subseteq [n]$ ,  $A \cap B = \emptyset$ ,  $|A| = k - p + 1$  and  $|B| = p - 1$ . Let us consider the event  $\bar{E}_{A,B}$  such that in  $M(A \cup B)$  there exists a row of the identity matrix



$I_k$  whose 1 appears among the  $k - p + 1$  columns of  $M(A)$ . The probability of the opposite “bad” event  $E_{A,B}$  that this does not hold is upper bounded by

$$\Pr(E_{A,B}) \leq \left( \frac{\binom{w(k-1)}{t}}{\binom{w}{t}} \right)^{k-p+1}. \tag{14}$$

By Fact 1 we get

$$\Pr(E_{A,B}) \leq \left( \frac{w(k-1) - \frac{w-1}{2}}{t - \frac{w-1}{2}} \right)^{w(k-p+1)} = Q. \tag{15}$$

By Lemma 2, we are interested in estimating the probability that *none* of the events  $E_{A,B}$  hold. There are

$$\binom{n}{k-p+1} \binom{n-(k-p+1)}{p-1}$$

such an events. Fixed an event  $E_{A,B}$ , all the events  $E_{A',B'}$  such that  $A' \subseteq [n] \setminus (A \cup B)$  and  $B' \subseteq [n] \setminus (A \cup B \cup A')$  are independent from  $E_{A,B}$ . The number of such event is

$$\binom{n-k}{k-p+1} \binom{(n-k)-(k-p+1)}{p-1},$$

and the number of events from which any event  $E_{A,B}$  can depend is upper bounded by

$$\begin{aligned} d &= \binom{n}{k-p+1} \binom{n-(k-p+1)}{p-1} - \binom{n-k}{k-p+1} \binom{(n-k)-(k-p+1)}{p-1} \\ &= \binom{k}{p-1} \left[ \binom{n}{k} - \binom{n-k}{k} \right]. \end{aligned} \tag{16}$$

By Lemma 1, if parameters  $Q$  and  $d$ , defined as in (15) and (16) respectively, satisfy  $eQd \leq 1$  then the probability that *none* of the “bad” events  $E_{A,B}$  occurs is strictly positive and the theorem is proved.  $\square$

By choosing the parameter  $t$  in such a way inequality (13) in Theorem 2 is satisfied, we get the following corollary.

**Corollary 4.** *There exists a  $(p, k, w, n)$ -selector of length  $t$ , where*

$$t \leq \left\lceil \frac{w-1}{2} + \left( w(k-1) - \frac{w-1}{2} \right) \left( e \binom{k}{p-1} \left[ \binom{n}{k} - \binom{n-k}{k} \right] \right)^{\frac{1}{w(k-p+1)}} \right\rceil.$$

By setting  $w = \frac{k}{k-p+1} \log n$  in above Corollary, and by performing simple algebra, one gets that there exists a  $(p, k, w, n)$ -selector of length at most

$$2 \left( \frac{e^{1+\frac{p}{k}}}{k^{1-\frac{p-1}{k}} (p-1)^{\frac{p-1}{k}}} \right)^{\frac{1}{\log n}} \frac{k}{k-p+1} \left( k - \frac{3}{2} \right) \log n + O \left( \frac{k}{k-p+1} \log n \right).$$

The second term in above expression is  $< 1$  for  $k \geq 9$ , allowing us to recover the bounds of [18, 25], on *weight-unrestricted* selectors, with a *much* smaller constant multiplying the leading factor.

**Acknowledgements.** We would like to thank the referees for their careful reading of the paper and their many useful suggestions.

## Appendix A: Proof of Fact 1

We first state and prove the following technical estimate.

**Fact 2.** *Given integers  $a, b, c$  with  $c \leq a \leq b$ , it holds that*

$$\frac{a}{b} \cdot \frac{a-c}{b-c} \leq \left( \frac{a-\frac{c}{2}}{b-\frac{c}{2}} \right)^2 \tag{17}$$

*Proof.* Since  $c \leq a \leq b$ , we have  $a(a-c) \leq b(b-c)$  and then  $a(a-c)c^2 \leq b(b-c)c^2$ . Adding the quantity  $4ab(a-c)(b-c)$  to both members of the above inequality, we have  $a(a-c)c^2 + 4ab(a-c)(b-c) \leq b(b-c)c^2 + 4ab(a-c)(b-c)$ , that immediately gives  $a(a-c)(2b-c)^2 \leq b(b-c)(2a-c)^2$  proving the fact.  $\square$

We can now prove Fact 1, that is,

Given integers  $a, b, c$  with  $c \leq a \leq b$ , it holds that:

$$\binom{a}{c} \times \binom{b}{c}^{-1} \leq \left( \frac{a-\frac{c-1}{2}}{b-\frac{c-1}{2}} \right)^c$$

*Proof.*

$$\begin{aligned} \binom{a}{c} \times \binom{b}{c}^{-1} &= \frac{a!}{(a-c)!} \cdot \frac{(b-c)!}{b!} \\ &= \frac{a}{b} \cdot \frac{a-1}{b-1} \cdot \dots \cdot \frac{a-(c-2)}{b-(c-2)} \cdot \frac{a-(c-1)}{b-(c-1)}. \end{aligned} \tag{18}$$

To upper bound expression (18), we consider two cases according to the situation in which  $c$  is even or odd.

- Let  $c$  be even. The  $c$  factors in the product  $\frac{a}{b} \cdot \frac{a-1}{b-1} \cdot \dots \cdot \frac{a-(c-2)}{b-(c-2)} \cdot \frac{a-(c-1)}{b-(c-1)}$  can be paired two by two as follows:

$$\frac{a-i}{b-i} \cdot \frac{a-(c-1-i)}{b-(c-1-i)} \quad \text{for } i = 0, \dots, \lceil \frac{c-1}{2} \rceil - 1 \tag{19}$$

Considering that  $\frac{a-(c-1-i)}{b-(c-1-i)} = \frac{a-i-(c-1-2i)}{b-i-(c-1-2i)}$ , we can apply Fact 2 to each pair in (19) having

$$\begin{aligned} \frac{a-i}{b-i} \cdot \frac{a-(c-1-i)}{b-(c-1-i)} &= \frac{a-i}{b-i} \cdot \frac{a-i-(c-1-2i)}{b-i-(c-1-2i)} \\ &\leq \left( \frac{a-i-\frac{c-1-2i}{2}}{b-i-\frac{c-1-2i}{2}} \right)^2 = \left( \frac{a-\frac{c-1}{2}}{b-\frac{c-1}{2}} \right)^2 \end{aligned}$$

- for  $i = 0, \dots, \lceil \frac{c-1}{2} \rceil - 1$  and Fact 1 follows in this case.
- Let  $c$  be odd. The product  $\frac{a}{b} \cdot \frac{a-1}{b-1} \cdot \dots \cdot \frac{a-(c-2)}{b-(c-2)} \cdot \frac{a-(c-1)}{b-(c-1)}$  has an odd number of factors and, except for the factor  $\frac{a - \frac{c-1}{2}}{b - \frac{c-1}{2}}$  in the middle, the remaining  $c - 1$  factors can be paired two by two as in (19). Reasoning as in the  $c$  even case, we can prove Fact 1 also in the odd case.  $\square$

## References

1. Alon, N., Asodi, V.: Learning a hidden subgraph. *SIAM J. Discr. Math.* **18**(4), 697–712 (2005)
2. Balding, D.J., Bruno, W.J., Torney, D.C., Knill, E.: A comparative survey of non-adaptive pooling design. In: Speed, T.P., Waterman, M.S. (eds.) *Genetic Mapping and DNA Sequencing. IMA Volumes in Mathematics and its Applications*, vol. 81, pp. 133–154. Springer, New York (1996). [https://doi.org/10.1007/978-1-4612-0751-1\\_8](https://doi.org/10.1007/978-1-4612-0751-1_8)
3. Bärtschi, A., Grandoni, F.: On conflict-free multi-coloring. In: Dehne, F., Sack, J.-R., Stege, U. (eds.) *WADS 2015. LNCS*, vol. 9214, pp. 103–114. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-21840-3\\_9](https://doi.org/10.1007/978-3-319-21840-3_9)
4. Bender, M.A., Kopelowitz, T., Pettie, S., Young, M.: Contention resolution with log-logstar channel accesses. In: *Proceedings of STOC 2016*, pp. 499–508 (2016)
5. van den Berg, E., Candés, E., Chinn, G., Levin, C., Olcott, P.D., Sing-Long, C.: Single-photon sampling architecture for solid-state imaging sensors. *Proc. Nat. Acad. Sci. U.S.A.* **110**(30), E2752–E2761 (2013)
6. Blundo, C., Galdi, C., Persiano, P.: Randomness recycling in constant-round private computations. In: Jayanti, P. (ed.) *DISC 1999. LNCS*, vol. 1693, pp. 140–149. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48169-9\\_10](https://doi.org/10.1007/3-540-48169-9_10)
7. Boyle, D., Kolcun, R., Yeatman, E.: Energy-efficient communication in wireless networks. In: Fagas, G. (ed.) *ICT - Energy Concepts for Energy Efficiency and Sustainability. InTech*, London (2017). <https://doi.org/10.5772/65986>
8. Bshouty, N.H.: Linear time constructions of some  $d$ -restriction problems. In: Paschos, V.T., Widmayer, P. (eds.) *CIAC 2015. LNCS*, vol. 9079, pp. 74–88. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-18173-8\\_5](https://doi.org/10.1007/978-3-319-18173-8_5)
9. Chang, Y.-J., Kopelowitz, T., Pettie, S., Wang, R., Zhan, W.: Exponential separations in the energy complexity of leader election. In: *Proceedings of STOC 2017*, pp. 771–783 (2017)
10. Chang, Y.-J., Dani, V., Hayes, T.P., He, Q., Li, W., Pettie, S.: The energy complexity of broadcast. [arXiv:1710.01800](https://arxiv.org/abs/1710.01800) [cs.DC] (2017)
11. Chaudhuri, S., Radhakrishnan, J.: Deterministic restrictions in circuit complexity. In: *Proceedings of 28th STOC*, pp. 30–36 (1996)
12. Cheng, Y., Du, D.-Z., Lin, G.: On the upper bounds of the minimum number of rows of disjunct matrices. *Optim. Lett.* **3**, 297–302 (2009)
13. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: *Proceedings of SODA 2001*, pp. 709–718 (2001)
14. Chlebus, B.S., Kowalski, D.R.: Almost optimal explicit selectors. In: Liśkiewicz, M., Reichuk, R. (eds.) *FCT 2005. LNCS*, vol. 3623, pp. 270–280. Springer, Heidelberg (2005). [https://doi.org/10.1007/11537311\\_24](https://doi.org/10.1007/11537311_24)

15. Clifford, R., Efremenko, K., Porat, E., Rothschild, A.:  $k$ -mismatch with don't cares. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 151–162. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-75520-3\\_15](https://doi.org/10.1007/978-3-540-75520-3_15)
16. Cormode, G., Muthukrishnan, S.: Combinatorial algorithms for compressed sensing. In: Flocchini, P., Gxcasieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 280–294. Springer, Heidelberg (2006). [https://doi.org/10.1007/11780823\\_22](https://doi.org/10.1007/11780823_22)
17. De Bonis, A., Vaccaro, U.: Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. *Theoret. Comput. Sci.* **306**, 223–243 (2003)
18. De Bonis, A., Gąsieniec, L., Vaccaro, U.: Optimal two-stage algorithms for group testing problems. *SIAM J. Comput.* **34**(5), 1253–1270 (2005)
19. Du, D.Z., Hwang, F.K.: *Combinatorial Group Testing and Its Applications*. World Scientific, River Edge (2000)
20. Du, D.Z., Hwang, F.K.: *Pooling Design and Nonadaptive Group Testing*. World Scientific, Singapore (2006)
21. D'yachkov, A.G.: Lectures on designing screening experiments, [arXiv:1401.7505](https://arxiv.org/abs/1401.7505) (2014)
22. D'yachkov, A.G., Rykov, V.V.: Bounds on the length of disjunct codes. *Probl. Pereda. Inf.* **18**(3), 7–13 (1982)
23. Erdős, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of  $r$  others. *Israel J. of Math.* **51**, 75–89 (1985)
24. Emmanuel, J.C., Bassett, M.T., Smith, H.J., Jacobs, J.A.: Pooling of sera for human immunodeficiency virus (HIV) testing: an economical method for use in developing countries. *J. Clin. Pathol.* **41**, 582–585 (1988)
25. Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved combinatorial group testing algorithms for real-world problem sizes. *SIAM J. Comput.* **36**(5), 1360–1375 (2007)
26. Finucane, M.M., Rowley, C.F., Paciorek, C.J., Essex, M., Pagano, M.: Estimating the prevalence of transmitted HIV drug resistance using pooled samples. *Stat. Methods Med. Res.* **25**(2), 917–935 (2016)
27. Füredi, Z.: On  $r$ -cover-free families. *J. Comb. Theory Ser. A* **73**, 172–173 (1996)
28. Gandikota, V., Grigorescu, E., Jaggi, S., Zhou, S.: Nearly optimal sparse group testing. In: 54th Annual Allerton Conference on Communication, Control, and Computing, pp. 401–408 (2016)
29. Gargano, L., Rescigno, A.A., Vaccaro, U.: On  $k$ -strong conflict-free multicoloring. In: Gao, X., Du, H., Han, M. (eds.) COCOA 2017. LNCS, vol. 10628, pp. 276–290. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-71147-8\\_19](https://doi.org/10.1007/978-3-319-71147-8_19)
30. Gasieniec, L., Kantor, E., Kowalski, D.R., Peleg, D., Su, C.: Time efficient  $k$ -shot broadcasting in known topology radio networks. *Distrib. Comput.* **21**, 117–127 (2008)
31. Gopi, S., Netrapalli, P., Jain, P., Nori, A.: One-bit compressed sensing: provable support and vector recovery. *JMLR* **28**(3), 154–162 (2013)
32. Kantor, E., Peleg, D.: Efficient  $k$ -shot broadcasting in radio networks. *Discret. Appl. Math.* **202**, 79–94 (2016)
33. Kautz, W.H., Singleton, R.C.: Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theory* **10**, 363–377 (1964)
34. Karmakar, S., Koutris, P., Pagourtzis, A., Sakavalas, D.: Energy-efficient broadcasting in ad hoc wireless networks. *J. Discret. Algorithms* **42**, 2–13 (2017)
35. Knuth, D.E.: *The Art of Computer Programming*, vol. 4, Fascicle 6, Satisfiability (2015)

36. Komlós, J., Greenberg, A.G.: An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels. *IEEE Trans. Inf. Theory* **31**, 303–306 (1985)
37. Kumar, R., Rajagopalan, S., Sahai, A.: Coding constructions for blacklisting problems without computational assumptions. In: Wiener, M. (ed.) *CRYPTO 1999*. LNCS, vol. 1666, pp. 609–623. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48405-1\\_38](https://doi.org/10.1007/3-540-48405-1_38)
38. Linial, N.: Locality in distributed graph algorithms. *SICOMP* **21**, 193–201 (1992)
39. Massey, J.L., Mathys, P.: The collision channel without feedback. *IEEE Trans. Inf. Theory* **31**, 192–204 (1985)
40. Moser, R.A., Tardos, G.: A constructive proof of the general Lovász local lemma. *J. ACM* **57**(2), 11 (2010)
41. Porat, B., Porat, E.: Exact and approximate pattern matching in the streaming model. In: *Proceedings of 50th FOCS*, pp. 315–323 (2009)
42. Porat, E., Rothschild, A.: Explicit non-adaptive combinatorial group testing schemes. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008*. LNCS, vol. 5125, pp. 748–759. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70575-8\\_61](https://doi.org/10.1007/978-3-540-70575-8_61)
43. Ruszinkó, M.: On the upper bound of the size of the  $r$ -cover-free families. *J. Comb. Theory Ser. A* **66**, 302–310 (1994)
44. Shangquan, C., Ge, G.: New bounds on the number of tests for disjunct matrices. *IEEE Trans. Inf. Theory* **62**, 7518–7521 (2016)
45. Sobel, M., Groll, P.A.: Binomial group-testing with an unknown proportion of defectives. *Technometrics* **8**(4), 631–656 (1966)
46. Wolf, J.: Born again group testing: multiaccess communications. *IEEE Trans. Inf. Theory* **31**, 185–191 (1985)
47. Zaverucha, G.: Hash families and cover-free families with cryptographic applications. Ph.D. Thesis, Waterloo, Ontario, Canada (2010)
48. Zenios, S.A., Wein, L.M.: Pooled testing for HIV prevalence estimation: exploiting the dilution effect. *Stat. Med.* **17**(13), 1447–1467 (1998)



# Single Vehicle's Package Delivery Strategy with Online Traffic Congestion of Certain Delay Time

Songhua Li<sup>1,2(✉)</sup> and Yinfeng Xu<sup>1,2</sup>

<sup>1</sup> School of Management, Xi'an Jiaotong University, Xi'an 710049, China  
lisonghua@stu.xjtu.edu.cn, yfxu@mail.xjtu.edu.cn

<sup>2</sup> The State Key Lab for Manufacturing Systems Engineering,  
Xi'an 710049, China

**Abstract.** Urban package delivery by single vehicle can be modeled as online Steiner travelling salesman problem. This paper investigates the scenario that delay time of traffic congestion is certain and of real-time information to delivery vehicle, the vehicle can obtain specific delay time of traffic congestion just upon reaching an end vertex of the corresponding edge. Given a graph  $G = (V, E)$  and a subset  $D$  in  $V$ , the goal is to design a minimum-weight closed tour that departs from the depot  $s$  in  $D$  and returns to  $s$  after visiting each destination node in  $D$  at least once. When delivery vehicle encounters at most  $k$  times of traffic congestion, we first show  $\min\{k + 1, \rho_\Sigma + 1\}$  is a lower bound on competitive ratio for the problem, where  $\rho_\Sigma$  is the ratio compared sum of delay time of all congested edges with cost of optimal route. We further present an online algorithm for the problem with its competitive ratio proved to be very close to the lower bound.

**Keywords:** Package delivery · Traffic congestion · Competitive analysis

## 1 Introduction

Package delivery service, especially “last mile” delivery service, is one of the key factor that leads E-commerce to succeed. As delivery vehicle in urban traffic network may face with complex traffic network and uncertain traffic conditions, it is essential to design effective delivery route for vehicles. Related researches mainly focus on Steiner Travelling Salesman Problem and Canadian Travelling Salesman Problem.

### 1.1 Literature Review

The Steiner Travelling Salesman Problem (STSP), coined by Cornuéjols et al. [1], is a special case of the General Routing Problem introduced by Orloff [5]. In STSP problem, we are given an edge-weighted undirected graph  $G = (V, E)$  and a set  $D \subseteq V$  of destination nodes. The optimization goal is to find a minimum-weight closed tour that departs from depot  $s \in D$  and returns to depot  $s$  after visiting each node in  $D$  at least once. Ratliff and Rosenthal [3] studied a very special case of STSP, that is order-picking in a rectangular warehouse which contains crossovers only at ends of aisles,

they proposed an algorithm for picking up an order in minimum time. More results on STSP problem with applications in order-picking are summarized in [6].

The Canadian Travelling Salesman Problem (CTP) is introduced by Papadimitriou and Yannakakis [4]. In CTP problem, we are given one edge-weighted graph  $G = (V, E)$ , the goal is to find a shortest path from a source vertex  $s \in V$  to a destination vertex  $t \in V$ . The variation is denoted as  $k$ -CTP problem when there are at most  $k$  times of edge blockages. Bar-Noy and Schieber [8] considered CTP problem with recoverable edge blockages and introduced  $k$ -CTP problem, they presented a strategy that minimizes the worst-case length. Westphal [9] firstly analyzed  $k$ -CTP problem by competitive analysis, they proved that there is no deterministic online algorithms within a  $(2k + 1)$ -competitive ratio for  $k$ -CTP problem and presented BACKTRACK algorithm that meets the lower bound, they also proposed a lower bound of  $(k + 1)$  on competitive ratio for online randomized algorithms. Bender and Westphal [11] recently proved this lower bound is tight by constructing a randomized online algorithm.

In recent works, Liao and Huang [12] studied double-valued graph problem, which is a generalization of  $k$ -CTP. They presented an adaptive algorithm with competitive ratio of  $\min\{r, 2k + 1\}$  when the number of traffic jams is up to a given constant  $k$ , where  $r$  is the worst-case performance ratio. They further extended the algorithm to recoverable  $k$ -CTP problem. Zhang et al. [7] studied the STSP problem with online non-recoverable edge blockages, in which blocked edges are of real-time information for the salesman. They proposed a lower bound of  $(k + 1)$  on competitive ratio for the problem, and presented an online exponential-time optimal algorithm, they also presented an online polynomial-time algorithm within a  $(k + 4)$ -competitive ratio.

## 1.2 Our Contributions

Recall assumptions in previous research, either there is no traffic congestion or traffic congestion (edge blockage) is not recoverable. In this paper, we study a more practical scenario that traffic congestion is recoverable, meaning that one traffic congestion can recover after some delay time. We consider the traffic congestion that is caused by traffic regulations or road constructions, and further assume delay time of traffic congestion is certain and of real-time information to delivery vehicle. When delivery vehicle encounters at most  $k$  times of traffic congestion, we first present a lower bound on competitive ratio for the problem, and further propose an online deterministic algorithm with its competitive ratio proved to be very close the lower bound.

The rest of this paper is organized as follows. In Sect. 2, we formally define STSP problem with certain delay time, and introduce some basic assumptions. Section 3 presents a lower bound on competitive ratio for the problem. In Sect. 4, we present an online polynomial-time near optimal algorithm for the problem, with its competitive ratio very close to the presented lower bound. Section 5 concludes this paper with some possible future work.

## 2 Preliminaries

### 2.1 Problem Statement

Urban traffic network can be modeled as a connected, undirected, and edge-weighted graph  $G = (V, E)$ , delivery destinations (customers' locations) can be modeled as nodes in set  $D \subseteq V$ . Delivery vehicle is required to depart from a depot  $s \in D$ , visit each destination in  $D$  to deliver packages, and finally return to depot  $s$ . Without loss of generality, we assume that edge weight, travel time and distance are used interchangeably in this paper. For each edge  $e = \{u, v\} \in E$ , its original weight  $w(e)$  denote the shortest traversal time from vertex  $u$  to vertex  $v$ . If the edge is not congested, delivery vehicle need to spend  $w(e)$  to travel this edge, otherwise, the edge needs some certain delay time  $\rho(e)$  to recover and vehicle needs to spend  $w^2(e) = \rho(e) + w(e)$  to travel this edge. We further assume that delivery vehicle meets at most  $k$  times of traffic congestion during its delivery process. Let  $\delta = \{e_1, e_2, \dots, e_k\}$  denote the sequence of online congested edges learnt by delivery vehicle, which will be revealed to the vehicle during the traversal. Let  $\delta_i = \{e_1, e_2, \dots, e_i\}$  denote the first  $i$  congested edges that are learnt by vehicle, let  $\rho_i = \{\rho(e_1), \rho(e_2), \dots, \rho(e_i)\}$  further denote the sequence of delay time of corresponding congested edges, where  $1 \leq i \leq k$ . The goal is to design a provably good delivery route for the vehicle to deliver packages facing with online traffic congestion, that departs from depot  $s$  and comes back to  $s$  after visiting each destination node in  $D$  at least once.

In this paper, we call above problem as *online blocked Steiner Travelling Salesman Problem with Certain Delay Time (online STSP-CDT problem)*. This way, an instance of this problem can be denoted as  $I = (V, E, D, s, \delta, \rho)$ . Our discussion is based on following assumptions.

#### Assumptions.

- (a) All destination nodes are connected in remainder graph such that feasible closed tours always exist.
- (b) The traffic congestion is of real-time information to delivery vehicle.

The graph is assumed to be always connected as otherwise package delivery will not succeed. Moreover, traffic congestion can be radioed to delivery vehicle in an online fashion with assistance of intelligent traffic system and widely used mobile devices. Assumption (b) means that delivery vehicle learns about congestion information once the congestion occurs, since adversary can choose when and which edge to block, this is equivalent to that delivery vehicle learns about one traffic congestion just on arrival at one vertex of the congested edge.

In this paper, we use  $A(I)$  to denote length (traversal time) of the closed tour obtained by an algorithm  $A$  on instance  $I = (V, E, D, s, \delta, \rho)$ . And we use  $OPT(I)$  to denote length of optimal (shortest) closed tour on  $I$  for the corresponding *offline STSP-CDT problem* where delivery vehicle knows all traffic congestion in advance. The competitive ratio is defined as



$$c_A = \sup_I \left[ \frac{A(I)}{OPT(I)} \right]$$

Where the supremum is taken over all instances. If  $r_A$  is finite, the algorithm  $A$  is called  $r_A$ -competitive. Note  $r_A \geq 1$  for any online algorithm, thus 1 is a trivial lower bound on competitive ratio for  $STSP$ - $CDT$  problem.

### 3 A Lower Bound

Recall that traffic congestion is of real-time information, delivery vehicle learns about one congested edge  $e = \{u, v\}$  just on arrival at vertex  $u$  or vertex  $v$ . In this section, we present a lower bound on competitive ratio for *online STSP-CDT problem*. We prove the lower bound by following instance  $I = (V, E, D, s, \delta, \rho)$ , in which graph  $G = (V, E)$  is shown in Fig. 1, destination set is  $D = \{s, v_{k+2}, v_{k+3}, \dots, v_{2k+2}\}$ , edges in  $G$  are labeled with original weights, where  $\varepsilon$  is a small positive real number, sequence of online congested edges is  $\delta = \{(v_m, v_{m+1+k}) | 1 \leq m \leq 2k+2\}$ . In the instance, we assume delay time of each congestion is a constant equal to  $\rho$ .

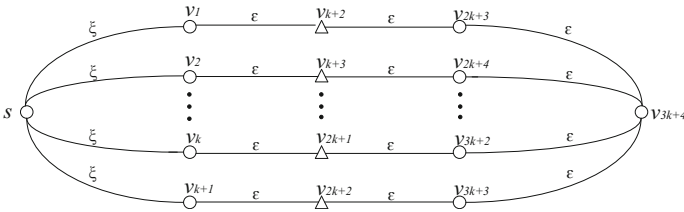


Fig. 1. The underlying graph  $G = (V, E)$  in the instance  $I$

**Theorem 1.** For online  $STSP$ - $CDT$  problem with at most  $k$  times of traffic congestion, there is no online deterministic algorithm that can achieve a competitive ratio less than  $\min\{1 + k, 1 + \rho_\Sigma\}$ , where  $\rho_\Sigma = \frac{\sum_{e \in \delta} \rho(e)}{OPT}$ .

Proof. We prove the lower bound by above instance with  $\xi = 1$ . Recall assumption in online  $STSP$ - $CDT$  problem, delivery vehicle can obtain delay time of a congested edge  $e = \{u, v\}$  just on arrival at vertex  $u$  or vertex  $v$ , and there are at most  $k$  times of traffic congestion. Clearly, there is not any congested edges in  $\{e = \{s, v_m\} | 1 \leq m \leq k+1\}$  in the worst case. Without loss of generality, we suppose the vehicle chooses  $e = \{s, v_i\}$  to traverse. Then adversary will block edge  $\{v_i, v_{i+k+1}\}$  when vehicle arrives at node  $v_i$  as otherwise vehicle can get one shorter route. If vehicle passes through this congested edge to visit the next destination, the cost is at most  $(\varepsilon + \rho)$ . If vehicle chooses

another path to visit the next destination, the cost is at least  $(2 + \varepsilon)$ . There are two cases according to the value of  $\rho$ .

Case 1.  $\rho < 2$ . When the vehicle meets a new congested edge  $\{v_i, v_{i+k+1}\}$ , he has to pass through it with traversal cost less than  $(2 + \varepsilon)$ . Vehicle can visit  $v_{3k+4}$  through edges  $\{v_{i+k+1}, v_{i+2k+2}\}$  and  $\{v_{i+2k+2}, v_{3k+4}\}$ , adversary can't congest any edges connected with vertex  $v_{3k+4}$ , which is similar to initial analysis. Delivery vehicle can visit another destination  $v_{j+k+1}$  along edge  $\{v_{3k+4}, v_{j+2k+2}\}$  and  $\{v_{j+2k+2}, v_{j+k+1}\}$ . As  $\varepsilon$  is sufficiently smaller than 1, the vehicle can come back to  $v_{3k+4}$  to visit the next destination. Then, adversary would block edge  $\{v_{j+k+1}, v_{j+2k+2}\}$ . Vehicle would pass through the new blocked edge with cost equal to  $(\rho + 2\varepsilon)$ . And so on for the next  $k - 2$  distinct attempts to go back to  $v_{3k+4}$  by the vehicle, the adversary would congest exactly  $(k - 2)$  out of  $(k - 1)$  edges in set  $\{\{v_{t+k+1}, v_{t+2k+2}\} | t \in \{1, 2, \dots, k+1\} \setminus \{i, j\}\}$ . Corresponding  $(k - 2)$  congested edges and  $\{\{v_{j+k+1}, v_{j+2k+2}\}, \{v_i, v_{i+k+1}\}\}$  can form the congested sequence  $\delta$ . At last, vehicle moves from  $v_{3k+4}$  to depot  $s$  without new traffic congestion, resulting in a closed tour with its length equal to

$$A(I) \geq 1 + (\varepsilon + \rho) + 2\varepsilon + (4\varepsilon + \rho) \cdot (k - 1) + 3\varepsilon + 1 = 2 + k\rho + (4k + 2)\varepsilon$$

The length of the optimal (offline) closed tour on  $I$  is

$$OPT(I) = 1 + 3\varepsilon + 4\varepsilon(k - 1) + 3\varepsilon + 1 = 2 + (4k + 2)\varepsilon$$

It follows that

$$c_A \geq \frac{A(I)}{OPT(I)} \geq \frac{2 + k\rho + (4k + 2)\varepsilon}{2 + (4k + 2)\varepsilon} \rightarrow 1 + \frac{k\rho}{2} = 1 + \rho_\Sigma, \text{ when } \varepsilon \rightarrow 0.$$

Case 2.  $\rho > 2$ . When delivery vehicle meets one congestion in edge  $\{v_i, v_{i+k+1}\}$ , vehicle have to go back to depot  $s$  and visit another destination. So on for the next  $k - 1$  distinct attempts to travel exactly  $(k - 1)$  out of  $\{\{v_t, v_{t+k+1}\} | t \in \{1, 2, \dots, k+1\} \setminus \{i\}\}$ . The vehicle will succeed in its  $(k + 1)$ st try, by visiting destination  $v_{m+k+1}$  along  $\{\{s, v_m\}, \{v_m, v_{m+k+1}\}\}$ , visiting  $v_{3k+4}$  along  $\{\{v_{m+k+1}, v_{m+2k+2}\}, \{v_{m+2k+2}, v_{3k+4}\}\}$ , and visiting the other destinations with minimal normal costs. Finally, vehicle goes back to depot  $s$ , resulting in a closed tour with length equal to

$$A(I) = 2k + 1 + 3\varepsilon + 4\varepsilon(k - 1) + 3\varepsilon + 1 = 2k + 2 + (4k + 2)\varepsilon$$

On the other hand, the length of the optimal (offline) closed tour on  $I$  is

$$OPT(I) = 1 + 3\varepsilon + 4\varepsilon(k - 1) + 3\varepsilon + 1 = 2 + (4k + 2)\varepsilon$$

It follows that

$$c_A \geq \frac{A(I)}{OPT(I)} \geq \frac{2 + 2k + (4k + 2)\varepsilon}{2 + (4k + 2)\varepsilon} \rightarrow 1 + k, \text{ when } \varepsilon \rightarrow 0.$$

Case 1 and Case 2 prove this theorem.

## 4 An Algorithm for Online *STSP-CDT* Problem

In this section, we present an online algorithm *Greedy Routing* for online *STSP-CRT* problem, and show that it is near-optimal in terms of competitive ratio.

### 4.1 Notations

$ST$ :	denote the simple Steiner tree spanning all destination vertices in $D$ ;
$CT$ :	denote the approximate closed tour derived by <i>Greedy Routing</i> ;
$e_i = \{v'_i, v''_i\}$ :	denote the $i$ th congestion encountered by delivery vehicle, where $v'_i$ is the first vertex of the congestion, $v''_i$ is the second node;
$\delta_i$ :	denote the first $i$ times of traffic congestion learned by delivery vehicle;
$d_i^b$ :	denote the closest visited destination vertex to vertex $v'_i$ in $CT$ ;
$d_i^a$ :	denote the closest unvisited destination vertex to vertex $v''_i$ in $CT$ ;
$v_c$ :	denote current position at which delivery vehicle is located;
$l_G^{\delta_i}(u, v)$ :	denote the shortest path from destination node $u$ to destination node $v$ under graph $G = (V, E, D, s)$ with known congested sequence $\delta_i$ , which is derived by Dijkstra's algorithm;
$l_{CT}^{\delta_i}(u, v)$ :	denote the travelling path from node $u$ to node $v$ along $CT$ with $\delta_i$ ;

### 4.2 Execution of Greedy Routing

Initially, *Greedy Routing* constructs a simple Steiner Tree ( $ST$ ) derived by algorithm *Subtree Traversal* [13], then it constructs an approximate closed tour ( $CT$ ) by a depth-first-search traversal on  $ST$ . Let delivery vehicle move along  $CT$ , during the traversal,  $v_c$  updates as delivery vehicle moves,  $\delta_i$  updates once encountering new congested edge. The algorithm *Greedy Routing* mainly consists of 3 routing strategies.

*Strategy 1.* Move along  $CT$  irrespective of remaining traffic congestion;

*Strategy 2.* Move along  $CT$  until meeting the next congestion;

*Strategy 3.* Go back to  $d_i^b$  from vertex  $v'_i$ , move to  $d_i^a$  along  $l_G^{\delta_i}(d_i^b, d_i^a)$ ;

---

**Algorithm. Greedy Routing**


---

**Input:** Graph  $G=(V,E,D,s)$ , congestion sequence  $\delta = \{e_1, e_2, \dots, e_k\}$  with corresponding delay **time**  $\rho = \{\rho(e_1), \rho(e_2), \dots, \rho(e_k)\}$ .

**Output:** A closed tour starting **from s and coming back to s** after visiting each destination node in  $D$  at least once.

1. Let  $i=0$ , current position  $v_c \rightarrow s$ ;
  2. Construct a simple Steiner tree ( $ST$ ) spanning destination nodes in  $D$ .
  3. Construct the initial closed tour ( $CT$ ) via depth-first-search traversal on  $ST$ ;
  3. Update  $v_c$  as delivery vehicle travel along  $CT$ ;
  4. **While** vehicle does not complete delivery **do**  
     calculate  $r = \frac{|(k-i) \max_{i+1 \leq t \leq k} \{\rho(e_t)\}|}{|ST|}$ ;  
     **if**  $r \leq k - i$  **then**  
         move along  $CT$ ;  
     **else if**  $r > k - i$  **then**  
         move along  $CT$  until meeting next congestion;  
         **While** meeting  $e_{i+1} = \{v'_{i+1}, v''_{i+1}\}$  **do**  
             update  $v_c \rightarrow v'_{i+1}$ ,  $\delta_{i+1} = \delta_i \cup \{e_{i+1}\}$ ;  
             **If** it satisfies *direct-moving condition* **then**  
                 move along  $CT$  until the next congestion;  
                 update  $v_c$ ,  $i=i+1$ ;  
             **otherwise**  
                 backtrack to  $d_{i+1}^b$ , **travel along**  $l_G^{\delta_{i+1}}(d_{i+1}^b, d_{i+1}^a)$ ,  
                 move along  $CT$  until the next congestion;  
                 update  $v_c$ ,  $i=i+1$ ;  
         **end while**  
     **end while**  
**end while**
- 

*Greedy Routing* firstly calculates ratio  $r = \frac{|(k-i) \max_{i+1 \leq t \leq k} \{\rho(e_t)\}|}{|ST|}$ , and follows different routing strategies according to different conditions.

If  $r \leq k - i$ , follows Strategy 1. Otherwise, it follows Strategy 2. When vehicle learns about new congested edge  $e_{i+1} = \{v'_{i+1}, v''_{i+1}\}$  on arrival at its first node  $v'_{i+1}$ . If it satisfies following condition (*Direct-moving condition*)

$$\rho(e_{i+1}) + w(e_{i+1}) + l_{CT}^{\delta_i}(v'_{i+1}, d_{i+1}^a) \leq l_G^{\delta_i}(v'_{i+1}, d_{i+1}^b) + l_G^{\delta_i}(d_{i+1}^b, d_{i+1}^a)$$

it follows Strategy 2 that passes through the blocked edge  $e_{i+1}$  to visit  $d_{i+1}^a$  until next congestion. Otherwise, it follows Strategy 3 that backtracks to the last destination  $d_{i+1}^b$ , and follows a new shortest path  $l_G^{\delta_{i+1}}(d_{i+1}^b, d_{i+1}^a)$  derived by a single-source shortest path algorithm in graph  $G = (V, E, D, s)$ , and then continues following  $CT$  until meeting a new congested edge.

### 4.3 Competitive Analysis on Greedy Routing

Notice vehicle's visiting sequence to destination nodes in  $D$  can't change during the traversal, and it is the same with visiting sequence in initial tour  $CT$ . We make  $d_0 = d_{n+1} = s$  and further denote the visiting sequence as  $d_0 \rightarrow d_1 \rightarrow \dots \rightarrow d_{n+1}$ . Recall that delivery vehicle must continue moving along  $CT$  after each time it handles the traffic congestion, no matter which strategy it follows by algorithm *Greedy Routing*. It takes the vehicle an additional cost each time it meets new congestion. Thus,

$$l_{CT}^{\delta_0}(d_i, d_{i+1}) \leq l_{CT}^{\delta_1}(d_i, d_{i+1}) \leq \dots \leq l_{CT}^{\delta_k}(d_i, d_{i+1}), \text{ for } \forall 0 < i < n$$

Further, we have

$$l_{CT}^{\delta_0}(d_i, d_j) \leq l_{CT}^{\delta_1}(d_i, d_j) \leq \dots \leq l_{CT}^{\delta_k}(d_i, d_j), \text{ for } \forall 0 \leq i < j \leq n$$

**Lemma 1.** *If the ratio  $r > k - i$  for each iteration  $i$  during the trip, total length of closed tour derived by Greedy Routing is at most  $\min\{(k+2)OPT, 2OPT + \sum_{e \in \delta} \rho(e)\}$ , when there are at most  $k$  times of traffic congestion.*

Proof. Recall that delivery vehicle learns about one congested edge  $e_i = \{v'_i, v''_i\}$  only on arrival at its first node  $v'_i$ , we denote  $d_{b_i}$  as the closest visited destination before node  $v'_i$ , for  $i \in \{1, 2, \dots, k\}$ . It is clearly that  $0 \leq b_1 \leq \dots \leq b_i \leq \dots \leq b_k \leq n$ , and  $e_i$  appears only once in the tour  $l_{CT}^{\delta_{i-1}}(d_{b_i}, d_{b_{i+1}})$ . We further denote  $d_{b_{k+1}} = s$ , denote  $l_i$  as the cost (i.e. the traversal time) from destination  $d_{b_i}$  to destination  $d_{b_{i+1}}$  in final closed tour derived by algorithm *Greedy Routing*. Let's consider the process traveling from  $d_{b_i}$  to  $d_{b_{i+1}}$ . Firstly, delivery vehicle travels from  $d_{b_i}$  to the first node  $v'_i$  of new congested edge  $e_i = \{v'_i, v''_i\}$  along  $CT$  with  $\delta_{i-1}$ , i.e.  $l_{i1} = l_{CT}^{\delta_{i-1}}(d_{b_i}, v'_i)$ . Then, vehicle meets  $e_i$ , and tries to visit  $d_{b_{i+1}}$  by *Greedy Routing*, with a cost equal to

$$l_{i2} = \min\{l_{CT}^{\delta_{i-1}}(v'_i, d_{b_{i+1}}) + \rho(e_i), l_G^{\delta_{i-1}}(v'_i, d_{b_i}) + l_G^{\delta_i}(d_{b_i}, d_{b_{i+1}})\}$$

Finally, vehicle travels from  $d_{b_{i+1}}$  to  $d_{b_{i+1}}$  along  $CT$ , i.e.  $l_{i3} = l_{CT}^{\delta_i}(d_{b_{i+1}}, d_{b_{i+1}})$ . Thus, traversal cost  $l_i = \sum_{j=1}^3 l_{ij}$ , for each all  $i = 1, 2, \dots, k$ . see Fig. 2.

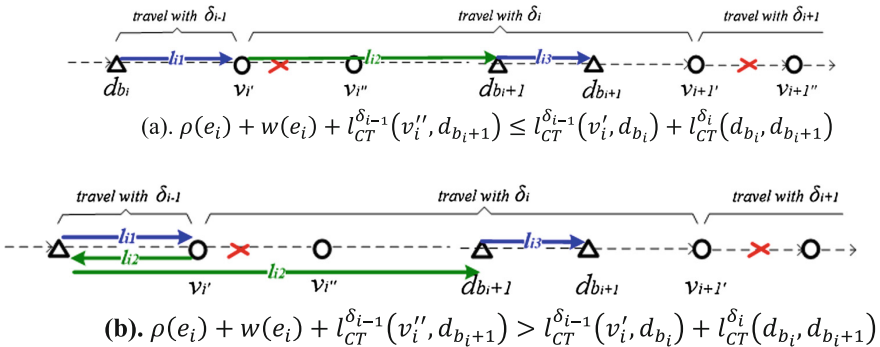


Fig. 2. The discussion in the traversal cost  $l_i$

According to which strategy *Greedy Routing* chooses, the above expression can be further discussed as follows

If  $\rho(e_i) + w(e_i) + l_{CT}^{\delta_{i-1}}(v'_i, d_{b_{i+1}}) \leq l_{CT}^{\delta_{i-1}}(v'_i, d_{b_i}) + l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}})$ , the delivery vehicle would pass through the congested edge and continue following *CT* until next congestion  $e_{i+1}$ . As we know that  $l_{i2} = l_{CT}^{\delta_{i-1}}(v'_i, d_{b_{i+1}}) + \rho(e_i)$ , and that

$$l_{CT}^{\delta_{i-1}}(d_{b_i}, v'_i) + l_{CT}^{\delta_{i-1}}(v'_i, d_{b_{i+1}}) + l_{CT}^{\delta_i}(d_{b_{i+1}}, d_{b_{i+1}}) \leq l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}}),$$

Thus,  $l_i \leq l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}}) + \rho(e_i)$ , which is also shown in Fig. 2(a).

If  $\rho(e_i) + w(e_i) + l_{CT}^{\delta_{i-1}}(v'_i, d_{b_{i+1}}) > l_{CT}^{\delta_{i-1}}(v'_i, d_{b_i}) + l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}})$ , it would go back to the previous destination  $d_{b_i}$ , move to  $d_{b_{i+1}}$  following a new shortest path, then continue following *CT* until next congestion  $e_{i+1}$ . We can get followings  $l_{CT}^{\delta_{i-1}}(d_{b_i}, v'_i) \leq l_{CT}^{\delta_{i-1}}(d_{b_i}, d_{b_{i+1}})$ ,  $l_G^{\delta_i}(v'_i, d_{b_i}) \leq l_G^{\delta_{i-1}}(d_{b_i}, d_{b_{i+1}})$  and

$$l_{CT}^{\delta_{i-1}}(d_{b_i}, d_{b_{i+1}}) + l_{CT}^{\delta_i}(d_{b_{i+1}}, d_{b_{i+1}}) \leq l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}})$$

Thus,  $l_i \leq l_{CT}^{\delta_i}(d_{b_{i+1}}, d_{b_{i+1}}) + l_{CT}^{\delta_{i-1}}(d_{b_i}, d_{b_{i+1}}) + l_G^{\delta_{i-1}}(d_{b_i}, d_{b_{i+1}}) + l_G^{\delta_i}(d_{b_i}, d_{b_{i+1}})$

Which is also shown in Fig. 2(b)

Obviously, the cost from depot  $s$  to  $d_{b_1}$  can be formulated as  $l_0 = l_{CT}^{\delta_0}(s, d_{b_1})$ . We further denote  $l_{GR}$  as the traversal cost by *Greedy Routing*. Clearly,  $l_{GR} = \sum_{i=0}^k l_i$ , and we analyze the relationship between  $l_{\Sigma}$  and the corresponding optimal traversal cost.

Case 1. All congestion  $e_i = \{v'_i, v''_i\}$  satisfies

$$\rho(e_i) + w(e_i) + l_{CT}^{\delta_{i-1}}(v''_i, d_{b_{i+1}}) \leq l_{CT}^{\delta_{i-1}}(v'_i, d_{b_i}) + l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}})$$

Recall that  $|ST|$  denote the weight (cost) of a simple Steiner tree *ST*. It's clearly that  $|ST| \leq OPT$ . Thus, we have

$$\begin{aligned} l_{GR} &= \sum_{i=0}^k l_i \\ &\leq l_{CT}^{\delta_0}(s, d_{b_1}) + \sum_{i=1}^k \left[ l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}}) + w^2(e_i) - w^1(e_i) \right] \\ &\leq l_{CT}^{\delta_k}(s, d_{b_1}) + \sum_{i=1}^k l_{CT}^{\delta_k}(d_{b_i}, d_{b_{i+1}}) + \sum_{i=1}^k [w^2(e_i) - w^1(e_i)] \\ &\leq 2|ST| + \sum_{i=1}^k [w^2(e_i) - w^1(e_i)] \\ &\leq 2OPT + \sum_{i=1}^k [w^2(e_i) - w^1(e_i)] \end{aligned}$$

The third inequality holds by Lemma 1, the last inequality holds by  $|ST| \leq OPT$ .

Case 2. There exists  $k_1 \leq k$  congested edges  $\{e_{i_1}, e_{i_2}, \dots, e_{i_{k_1}}\}$  that satisfy following inequality  $\rho(e_i) + w(e_i) + l_{CT}^{\delta_{i-1}}(v'_i, d_{b_{i+1}}) > l_{CT}^{\delta_{i-1}}(v'_i, d_{b_i}) + l_{CT}^{\delta_i}(d_{b_i}, d_{b_{i+1}})$ , while other congested edges don't. Total traversal cost can be formulated as

$$\begin{aligned}
l_{GR} &= \sum_{i=0}^k l_i \\
&\leq l_{CT}^{\delta_0}(s, d_{b_1}) + \sum_{j=1}^{k_1} \left[ l_{CT}^{\delta_j}(d_{b_j}, d_{b_{j+1}}) + \rho(e_{ij}) \right] \\
&\quad + \sum_{j \in \{1, 2, \dots, k\}} \left[ l_{CT}^{\delta_j}(d_{b_j}, d_{b_{j+1}}) + 2l_{CT}^{\delta_j}(d_{b_j}, d_{b_{j+1}}) \right] \\
&\quad \quad \quad j \neq i_1, \dots, i_{k_1} \\
&\leq l_{CT}^{\delta_0}(s, d_{b_1}) + \sum_{j=1}^k l_{CT}^{\delta_k}(d_{b_j}, d_{b_{j+1}}) + 2 \sum_{j=1}^k l_G^{\delta_k}(d_{b_j}, d_{b_{j+1}}) \\
&\leq 2|ST_{\delta_k}| + 2k \cdot \max_{1 \leq j \leq k} \left\{ l_G^{\delta_k}(d_{b_j}, d_{b_{j+1}}) \right\} \\
&\leq (k+2) \cdot OPT
\end{aligned}$$

in which the second inequation holds for that

$$\sum_{j=1}^{k_1} \rho(e_{ij}) + \sum_{j \in \{1, 2, \dots, k\}} 2l_G^{\delta_j}(d_{b_j}, d_{b_{j+1}}) \leq 2 \sum_{j=1}^k l_G^{\delta_k}(d_{b_j}, d_{b_{j+1}})$$

Case 1 and case 2 prove Lemma 1.

**Theorem 3.** For online STSP-CDT problem, algorithm Greedy Routing runs in polynomial time, and when the times of online traffic congestion is up to a given constant  $k$ , Greedy Routing is  $\min\{k+2, \rho_\Sigma + 2\}$ -competitive, where  $\rho_\Sigma = \frac{\sum_{e \in \delta} \rho(e)}{OPT}$ .

Proof. Step 1 to Step 3 in algorithm Greedy Routing can be done in  $O(|v|^3)$ . In each iteration in Step 4, a shortest path  $l_G^{\delta_i}(d_{i+1}^b, d_{i+1}^a)$  is computed in  $O(|v|^2)$  time. Either pass through the blocked edge  $e_{i+1} = \{v'_{i+1}, v''_{i+1}\}$  or travel along  $l_G^{\delta_{i+1}}(d_{i+1}^b, d_{i+1}^a)$ , vehicle would continue following CT which can be done at most in  $O(|v|^2)$  time as well. Thus, Step 4 in algorithm Greedy Routing can be done in  $O(|v| + k|v|^2)$ . It follows that the overall running time is in  $O(\max\{|v| + k\} \cdot |v|^2)$ .

If the ratio  $r < k$  initially, then delivery vehicle chooses Strategy 1 that follows the path CT, and the total traversal cost is at most

$$l_{GR} < l_{CT}^{\delta_0}(d_0, d_{n+1}) + l_{CT}^{\delta_0}(d_0) \leq 2|ST| + k|ST| \leq (k+2)OPT$$

And the competitive ratio is at most  $c_{GR} = \frac{l_{GR}}{OPT} \leq k+2$ .

If vehicle follows the path CT until the next congestion. Assume without loss of generality that  $r < k - i$  firstly hold when vehicle arrives at  $v_c$  before learning the next blocked edge  $e_i = \{v'_i, v''_i\}$ , for some iteration  $i$ . Then total cost can be formulated as follows

$$\begin{aligned}
l_{GR} &= \sum_{j=0}^{i-1} l_j + l_{CT}^{\delta_{i-1}}(d_b, v'_i) + l_{CT}^{\delta_i}(d_{b_{i+1}}, d_{n+1}) + l_{CT}^{\delta_i}(d_{b_{i+1}}) \\
&\quad + \min \left\{ l_{CT}^{\delta_i}(v'_i, d_{b_{i+1}}) + \rho(e_i), l_G^{\delta_{i-1}}(v'_i, d_{b_i}) + l_G^{\delta_i}(d_{b_i}, d_{b_{i+1}}) \right\}
\end{aligned}$$

take notice that

$$\begin{aligned} & \min \left\{ l_{CT}^{\delta_i}(v'_i, d_{b_i+1}) + \rho(e_i), l_G^{\delta_{i-1}}(v'_i, d_{b_i}) + l_G^{\delta_i}(d_{b_i}, d_{b_i+1}) \right\} \\ & \leq l_G^{\delta_{i-1}}(v'_i, d_{b_i}) + l_G^{\delta_i}(d_{b_i}, d_{b_i+1}) \end{aligned}$$

and that

$$\begin{aligned} \sum_{j=0}^{i-1} l_j & \leq l_{CT}^{\delta_{i-1}}(d_0, d_{b_i}) + 2(i-1) \max_{1 \leq j \leq i-1} l_G^{\delta_{i-1}}(d_{b_j}, d_{b_j+1}) \\ & \leq l_{CT}^{\delta_i}(d_0, d_{b_i}) + 2(i-1) \frac{OPT}{2}. \end{aligned}$$

Thus, we further have that

$$\begin{aligned} l_{GR} & \leq l_{CT}^{\delta_i}(d_0, d_{b_i}) + 2(i-1) \frac{OPT}{2} + l_{CT}^{\delta_i}(d_{b_i}, d_{b_i+1}) + 2l_G^{\delta_i}(d_{b_i}, d_{b_i+1}) \\ & \quad + l_{CT}^{\delta_i}(d_{b_i+1}, d_{n+1}) + lw_{CT}^{\delta_i}(d_{b_i+1}) \\ & \leq l_{CT}^{\delta_i}(d_0, d_{n+1}) + 2(i-1) \frac{OPT}{2} + 2l_G^{\delta_i}(d_{b_i}, d_{b_i+1}) + lw_{CT}^{\delta_i}(d_{b_i+1}) \\ & \leq 2|ST| + 2(i-1) \frac{OPT}{2} + OPT + (k-i)ST_{E_i} \\ & \leq (k+2)OPT \end{aligned}$$

The competitive ratio is at most  $c_{GR} = \frac{l_{GR}}{OPT} \leq k+2$ .

Otherwise, if  $r > k-i$  holds for each iteration  $i$  during the trip. Then, total cost is at most  $\min \left\{ (k+2)OPT, 2OPT + \sum_{i=1}^k \rho(e_i) \right\}$ . The competitive ratio is at most

$$c_{GR} \leq \frac{\min \left\{ (k+2)OPT, 2OPT + \sum_{i=1}^k \rho(e_i) \right\}}{OPT} \leq \min \{ k+2, \rho_{\Sigma} + 2 \}$$

where  $\rho_{\Sigma} = \sum_{i=1}^k \rho(e_i)$ .

This proves the theorem.

## 5 Conclusion

This paper studies single vehicle’s delivery strategy in urban traffic network with traffic congestion of certain delay time. Recall assumptions in previous research that either there is no traffic congestion or traffic congestion is not recoverable. In this paper, we study a more practical scenario that traffic congestion is recoverable, meaning that traffic congestion can recover after some delay time. We assume delay time of traffic congestion is certain to delivery vehicle on arrival at the first node of the edge. When the number of online congested edges is up to a given constant  $k$ , we firstly present a



lower bound of  $\min\{1+k, 1+\rho_\Sigma\}$  on competitive ratio for the problem, where  $\rho_\Sigma$  is the ratio compared sum of delay time of online congestion with cost of optimal route. We further present an online polynomial-time algorithm Greedy Routing with its competitive ratio no more than  $\min\{2+k, 2+\rho_\Sigma\}$ .

Theoretically, it would be interesting to design deterministic algorithm for *STSP* problem with online uncertain congestion where delay time of congestion is random.

**Acknowledgments.** This work was partially supported by the NSFC (Grant No. 71601152), and China Postdoctoral Science Foundation (Grant No. 2016M592811).

## References

1. Cornuéjols, G., Fonlupt, J., Naddef, D.: The traveling salesman problem on a graph and some related integer polyhedra. *Math. Program.* **33**, 1–27 (1985)
2. Fleischmann, B.: A cutting plane procedure for the travelling salesman problem on road networks. *Eur. J. Oper. Res.* **21**, 307–317 (1985)
3. Ratliff, H.D., Rosenthal, A.S.: Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Oper. Res.* **31**, 507–521 (1983)
4. Papadimitriou, C.H., Yannakakis, M.: Shortest paths without a map. *Theor. Comput. Sci.* **84**, 127–150 (1991)
5. Orloff, C.S.: A fundamental problem in vehicle routing. *Networks* **4**, 35–64 (1974)
6. De Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: a literature review. *Eur. J. Oper. Res.* **182**, 481–501 (2007)
7. Zhang, H., Tong, W., Xu, Y., Lin, G.: The Steiner traveling salesman problem with online edge blockages. *Eur. J. Oper. Res.* **243**, 30–40 (2015)
8. Bar-Noy, A., Schieber, B.: The Canadian traveller problem. In: *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 261–270 (1991)
9. Westphal, S.: A note on the k-Canadian traveller problem. *Inf. Process. Lett.* **106**, 87–89 (2015)
10. Fu, L.: An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transp. Res. Part B* **35**(8), 749–765 (2001)
11. Bender, M., Westphal, S.: An optimal randomized online algorithm for the k-Canadian traveller problem on node-disjoint paths. *J. Comb. Optim.* **30**(1), 87–96 (2015)
12. Liao, C.S., Huang, Y.: Generalized Canadian traveller problems. *J. Comb. Optim.* **29**(4), 701–712 (2015)
13. Zhang, H., Tong, W., Xu, Y., Lin, G.: The Steiner traveling salesman problem with online advanced edge blockages. *Comput. Oper. Res.* **70**, 26–38 (2016)



# The Complexity of Weak Consistency

Gaoang Liu<sup>1,2</sup>(✉)  and Xiuying Liu<sup>2,3</sup>

<sup>1</sup> State Key Laboratory of Computer Science, Institute of Software,  
Chinese Academy of Science, Beijing, China

Gaoang@ios.ac.cn

<sup>2</sup> University of Chinese Academy of Sciences, Beijing, China

<sup>3</sup> Wispirit Technology Ltd., Beijing, China

Liuxiuying@66nao.com

**Abstract.** Weak consistency is a memory model that is frequently considered for shared memory systems. Its most distinguishable feature lies in a category of operations in two types: data operations and synchronization operations. For highly parallel shared memory systems, this model offers greater performance potential than strong models such as sequential consistency by permitting unconstrained optimization on updates propagation before synchronization is invoked. It captures the intuition that delaying updates produced by data operations before triggering a synchronization operation does not typically affect the program correctness.

To formalize the connection between concrete executions and the corresponding specification, we propose in this work a new approach to define weak consistency. This formalization, defined in terms of distributed histories abstracted from concrete executions, provides an additional perception of the concept and facilitates automatic analysis of system behaviors. We then investigate the problems on verifying whether implementations have correctly implemented weak consistency. Specifically, we consider two problems: (1) the testing problem that checks whether *one* single execution is weakly consistent, a critical problem for designing efficient testing and bug hunting algorithms, and (2) the model checking problem that determines whether *all* executions of an implementation are weakly consistent. We show that the testing problem is NP-complete, even for finite processes and short programs. The model checking problem is proven to be undecidable.

## 1 Introduction

Many modern computer architectures and multiprocessors support shared memory in hardware, a design that facilitates fast access and provides user-friendly programming perspective to memory. A shared memory system permits concurrent accesses from multiple processes to a single address space. To avoid undesirable behaviors, memory consistency must be properly maintained. Informally, a memory consistency specifies the guarantees that a system makes on the value of read operations from the shared memory. Strong models such as atomic consistency (also known as linearizability) [1] and sequential consistency (SC) [2]

are intuitively composed but restrictive in performance as they would severely restrict the set of possible optimization, such as pipelining write accesses and caching write operations. Also, implementing these strong consistency criteria in message-passing system is very expensive. For better performance, in the past several decades many weaker consistency models, for example weak consistency (WC) [3], causal consistency (CC) [4] and PRAM consistency [5], have been proposed, explored and revised as various attempts.

Weak consistency was proposed by Dubois et al. [3] as *weak ordering*, with a motivation to address the logical problems of memory accesses buffering arose from multiprocessor systems. Adve and Hill generalized weak consistency to the order strictly necessary for programmers with “SC for DRF” [6], which was re-specified and served as the cornerstone for Java memory model [7] and C++ [8]. A weak consistent system distinguishes two types of variables: shared program variables that are visible to all processes; and synchronization variables for concurrent executions synchronization. The latter can be recognized by instructions such as TEST\_AND\_SET (TAS), COMPARE\_AND\_SWAP (CAS), or special LOAD and STORE instructions. In the Java memory model, for example, to allow synchronization races but not data races, variables having potential to race must be tagged as synchronization, using keywords such as **volatile** or **atomic**. Programmers can also create synchronization locks implicitly with Java’s monitor-like **synchronized** methods.

Based on the type of variable it accesses, an operation in a weakly consistent system is either a data operation and a synchronization operation. The latter works in a way similar to fence instructions, whose initiation suggests all previous references to the shared variables have completed and all future references have to wait. The idea of exploiting fence operations to achieve synchronized concurrency is also widely adopted by many commercial models. For example, the Power model [9] implemented by IBM PowerPC is similar in spirit to weak consistency by utilizing varieties of fences to synchronize concurrently executing processes. PowerPC uses an instruction called *Hwsync* to keep all writes in a consistent total order, it can also achieve sequentially consistent behaviors if *Hwsync* is used together with address dependencies and message passing [10]. In addition to PowerPC, another commercial example implementing a model that relies heavily on proper synchronization is ARM multiprocessor [11]. Similarly, ARM has multiple flavors of fences, including one data memory barrier that can order all memory accesses.

Generally, developing distributed implementations to satisfy a relaxed consistency model is very challenging because of the complicated issues related to communication. At different stages of development, developers rely on different testing and verification techniques to validate the system they have built so far. This highlights the need for more research on the feasibility and efficiency of algorithms that are utilized during the development. We thus investigate in this paper two fundamental problems: (1) the testing problem that asks whether one given execution of an implementation is weakly consistent, and (2) the model checking problem (or verifying problem) that asks whether all the executions

of an implementation are weakly consistent. The behaviors of implementations we consider in this study are restricted to the setting of **read/write** memory (RWM) abstraction, which is at the base of many distributed data structures used in practice.

We show that the complexity of the testing problem is NP-complete. The is achieved by reducing the serialization problem, a NP-complete problem for database transactions to TWC-read problem, which is a restricted version of the testing problem. We also investigate the problems with limited accesses per process and limited processes in an instance. Both problems are proven to be NP-complete, even when the access number is limited to two and the process number to three. For the TWC-read problem restricted to two processes, we prove there exists a polynomial algorithm. We then prove that the model checking problem is undecidable by a reduction from the Post Corresponding Problem (PCP). The idea is to relate each PCP instance to a WC implementation, such that the instance has a valid witness exactly when the implementation is weakly consistent. Due to space constraints, we defer some proofs to the long version [12].

The remainder of this paper is organized as follows. Section 2 presents the notion of distributed history and the specification for **read/write** memory. Section 3 briefly reviews the intuition behind WC and then shows how to formalize this concept. We present in Sects. 4 and 5 the complexity results for the testing problem. The decidability result of the model checking problem is proved in Sect. 6. Section 7 discusses the related work. Finally, Sect. 8 concludes the paper.

## 2 Preliminaries

### 2.1 Sets, Relations and Labeled Posts

Given a set  $\mathbb{E}$  and a relation  $R \subseteq \mathbb{E} \times \mathbb{E}$ , we denote by  $e_1 \prec_R e_2$  the fact that  $(e_1, e_2) \in R$ . We write  $(R)^+$  to denote the *transitive closure* of  $R$ . A relation is a *strict partial order* if it is transitive and irreflexive.

A *poset* is a pair  $(\mathbb{E}, \prec)$  where  $\prec$  is a strict partial order. Given a set  $\Sigma$ , a  $\Sigma$  *labeled poset*  $\rho$  is a tuple  $(\mathbb{E}, \prec, \iota)$  where  $(\mathbb{E}, \prec)$  is a poset and  $\iota : \mathbb{E} \rightarrow \Sigma$  is the labeling function.

We introduce a relation on labeled posets, denoted  $\triangleright$ . Let  $\sigma = (\mathbb{E}, \prec, \iota)$ ,  $\sigma' = (\mathbb{E}, \prec', \iota')$  be two posets labeled by the same set  $\Sigma$ . We denote by  $\sigma \triangleright \sigma'$  the fact that  $\sigma$  imposes less constraints on operations in  $\mathbb{E}$ . Formally,  $\sigma \triangleright \sigma'$  if  $\prec \subseteq \prec'$  and  $\iota = \iota'$ , i.e., for all operation  $e \in \mathbb{E}$ ,  $\iota(e) = \iota'(e)$ .

### 2.2 Histories and Specifications

A distributed system is a composition of a set of processes/participants invoking methods on shared objects (registers, queues, etc.). An object implements a programming interface (API) defined by a set of *methods*  $\mathbb{M}$  with input and output from a data domain  $\mathbb{D}$ . The behaviors of a system can be characterized by a set

of executions. Each execution is a labeled poset consist of a collection of events, representing operation invocations by different participants, and a relation on events, describing abstractly how the system processes the corresponding operations. These executions can be characterized by *distributed histories*. Formally,

**Definition 1.** *A distributed history is a poset  $\mathcal{H} = \{\mathbb{E}, \prec_{\text{PO}}, \iota\}$  labeled by  $\mathbb{M} \times \mathbb{D} \times \mathbb{D}$ , where  $\mathbb{E}$  is a countable set of events;  $\prec_{\text{PO}} \subset (\mathbb{E} \times \mathbb{E})$  is a strict partial order called program order; and  $\iota : \mathbb{E} \rightarrow (\mathbb{M} \times \mathbb{D} \times \mathbb{D})$  is a labeling function.*

The labeling function  $\iota$  maps each event to its corresponding operation that invoke a method from  $\mathbb{M}$ . To model events that return no value, we use a value  $\perp \in \mathbb{D}$ , which is often omitted for better readability.

The consistency of shared objects is a criterion that links the distributed executions to a particular specification, which characterizes the admissible behaviors for a program that uses the objects. To model the `read/write` memory, a sequential specification will suffice. We define the specification  $S_{r/w}$  as a set of sequences where each value read matches the most recent write. Formally, it is the smallest set of sequences by inductively adopting the following rules:

- $\epsilon \in S_{r/w}$ ,
- $\sigma \cdot w(v, d) \in S_{r/w}$  if  $\sigma \in S_{r/w}$ ,
- $\sigma \cdot r(v, 0) \in S_{r/w}$  if  $\sigma \in S_{r/w}$  and  $\sigma$  contains no write on  $v$ ,
- $\sigma \cdot r(v, d) \in S_{r/w}$  if  $\sigma \in S_{r/w}$  and the last write on  $v$  in  $\sigma$  is  $w(v, d)$ ,

where  $v \in \mathbb{V}$ ,  $d \in \mathbb{D}$  and  $\epsilon$  is the empty sequence.

### 3 Weak Consistency

#### 3.1 Weak Consistency: Informal Description

Dubois et al. [3] introduced the concept of WC by enforcing on storage accesses the following constraints:

**Definition 2.** *In a multiprocessor system, storage accesses are weakly ordered if (1) accesses to synchronizing variables are strongly ordered, (2) no access to a synchronizing variable is allowed to be performed until all previous writes have completed everywhere, and (3) no data access (read or write operation) is allowed to be performed until all previous accesses to synchronizing variables have been performed everywhere.*

Weak consistency categorizes memory accesses into two types, and imposes different ordering constraints based on the access type. For data operations, buffered requests to memory are allowed to pass each other in store buffer, a behavior referred to as *jockeying* [3] and is often permitted between requests for different memory locations. The synchronization operations are requested to be *strongly ordered* [3], but jockeying with data operations is forbidden. Informally, weak consistency ensures that reordering memory operations to shared data between synchronization operations does not typically affect program correctness.

*Example 1.* History (a) in Fig. 1 is weakly consistent while (b) is not. Because of jockeying, it is possible for  $P_3$  in (a) to read the initial value of  $x$  even when a previous read on  $y$  returns the latest value. In (b), however, the sync operation forces out all previous updates to  $P_2$ , making the read  $r(x, 0)$  impossible.

$$\begin{array}{lll}
 P_1: & w(x, 1) & w(y, 1) \\
 P_2: & r(y, 1) & \text{SYNC} & r(x, 1) \\
 P_3: & r(y, 1) & r(x, 0) \\
 & & \text{(a)}
 \end{array}
 \qquad
 \begin{array}{lll}
 P_1: & w(x, 1) & \text{SYNC} & w(y, 1) \\
 P_2: & r(y, 1) & r(x, 0) \\
 & & \text{(b)}
 \end{array}$$

**Fig. 1.** The visualization of WC, where SYNC represents a synchronization operation on a variable other than  $x$  and  $y$ . Initially,  $x = y = 0$ .

### 3.2 Weak Consistency: A Formal Definition

We present here a characterization of concrete executions of concurrent systems implementing weak consistency. This characterization links each execution to a set of sequences, among which there is at least one can provide a reasonable explanation as to why the execution is weakly consistent.

Given a history  $\mathcal{H}$  with an event set  $\mathbb{E}$ , we denote by  $\mathbb{E}_S$  the set of synchronization events and  $\mathbb{E}_D$  the set of data events. Two events from different processes can be ordered only if there exists an intervening synchronization between them. To capture this property, we introduce *happen-before* relations for events in any history. Two types of relation are considered: program order  $\prec_{PO}$ , and synchronization order  $\prec_{SO}$ . Formally, let  $e_u$  and  $e_v$  be any two operations occurring in  $\mathcal{H}$ . Then:

- $e_u \prec_{PO} e_v$  iff  $e_u$  occurs before  $e_v$  in the same process.
- $e_u \prec_{SO} e_v$  iff  $e_u, e_v \in \mathbb{E}_S \wedge \text{var}(e_u) = \text{var}(e_v)$  and  $e_u$  is performed before  $e_v$ ,

where  $\text{var}(e) \in \mathbb{V}$  is the variable accessed by  $e$ . The concept of *performed* is borrowed from [6], where a read is said to be *performed* at a point in time when no subsequent write, from the same or another process, can affect the value returned. Similarly, a write *performed* when all subsequent reads return the written value until another write to the same memory location is performed.

**Definition 3 (Weak Order).** *A weak order  $\prec_{WO}$  of a history  $\mathcal{H}$  is the irreflexive transitive closure of program order and synchronization order, that is  $\prec_{WO} = (\prec_{PO} \cup \prec_{SO})^+$ . The set of events happening before  $e$  w.r.t. the  $\prec_{WO}$  is denoted by  $[e] = \{e' \in \mathbb{E} : e' \prec_{WO} e\}$ .*

Intuitively,  $[e]$  is the *weak past* of  $e$ , i.e., the set of operations whose effects are visible to  $e$ . To associate any weakly consistent history to the sequential specification, we need to define a way to explain how events on a history are generated.

For example, it should explain which write is responsible for a read that is accessing the same variable. This is achieved by defining a function, called *observation function* below, that links each event to an event set, which, if considered together with  $\prec_{\text{WO}}$ , will be sufficient to explain why the history is admissible by the specification  $S_{r/w}$ .

**Definition 4.** Given a history  $\mathcal{H}$ , a function  $\tau : \mathbb{E} \rightarrow 2^{\mathbb{E}}$  is an *observation function* on  $\mathbb{E}$ , if  $\forall e \in \mathbb{E}, \tau(e) = \{e\} \cup [e] \cup C_e$  for some set  $C_e \subseteq \mathbb{E}$  whose elements are concurrent events of  $e$ . The set  $\tau(e)$  is called an *observation set* of  $e$ .

The observation set  $\tau(e)$  is actually a snapshot of updates observed by  $e$ . It should confirm to all constraints imposed by WC. Apart from the constraints required by  $\prec_{\text{WO}}$ , it should also respect what we call *update inheritance* — updates observed by one event are inherited to its successors. Specifically, if an event  $e$  observes updates happened before a synchronization, then all events observed  $e$  should also have observed those updates. Formally, for two events  $e_u, e_v \in \mathbb{E}$ , if

- (a)  $e_u \in \tau(e_v)$  and at least one of  $e_u, e_v$  belongs to  $\mathbb{E}_S$ , or
  - (b)  $e_u \in \mathbb{E}_S \wedge \exists e_w. (e_w \in \tau(e_v) \wedge e_u \in \tau(e_w))$ ,
- (\*)

then  $\tau(e_u) \subseteq \tau(e_v)$ . We call such a function a *observation closed function* (OCF).

**Definition 5.** A history  $\mathcal{H}$  is *weakly consistent with respect to  $S_{r/w}$*  if there exists an OCF  $\tau$  such that for any  $e \in \mathbb{E}$ , there exists a sequence  $\sigma_e \in S_{r/w}$  such that  $(\tau(e), \prec_{\text{WO}}, \iota) \triangleright \sigma_e$ .

*Example 2.* To illustrate, the history in Fig. 1(b) has no OCF  $\tau$  for its event set. For otherwise we have  $e_{w(x,1)} \in \tau(e_{\text{SYNC}})$  and  $\tau(e_{\text{SYNC}}) \subseteq \tau(e_{r(x,1)})$ , and by the condition (\*) the observation set  $\tau(e_{r(x,0)})$  will contain  $e_{w(x,1)}$ , implying the update on  $x$  is observable to  $e_{r(x,0)}$ , whose return value should thus be 1 instead of 0.

## 4 The Testing Problem of Weak Consistency

We first investigate the testing problem of weak consistency (TWC), which is relevant for instance in the context of testing a given distributed object. We show this problem is NP-complete by a reduction from the serializability problem for database histories.

The membership in NP can be easily proved, this follows from the fact that, for any given instance (history)  $\mathcal{H}$ , one can guess an observation function  $\tau$ , and a sequence  $\sigma_e \in S_{r/w}$  for each event  $e$ , and then check in polynomial time whether  $\tau$  is an OCF and the relation  $\triangleright$  in Definition 5 holds.

To prove the NP-hardness, we first define a restricted version of the TWC problem and reduce the restricted problem to the serializability problem. We consider the case in which for each read, it is known precisely which write was responsible for the value read. We call this the TWC-read problem. The function mapping each read to the responsible write is called a *read-mapping*.

The serializability problem for database transactions is one exploring the existence of a schedule that is equivalent to one that executes the transactions serially in some order. One major type of the problem is view serializability [13], in which we are given a history, a total order on a set of reads and writes, where each read or write is associated with a particular transaction. The problem asks if there is a total order on the transactions that preserves the reads-from mapping of the original history. The view serializability problem is NP-complete [13]. TWC-read is a problem more general than view serializability by permitting solutions in which the accesses for a processor (transaction) are in order but may not be consecutive. To illustrate, the instance in Fig. 2 is a “yes” instance for TWC-read problem, but a “no” instance for view serializability, because both  $P = P_1P_2$  and  $P = P_2P_1$  have reads-from violations.

$$\begin{aligned} P_1 &: w(x, 0), w(y, 1), r(x, 1) \\ P_2 &: r(y, 1), w(x, 1) \end{aligned}$$

**Fig. 2.** A “yes” instance of the TWC-read problem

**Lemma 1.** *The TWC-read problem is NP-complete.*

*Proof.* Given  $\mathcal{H}$ , an instance of a view serializability problem, we construct an instance of TWC-read as follows. Let  $v$  be a variable not accessed by any operation in  $\mathcal{H}$ , and  $\#_\alpha$  a synchronization operation on a variable  $\alpha$ . Let  $P_i$  be the sequence of operations in  $\mathcal{H}$  for transaction  $i$  ( $i \in \{1, \dots, n\}$ ), where each write in a transaction is assigned a unique value to write, and each read is assigned the value of the closest previous write to the same address in  $\mathcal{H}$ .

For all transactions  $i$ , let  $P'_i = w(v, i)\#_\alpha P_i\#_\alpha r(v, i)$ . Our TWC-read instance is  $\mathcal{H}' = \parallel_{i \in \{1, \dots, n\}} P'_i$ . The intervening  $\#_\alpha$  guarantees the update to  $v$  is observed by other transactions before  $P_i$  initiates and updates from other transactions were brought in before reading  $v$ . This construction ensures that for each  $P'_i$  once the write  $w(v, i)$  starts, the remainder (i.e.  $\#_\alpha P_i\#_\alpha r(v, i)$ ) must be scheduled consecutively. If two transactions  $P'_i, P'_j$  interleaves, then at least one of them, say  $P'_i$ , will return for its last read a value other than  $i$ , and thus violates the read-from relation. It follows that  $\mathcal{H}'$  is in TWC-read if and only if  $\mathcal{H}$  is view serializable.  $\square$

The above result on TWC-read implies that the general TWC problem is at least NP-hard, since the problem is also in NP, we have the following theory.

**Theorem 1.** *Checking whether a distributed history  $\mathcal{H}$  is weakly consistent with respect to  $S_{r/w}$  is NP-complete.*

## 5 Restricted TWC Problems

The previous section shows that the TWC problem is generally NP-complete. In this section, we investigate two restricted TWC problems that consider only



instances with limited number of accesses or processes. Such problems are interesting because many multiprocessors usually have a small number of processors, e.g., 8, 16 or 32; and when it comes to testing, the size of instances are usually small.

We show that these restricted problems remain NP-complete, even when the number of accesses per process is limited to two and the number of processes to three, which implies the testing problem of weak consistency is intrinsically hard. For a very rare case of the problem, in which only two processes are involved and a read-mapping is provided, we prove there exists a polynomial algorithm.

### 5.1 Restricting the Number of Accesses

We now investigate the TWC problem in which each process perform at most two data accesses and each data variable is written to at most twice. We show this problem is NP-complete.

The result is proved by a reduction from 3SAT. Let  $\mathcal{F}$  be a 3SAT instance. For a literal  $l_i$  in a clause, we use the notation  $\mathcal{B}(l_i)$  to represent  $T$  (**True**) if  $l_i$  is a positive literal (i.e., a variable), and  $F$  (**False**) otherwise. We need to simulate the logical connectives (i.e., an OR and an AND), as well as an assignment of variables that remains in effect until the formula is evaluated. We observe that (1) a read must wait for its responsible write to occur, (2) the second access at a process must wait for the first. Then the assignment to  $v_i$  can be simulated by following columns, of which each one represents a sequence.

$$\begin{array}{cc} w(v_i, T) & r(x, 1) & w(v_i, F) & r(x, 1) \\ & \#_\alpha & & \#_\alpha \\ & r(v_i, T) & & r(v_i, F), \end{array}$$

where  $x$  is initially 0 and  $\#_\alpha$  a synchronization operation on variable  $\alpha$ .

An OR is simulated by separating the literals of a clause into three reads, whose executions determine the truth value of that clause. For each clause,  $C_i = l_p \vee l_q \vee l_r$ , we have four sequences:

$$\begin{array}{cccc} r(l_p, \mathcal{B}(l_p)) & r(l_q, \mathcal{B}(l_q)) & r(l_r, \mathcal{B}(l_r)) & r(d_i, T) \\ w(d_i, T) & w(d_i, T) & w(c_i, T) & w(c_i, T) \end{array}$$

By the above two observations, this ensures that  $C_i$  can not be true unless clause  $i$  is satisfied by the guessed truth assignment.

The AND relation of clauses can be easily simulated by a single sequence  $r(c_1, T), r(c_2, T), \dots, r(c_m, T), w(x, 1)$ . But this sequence involves  $m + 1 > 2$  data operations. To have only two accesses per process, we separate this sequence into  $m + 1$  sequence: a single write  $w(x, 1)$  and  $m$  sequences ending with a read  $r(x, 0)$ :

$$\begin{array}{cccc} w(x, 1) & r(c_1, T) & r(c_2, T) & \dots & r(c_m, T) \\ & \#_\alpha & \#_\alpha & \dots & \#_\alpha \\ & r(x, 0) & r(x, 0) & \dots & r(x, 0) \end{array}$$

The above write  $w(x, 1)$  is possible only when every read on  $C_i$  has returned true, that is all clauses have been satisfied by the guessed assignment. This ensures the initial assignment to each  $v_i$  remain in effect until the satisfiability of  $\mathcal{F}$  has been simulated.

**Lemma 2.** *Let  $\mathcal{F}$  be an instance of 3SAT, and  $\mathcal{W}$  the instance of the TWC problem constructed as described above. Then  $\mathcal{W}$  is weakly consistent if and only if  $\mathcal{F}$  is satisfiable.*

*Proof (sketch).* Assume there is a satisfying assignment for  $\mathcal{F}$ . We can construct a corresponding schedule for  $\mathcal{W}$  such that it can be sequentially ordered by that schedule. Then  $\mathcal{W}$  is sequentially consistent and, hence, weakly consistent. Conversely, if  $\mathcal{W}$  is weakly consistent, the first value written to each variable  $v_i$  forms the satisfying assignment.  $\square$

By this lemma, the following result is straightforward.

**Theorem 2.** *The TWC problem, restricted to instance in which each sequence contains at most two data operations and each data variable occurs in at most two write operations, is NP-complete.*

## 5.2 Restricting the Number of Processes

We have shown that the TWC problem is NP-complete with  $O(n)$  processes ( $n$  is the number of data variables). In this problem, the number of processes per instance grows proportionally with the number of variables, this raises another question: what is the complexity for TWC problems with a fixed small number of processes. This question is answered by the theory below, showing that the problem remains NP-complete even when the number of processes per instance is limited to three.

To simulate a 3SAT instance, the proof of Lemma 2 constructs a disjoint set of processes for each variable. This strategy can not be transferred here as the number of processes is limited. We consider to analyze the problem with a reduction from 1-in-3 SAT problem, a variant of 3SAT where the input instance is the same, but the question is to determine whether there exists a satisfying assignment so that exactly one literal in each clause is set to true. This problem is known to be NP-complete. A monotone version of this problem, positive 1-in-3 SAT, where each clause contains only positive literals, remains NP-complete. For a given positive 1-in-3 SAT instance  $\mathcal{F}$ , we construct an instance  $\mathcal{W}$  of the TWC problem, using only three processes, such that  $\mathcal{F}$  has a satisfying assignment exactly when  $\mathcal{W}$  is weakly consistent. The TWC instance  $\mathcal{W}$  for each  $\mathcal{F}$  is depicted in Fig. 3.

The synchronization parts (as illustrated by # in the figure) separate the construction into  $m + 1$  stages. The key idea behind this construction is to use the value at the end of the (INIT) stage as the assignment. The history is weakly consistent if every stage is weakly consistent. For each clause, each of the three processes is satisfied by a particular assignment. The subtle part is the writes

	Processor 1	Processor 2	Processor 3
(INIT)	$w(v_1, F)$ $\dots$ $w(v_n, F)$	$w(v_1, T)$ $\dots$ $w(v_n, T)$	
(#)	$w(u, 1)$ $\#_\alpha$ $r(u, 3)$	$w(u, 2)$ $\#_\alpha$ $r(u, 3)$	$r(u, 1)$ $r(u, 2)$ $\#_\alpha$ $w(u, 3)$
( $C_1$ )	$r(v_{x_1}, T)$ $r(v_{y_1}, F)$ $r(v_{z_1}, F)$ $w(v_{x_1}, F)$ $w(v_{y_1}, T)$	$r(v_{y_1}, T)$ $r(v_{z_1}, F)$ $r(v_{x_1}, F)$ $w(v_{y_1}, F)$ $w(v_{z_1}, T)$	$r(v_{z_1}, T)$ $r(v_{x_1}, F)$ $r(v_{y_1}, F)$ $w(v_{z_1}, F)$ $w(v_{x_1}, T)$
(#)	...	...	...
( $C_m$ )	...	...	...

**Fig. 3.** Transforming an instance of positive 1-in-3 SAT to an instance of TWC. The 3SAT instance contains  $n$  variables  $v_1, \dots, v_n$  and  $m$  clauses:  $C_1, \dots, C_m$ , where  $C_i = (v_{x_i} \vee v_{y_i} \vee v_{z_i})$  for some  $x_i, y_i, z_i \in [1..n]$ . (Color figure online)

highlighted in blue. Once a way of satisfying a clause is settled, the writes free up the other two processes by negating variables, and then return all variables to their initial setting (for the next stage). Conversely, for any assignment of  $\mathcal{F}$  that does not satisfy this clause, there is no way to prove the weak consistency of this stage.

**Theorem 3.** *The TWC problem restricted to three processes is NP-complete.*

In the above reduction, each clause of the 3SAT instance requires at least three processes for the simulation procedure. This is the simplest reduction we are aware of, leaving open the TWC problem restricted to two processes. Nevertheless, we prove below there is a polynomial algorithm for TWC-read problem restricted to two processes.

### 5.3 TWC-Read Problem with Two Processes is in P

For an instance  $\mathcal{H}$  of two-processes TWC-read problem, every read in  $\mathcal{H}$  knows precisely its responsible write, which means if two writes  $w(v, d_1), w(v, d_2)$  are accessing the same location  $v$ , then  $d_1$  and  $d_2$  must differ. To solve the two-processes TWC-read problem, we begin by constructing an OCF,  $\tau$ , capturing all events observed by each access. The instance  $\mathcal{H}$  is in TWC-read exactly when there exists a witness OCF that respects certain constraints, and involves none of the anomalous forms (as defined in the proof for Lemma 3). Constructing this OCF and checking whether certain conditions are met can be done in polynomial time. That is, there is a polynomial algorithm to solve this problem in

two steps: (1) constructing for an instance a witness OCF, (2) checking whether this OCF meets required conditions.

**Lemma 3.** *The TWC-read problem restricted to two processes is in P.*

## 6 Undecidability of Verifying Weak Consistency

We consider in this section the model checking problem of weak consistency. For systems where memory coherence is properly maintained, WC implies SC under the assumption that every memory access is synchronized by the same sync variable [14]. This implies the model checking problem of WC is undecidable since the same problem of SC is generally undecidable [15]. Nevertheless, we offer here another proof with PCP reduction, which may provide some additional insight into the reason why verifying weak consistency is undecidable. For the sake of argument, we assume the implementations are regular and specification is restricted to  $S_{r/w}$  with a fixed number of variables whose domain sizes are also fixed, and thus the specification corresponds to a particular regular language.

**Definition 6.** *Let  $\Sigma$  be an alphabet with at least two letters. An instance of PCP is given by two sequences  $U = \{u_1, \dots, u_n\}$  and  $V = \{v_1, \dots, v_n\}$  of words over  $\Sigma$ . The problem is to determine whether there is a sequence  $(i_1, \dots, i_p)$  with  $i_j \in \{1, \dots, n\}$  and  $p > 1$  such that  $u_{i_1} \cdots u_{i_p} = v_{i_1} \cdots v_{i_p}$ .*

**Theorem 4.** [16] *The Post Correspondence Problem is undecidable.*

A pair of words  $\langle u, v \rangle \in \langle \Sigma^* \times \Sigma^* \rangle$  is a *witness* of a PCP instance  $\mathcal{P}$  if they can be decomposed into  $u = u_{i_1} \cdot u_{i_2} \cdots u_{i_p}$  and  $v = v_{i_1} \cdot v_{i_2} \cdots v_{i_p}$  such that  $u_i = U[i]$  and  $v_i = V[i]$ . If there is also  $u = v$ , we call such a pair a *valid witness*, which corresponds to a positive answer to the PCP problem. Our goal is to build an implementation  $\mathcal{I}$  that is *not* weakly consistent with respect to the **read/write memory** if and only if the instance  $\mathcal{P}$  has a valid witness. That is the implementation  $\mathcal{I}$  produces, for each pair of words  $\langle u, v \rangle$ , an execution  $\mathcal{H}_{uv}$  that is not weakly consistent if and only if  $\langle u, v \rangle$  forms a valid witness. The construction of each history  $\mathcal{H}_{uv}$  relies on ten processes and seven variables (six data variable and one synchronization variable).

**Theorem 5.** *Given an implementation  $\mathcal{I}$  as a regular language, checking whether all executions of  $\mathcal{I}$  are weakly consistent with respect to  $S_{r/w}$  is undecidable.*

## 7 Related Work

The testing problems of relaxed consistency models have been intensively investigated in the literature. Wei et al. [17] proved that complexity of testing PRAM consistency is NP-complete. Bouajjani et al. [18] studied the complexity of verifying causal consistency for one history. It was proved that the problem is NP-complete for all the three variations of CC (causal consistency, causal convergence

and causal memory). A recent work by Furbach et al. [19] showed that the testing problem for any criterion weaker than sequential consistency and stronger than slow consistency is NP-complete. This range covers many relaxed memory consistency models, including (a weaker variation of) CC, TSO, PSO and PRAM consistency, but it does not cover WC. It is easy to construct executions that conform to WC but violate slow consistency.

For the model checking problem, it was shown that verifying linearizability is EXPSPACE-complete when the number of processes is bounded and undecidable otherwise [20]. Alur et al. [15] proved that checking sequential consistency is in general undecidable. The same conclusion holds for systems with only four objects. Wang et al. [21] studied the model checking problem of quasi-linearizability and proved that it is undecidable. For consistency criteria weaker than sequential consistency, eventual consistency has been shown to be decidable [22], and causal consistency was proved to be undecidable [18] in general.

The method we used to formalize weak consistency in terms of distributed histories is inspired by the work of [23], which extends the definition of CC to all abstract data types. Their work uses transition systems to specify sequentially abstract data types, which is modeled as transducers, a model that is very similar to Mealy machines [24]. Their approach for CC, however, does not easily transfer here. For WC, we have to consider the different roles played by data and synchronization operations, while causal consistency does not distinguish memory access categories.

## 8 Conclusion

This paper explores the complexity of deciding whether an execution of a shared memory system is weakly consistent. We prove that the TWC problem is NP-complete, even for systems with only three processes or programs in which each process is permitted to have only two memory (data) accesses. We show the TWC-read problem is also NP-complete, which implies tagging each read with the identity of the responsible write does not reduce the complexity. However, for TWC-read, if we restrict the process number to two, then a polynomial algorithm exists.

A new approach is proposed to formalize weak consistency. This new formalization, unlike those from previous work [3,6], is given in terms of distributed histories abstracted from concrete executions, making possible a direct application of this formalization into automatic verification. We have also explored the model checking problem of weak consistency. Generally, deciding whether an implementation has correctly implemented the read/write memory is undecidable, even when the implementation and specification are both regular. These results on TWC and the model checking problem suggest that reasoning about weak consistency is intrinsically hard.

Although the read-mapping does not help with reducing the complexity, it would be interesting to investigate the TWC and model checking problem for implementations under certain constraints, such as data independence, a property ensuring the system behaviors are independent to particular data values

stored at particular memory locations. These investigations remain future work. Moreover, as release consistency [25] is a consistency model that refines weak consistency by dividing synchronization into two types, the results presented in this work may well be applicable (with minor adaptations) to release consistency.

**Acknowledgement.** We thank the anonymous reviewers for their helpful feedback. This research was supported by National Natural Science Foundation of China Grant No. 60833001.


## References

1. Herlihy, M.P., Wing, J.M.: Linearizability - a correctness condition for concurrent objects. *Acm Trans. Program. Lang. Syst.* **12**(3), 463–492 (1990)
2. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comput.* **28**(9), 690–691 (1979)
3. Dubois, M., Scheurich, C., Briggs, F.: Memory access buffering in multiprocessors. *ACM SIGARCH Comput. Architect. News* **14**, 434–442 (1986). IEEE Computer Society Press
4. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* **21**(7), 558–565 (1978)
5. Lipton, R.J., Sandberg, J.S.: PRAM: a scalable shared memory. Princeton University, Department of Computer Science (1988)
6. Adve, S.V., Hill, M.D.: Weak ordering - a new definition. *ACM SIGARCH Comput. Architect. News* **18**, 2–14 (1990). ACM
7. Manson, J., Pugh, W., Adve, S.V.: The Java memory model. In: *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005*, pp. 378–391. ACM, New York (2005)
8. Boehm, H.J., Adve, S.V.: Foundations of the C++ concurrency memory model. In: *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2008*, pp. 68–78. ACM, New York (2008)
9. Version, I.P.I.: 2.06 revision b. Book I: Power ISA User Instruction Set Architecture (2010)
10. Sarkar, S., Sewell, P., Alglave, J., Maranget, L., Williams, D.: Understanding power multiprocessors. *ACM SIGPLAN Not.* **46**(6), 175–186 (2011)
11. Alglave, J., Fox, A., Ishtiaq, S., Myreen, M.O., Sarkar, S., Sewell, P., Nardelli, F.Z.: The semantics of power and arm multiprocessor machine code. In: *Proceedings of the 4th Workshop on Declarative Aspects of Multicore Programming*, pp. 13–24. ACM (2009)
12. Gaoang Liu, X.L.: The complexity of weak consistency, December 2017. <https://github.com/GaoangLiu/github.io/blob/master/complexitywc/the-complexity-of-wc-full-description.pdf>
13. Papadimitriou, C.: *The Theory of Database Concurrency Control*. Computer Science Press Inc., Rockville (1986)
14. Gharachorloo, K.: *Memory consistency models for shared-memory multiprocessors*. Ph.D. thesis, Stanford University (1995)
15. Alur, R., McMillan, K., Peled, D.: Model-checking of correctness conditions for concurrent objects. In: *11th Annual IEEE Symposium on Logic in Computer Science, Proceedings*, pp. 219–228 (1996)

16. Post, E.L.: A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.* **52**(4), 264–268 (1946)
17. Wei, H., De Biasi, M., Huang, Y., Cao, J., Lu, J.: Verifying pram consistency over read/write traces of data replicas. arXiv preprint [arXiv:1302.5161](https://arxiv.org/abs/1302.5161) (2013)
18. Bouajjani, A., Enea, C., Guerraoui, R., Hamza, J.: On verifying causal consistency. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, pp. 626–638. ACM (2017)
19. Furbach, F., Meyer, R., Schneider, K., Senftleben, M.: Memory-model-aware testing: a unified complexity analysis. *ACM Trans. Embedded Comput. Syst. (TECS)* **14**(4), 63 (2015)
20. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Verifying concurrent programs against sequential specifications. *Program. Lang. Syst.* **7792**, 290–309 (2013)
21. Wang, C., Lv, Y., Liu, G., Wu, P.: Quasi-linearizability is undecidable. In: Feng, X., Park, S. (eds.) *APLAS 2015*. LNCS, vol. 9458, pp. 369–386. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-26529-2\\_20](https://doi.org/10.1007/978-3-319-26529-2_20)
22. Bouajjani, A., Enea, C., Hamza, J.: Verifying eventual consistency of optimistic replication systems. *ACM SIGPLAN Not.* **49**, 285–296 (2014). ACM
23. Perrin, M., Mostefaoui, A., Jard, C.: Causal consistency: beyond memory. In: *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, p. 26. ACM (2016)
24. Mealy, G.H.: A method for synthesizing sequential circuits. *Bell Syst. Tech. J.* **34**(5), 1045–1079 (1955)
25. Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., Hennessy, J.: Memory consistency and event ordering in scalable shared-memory multiprocessors. *SIGARCH Comput. Archit. News* **18**(2SI), 15–26 (1990)



# Balanced Random Constraint Satisfaction: Phase Transition and Hardness

Tian Liu<sup>1</sup>(✉) , Chaoyi Wang<sup>1</sup>, and Wei Xu<sup>2</sup>(✉)

<sup>1</sup> Key Laboratory of High Confidence Software Technologies, Ministry of Education, School of EECS, Peking University, Beijing 100871, China

lt@pku.edu.cn

<sup>2</sup> School of Mathematics and Physics, University of Science and Technology Beijing, Beijing 100083, China

xuwei@ustb.edu.cn

**Abstract.** Two models of nearly balanced random constraint satisfaction problems, called Model NB and UB respectively, are defined in this paper. By *nearly balanced* it means that most variables appear in the same number of constraints. Exact satisfiability thresholds for these models are proven, which are of the same values as that for Model RB. Experiments on random instances around the thresholds for these three models are conducted. The results show that these balanced models are much harder to solve than their unbalanced counterpart.

**Keywords:** Random instances · Phase transition · Hardness

## 1 Introduction

Since the discovery of the satisfiability phase transitions and many hard instances around the satisfiability thresholds of random constraint satisfaction problems, in e.g. [1, 6–16, 18, 20–22, 24, 25, 27, 28], benchmarks based on random constraint satisfaction have become important tools in algorithm evaluations [23]. The initial standard models A, B, C, D [17, 24] all suffered from asymptotic unsatisfiability [1, 18], thus were unable to generate many hard instances as benchmarks. Subsequent models were constructed to overcome this deficiency of standard models [1, 17, 21, 24], either by embedding structures [16, 17], or by controlling parameters [11, 12, 27, 28].

In this paper, partially inspired by previous results in [2–4], we define two nearly balanced CSP models with both embedded structures and controlled parameters to generate even harder instances as benchmarks. By *nearly balanced* it means that most variables appear in the same number of constraints. We embed perhaps the simplest structure into the constraint hypergraphs, that is, to make the hypergraphs nearly regular, in a simple way as in *configuration models* [5].



We control parameters of our models as in Model RB [27,28], the revised B Model, which probably is the most applied random CSP model in generating benchmarks, besides the classical random  $k$ -SAT model. Model RB is flexible in generating various kinds of instances, ranging from Boolean Satisfiability, Pseudo-Boolean, Vertex Cover (equivalently Independent Set and Clique), to Integer Programming, etc. [26] Model RB is also robust in hiding optimal solutions into the instances while keeping their hardness [27].

Our first model is called Model NB, which is a nearly balanced revised B model. Our second model is called Model UB, whose constraint hypergraphs are uniform, i.e. each constraint involves the same number of variables, while Model NB is not uniform. Exact phase transitions of Model NB and UB are shown, where the satisfiability thresholds are of the same values as that of Model RB. Experiments are conducted with complete solver, and also with local search solver on forced satisfiable instances, on Model NB and UB, as well as on Model RB. The results show that Model UB is only slightly harder than Model NB, but both Model NB and UB are much harder than Model RB, at least ten times harder in moderate input size.

This paper is structured as follows. After introducing CSPs, Model RB and Model RD, as well as notions on hypergraphs (Sect. 2), we define Model NB and Model UB (Sect. 3) and show exact thresholds of satisfiability for these models (Sect. 4). Experimental results are presented (Sect. 5) before Conclusion (Sect. 6).

## 2 Preliminaries

In *constraint satisfaction problem* (CSP), each instance is a triple  $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ , where  $\mathcal{V} = \{v_1, \dots, v_n\}$  is a set of  $n$  variables, *domain*  $\mathcal{D} = \{1, \dots, d\}$  is a set of  $d$  values,  $\mathcal{C} = \{C_1, \dots, C_m\}$  is a set of  $m$  constraints. Each constraint  $C_i$  is a pair  $(S_i, R_i)$ , where *constraint scope*  $S_i = (v_{i_1}, \dots, v_{i_k})$  with *arity*  $k$  is a  $k$ -tuple of different variables, and *constraint relation*  $R_i \subseteq \mathcal{D}^k$  with *constraint slackness*  $|R_i|/|\mathcal{D}^k|$ . An *assignment*  $a = (a_1, \dots, a_n)$  is a mapping from  $\mathcal{V}$  to  $\mathcal{D}$ , where  $a_i$  is the value of  $v_i$ . A constraint  $C_i = (S_i, R_i)$  is satisfied by assignment  $a$ , if the  $k_i$ -tuple of values assigned to variables in  $S_i$  is contained in  $R_i$ . A *solution* is an assignment that satisfies all constraints. A CSP instance is *satisfiable*, if it has a solution, otherwise *unsatisfiable*.

Model RB and RD were introduced by [28]. Let  $n, m, d, k, p$  be number of variables, number of constraints, domain size, constraint arity and constraint slackness, respectively. Model RB and Model RD are defined with growing domain size  $d = n^\alpha$  and superlinear number  $m = rn \ln n$  of constraints, where constants  $\alpha > 0$ ,  $r > 0$  (called *density parameter*), and  $k = 2, 3, \dots$ , as follows.

1. Select with repetition  $m = rn \ln n$  random constraint scopes, and for each scope select without repetition  $k$  variables into it;
2. Select uniformly at random without repetition  $p \cdot d^k$  compatible assignments for each constraint (for Model RB), or select each assignment for the  $k$  variables with probability  $p$  as compatible independently (for Model RD).

Model RB and RD have been shown with exact satisfiability phase transition [28], as well as with many hard instances both theoretically [19, 27, 29] and experimentally [27].

Let  $\mathcal{H} = (V, \mathcal{E})$  be a *hypergraph*, consisting of a set  $V$  of vertices, and a set  $\mathcal{E}$  of hyperedges, where each hyperedge is a subset of  $V$ . The *constraint hypergraph* of a CSP instance consists of variables as vertices and constraint scopes as hyperedges. The *degree* of a vertex  $v$  in a hypergraph is the number of hyperedges which contain  $v$ . If for all vertices, the degrees are at the same value, we call the hypergraph *regular*. If for all vertices, the degrees differ by at most one, we call the hypergraph *balanced*. If for most (i.e.  $n - o(n)$ ) vertices, the degrees differ by at most one, we call the hypergraph *nearly balanced*. If all hyperedges contain the same number of vertices, we call the hypergraph *uniform*.

It is not convenient to study the probability space of random balanced hypergraphs. Instead, we study the probability space of random balanced multihypergraphs. A *multihypergraph*  $\mathcal{H} = (V, \mathcal{E})$  consists of a set  $V$  of vertices and a *multiset*  $\mathcal{E}$  of hyperedges. We can similarly define multihypergraphs to be the *regular, balanced, nearly balanced and uniform* as above. In *double* multihypergraphs, besides multihyperedges, each hyperedge can also be a *multiple* subset of  $V$ . Hence the degree of a vertex in a (double) multihypergraph is the total number of its appearance in the hyperedges. For double multihypergraphs, the appearance of the same vertex is summed up.

### 3 Model NB and Model UB

For  $n$  vertices,  $m$  hyperedges, and  $k$  vertices in each hyperedge, let  $NB(n, m, k)$  denote the set of all such nearly balanced *double multihypergraphs*, and  $UB(n, m, k)$  the set of all such nearly balanced *uniform* multihypergraphs. The average degree of such hypergraph is  $km/n$ . Define  $\delta = \lfloor km/n \rfloor$ , then such a hypergraph has  $km - n\delta$  vertices of degree  $\delta + 1$ , and all others of degree  $\delta$ .

For random graphs, the so-called *configuration model* is used to generate graphs with a given distribution of degrees (see e.g. page 52 in [5]). We extend the configuration model to hypergraphs to generate (nearly) balanced multihypergraphs as follows. Let  $W = \bigcup_{j=1}^n W_j$  be a fixed set of  $km = \sum_{j=1}^n d_j$  labeled vertices, where  $W_j = \{v_{j,i} | i = 1, 2, \dots, d_j\}$  and  $d_j$  is degree of the  $j$ -th vertex. A *configuration*  $F$  is a partition of  $W$  into  $m$  subsets of labeled vertices, such that each subset has  $k$  labeled vertices. These subsets are called *hyperedges* of  $F$ . Let  $\Phi$  be the set of configurations. So

$$|\Phi| = \frac{\prod_{i=0}^{m-1} \binom{km-ik}{k}}{m!} = \frac{(km)!}{m!(k!)^m}.$$

Each  $F$  can be transformed into a multihypergraph  $\phi(F) = G = (V, \mathcal{E})$  with  $V = \{v_1, \dots, v_n\}$ , and hyperedges  $\mathcal{E} = \{\{v_i | W_i \cap e \neq \emptyset\} | e \text{ is a subset in } F\}$ . Such a  $G$  can also be a double multihypergraph, where  $v_i$  appears  $|W_i \cap e|$  times in the hyperedge transformed from  $e$ . So for a random  $G$  in  $NB(n, m, k)$ ,

$$\Pr(G) = \frac{1}{|\Phi|} = \frac{m!(k!)^m}{(km)!}. \tag{1}$$

As multihypergraphs,  $G$  may be not uniform, i.e. the number of vertices in a hyperedge maybe less than  $k$ . Such hyperedges are called *degenerated*. We use following algorithm to make  $G$  uniform when  $k$  is not very large.

---

**Algorithm 1.** Making multihypergraph uniform

---

```

1 uniform( $G$ )
   Input: A balanced multihypergraph  $G = (V, \mathcal{E})$ 
   Output: A balanced uniform hypergraph  $G' = (V, \mathcal{E}')$ 
2 begin
3   for each degenerated hyperedge  $e_1$  in  $\mathcal{E}$  do
4     randomly choose  $k - 1$  nondegenerate hyperedge  $e_2, \dots, e_k$ ;
5     while true do
6       if every vertex in multiset  $E = e_1 \cup e_2 \cup \dots \cup e_k$  occurs at most  $k$ 
7         times then
8           randomly put each duplicated vertices into at most  $k$  different
9           set  $e'_1, e'_2, \dots, e'_k$  and randomly put other vertices in
10           $e'_1, e'_2, \dots, e'_k$  such that every set has the same number of
           vertices; replace  $e_1, e_2, \dots, e_k$  by  $e'_1, e'_2, \dots, e'_k$  in  $\mathcal{E}$ ;
           break;
        else
           continue;

```

---

We give a simple explanation of the correctness of Algorithm 1 when  $k \leq \sqrt{n}$ . The most interesting case is when  $k$  is not very large. Since  $e_1$  is a degenerated hyperedge, there are at most  $k - 1$  different vertices in  $e_1$ . For these  $k - 1$  vertices, there are at most  $(k - 1)\delta$  duplicated vertices out of  $e_1$ . Thus when  $(k - 1)\delta \leq m - k$ , there are  $k - 1$  nondegenerate hyperedges which can pass the judgment in line 6. Notice that  $\delta = \lfloor km/n \rfloor$  and if  $k \leq \sqrt{n}$ ,  $k = o(m)$ , we get  $(k - 1)\delta \leq m - k$ . So if we try enough steps, the algorithm can end up.

We can sample  $G$  from  $UB(n, m, k)$  by first sampling  $G$  from  $NB(n, m, k)$  and then running Algorithm 1. To sample  $G$  from  $NB(n, m, k)$ , we use the configuration model, by taking a random permutation on the set  $W$  and cutting it into  $m$  intervals each of length  $k$ , which costs  $O(km)$  time.

Now we define Model NB and Model UB, as well as Model ND and Model UD, as follows. For constants  $\alpha > 0$ ,  $r > 0$  (called *density parameter*),  $0 < p < 1$  (called *constraint slackness*), and integer  $k = 2, 3, \dots$ , let  $n, m = rn \ln n, d = n^\alpha, k, p$  be number of variables, number of constraints, domain size, constraint arity and constraint slackness, respectively.

- Sample a hypergraph from  $NB(n, m, k)$  (for Model NB and ND) or  $UB(n, m, k)$  (for Model UB and UD); For model UB and UD, the hypergraph can be seen as a constraint graph directly; For model NB and ND, the selected hypergraph can not be the constraint graph, so we delete the duplicate vertices in every hyperedge and we get the constraint graph;

- For each hyperedges of cardinality  $k'$ , select uniformly at random without repetition  $p \cdot d^{k'}$  compatible assignments (for Model NB and UB), or select each assignment for the  $k'$  variables with probability  $p$  as compatible independently (for Model ND and UD).

Below we only consider Model NB and UB, since the properties of Model ND and Model UD are similar. Model RB and UB have the same arity  $k$  for all constraints. In Model NB, the arity of each constraint is at most  $k$ .

**Theorem 1.** *Assume that  $r > 0$  and  $k = 2, 3, \dots$  are constants and  $m = rn \ln n$ . Also assume that  $\delta = \frac{km}{n} = rk \ln n$  is an integer. Let  $\mathcal{H} = (V, \mathcal{E})$  be the random constraint hypergraph of Model NB and UB. For any constant  $\epsilon > 0$ , we have the following results:*

1. *Except  $O(\ln^{1+\epsilon} n)$  vertices, all the remaining vertices in  $\mathcal{H}$  have degree  $\delta$  with high probability.*
2. *Except  $O(\ln^{2+\epsilon} n)$  constraints, all the remaining constraints have arity  $k$  with high probability.*

*Proof.* For fixed vertex  $v$ , let  $Deg(v)$  denote the degree of  $v$ . If  $Deg(v) < \delta$ , we call  $v$  degenerated. If a constraint  $C$  has arity less than  $k$ , we call the constraint degenerated. Use the idea of random permutation, the probability of  $Deg(v) = \delta$  is  $\Pr(Deg(v) = \delta) = \frac{\binom{m}{\delta} \delta! k^\delta (km - \delta)!}{(km)!} = \frac{k^\delta m(m-1) \dots (m-\delta+1)}{km(km-1) \dots (km-\delta+1)} \geq (\frac{km-k\delta}{km})^\delta = f(\frac{n}{k})^{\frac{k^2 m}{n^2}}$ , where  $f(x) = (1 - 1/x)^x$ . When  $x$  is unbounded,  $f(x)$  approaches a constant  $1/e > 0$ ,  $1 - e^{-x}$  approaches  $x$ , the expected number of degenerated vertices  $E(N) = n(1 - f(n/k)^{\frac{k^2 m}{n^2}})$  approaches  $n(1 - e^{-\frac{k^2 m}{n^2}}) = O(\ln n)$ . According to Markov inequality, we have the first conclusion. If the number of degenerated vertices is  $n_d$ , the degenerated constraints is at most  $\delta n_d/2$ . Hence the number of degenerated constraints is at most  $O(\delta \ln^{1+\epsilon} n) = O(\ln^{2+\epsilon} n)$  with high probability. □

## 4 Phase Transitions of Model NB and UB

In this section, we give the proof of phase transitions of Model NB and UB. The proof framework is similar to the previous papers [12, 28]. But since the balanced model is different from the previous random model, the term and the interval of the summation formula is different in details, thus resulting in a more complex proof than [28].

For Model NB, define the following events:

- $Sat(a, i)$ : assignment  $a$  satisfies constraint  $C_i$ ;
- $Sat(a)$ :  $a$  is a solution, that is  $a$  satisfies all constraints;
- $Sat(a, b, i)$ : both  $a$  and  $b$  satisfy  $C_i$ ;
- $Sat(a, b)$ : both  $a$  and  $b$  are solutions;
- $Sat$ : the random instance is satisfiable.

The similarity number of pairs of assignments is defined as  $S(a, b) = n - \text{Ham}(a, b)$ , where  $\text{Ham}(a, b)$  is the Hamming distance between  $a$  and  $b$ .

**Theorem 2.** *Let constants  $\alpha > 0$ ,  $r > 0$ ,  $0 < p < 1$ , and  $k = 2, 3, \dots$ . Let  $d = n^\alpha$ ,  $m = rn \ln n$ ,  $r_{cr} = -\alpha / \ln p$ . For  $\alpha > 1/k$  and  $p > 1/k$ , we have*

$$\lim_{n \rightarrow \infty} \Pr(\text{Sat}) = \begin{cases} 0, & r > r_{cr} \\ 1, & r < r_{cr} \end{cases}$$

*Proof.* The number of solutions  $N$  is  $\sum_{a \in \mathcal{D}^n} I(a)$ , where

$$I(a) = \begin{cases} 1, & \text{if } a \text{ is a solution,} \\ 0, & \text{otherwise.} \end{cases}$$

The expectation of  $N$  is  $\mathbb{E}(N) = \sum_{a \in \mathcal{D}^n} \Pr(\text{Sat}(a)) = d^n \prod_{i=1}^m \Pr(\text{Sat}(a, i)) = d^n p^m$ . By the Markov Inequality,  $\Pr(\text{Sat}) = \Pr(N \geq 1) \leq \mathbb{E}(N)$ . When  $r > r_{cr}$ ,  $\lim_{n \rightarrow \infty} \Pr(\text{Sat}) \leq \lim_{n \rightarrow \infty} \mathbb{E}(N) = 0$ .

By the Cauchy-Schwartz Inequality,  $\Pr(\text{Sat}) = \Pr(N \neq 0) \geq \frac{\mathbb{E}^2(N)}{\mathbb{E}(N^2)}$ . The expectation of  $N^2$  is  $\mathbb{E}(N^2) = \sum_{a, b \in \mathcal{D}^n} \mathbb{E}(I(a)I(b)) = \sum_{a, b \in \mathcal{D}^n} \Pr(\text{Sat}(a, b))$ . Clearly,  $\Pr(\text{Sat}(a, b)) = \prod_{i=1}^m \Pr(\text{Sat}(a, b, i))$ .

For a constraint  $C_i$  of arity  $k_i$ , the number of incompatible tuples is  $q_i = (1 - p)d^{k_i}$ . Consider the following two cases:

1. Each variable of  $C_i$  is assigned the same value in  $a$  as that in  $b$ . In this case,  $\Pr(\text{Sat}(a, b, i)) = \binom{d^{k_i} - 1}{q_i} / \binom{d^{k_i}}{q_i} = p$ .
2. Otherwise,  $\Pr(\text{Sat}(a, b, i)) = \binom{d^{k_i} - 2}{q_i} / \binom{d^{k_i}}{q_i} \leq p^2$ .

Suppose the similarity number of  $(a, b)$  is  $S(a, b) = S$ , then the probability of the first case is  $\frac{\binom{S\delta}{k} k! (n\delta - k)!}{(n\delta)!} = \frac{\binom{S\delta}{k}}{\binom{n\delta}{k}}$ . Denoted this probability by  $\sigma_{S,i}$ , then the probability of the second case is  $1 - \sigma_{S,i}$ . For  $S \leq n$ ,  $\frac{S\delta - i}{n\delta - i} \leq \frac{S\delta}{n\delta} = \frac{S}{n}$ , so  $\sigma_{S,i} = \frac{\binom{S\delta}{k}}{\binom{n\delta}{k}} = \frac{S\delta(S\delta - 1) \dots (S\delta - k + 1)}{n\delta(n\delta - 1) \dots (n\delta - k + 1)} \leq \left(\frac{S}{n}\right)^k$ . Thus  $\Pr(\text{Sat}(a, b, i)) = \frac{\binom{d^{k_i} - 1}{q_i}}{\binom{d^{k_i}}{q_i}}$ .

$$\sigma_{S,i} + \frac{\binom{d^{k_i} - 2}{q_i}}{\binom{d^{k_i}}{q_i}} \cdot (1 - \sigma_{S,i}) \leq p \cdot \sigma_{S,i} + p^2 \cdot (1 - \sigma_{S,i}) \leq p \cdot \left(\frac{S}{n}\right)^k + p^2 \cdot \left(1 - \left(\frac{S}{n}\right)^k\right).$$

Let  $A_S$  be the set of assignment pairs whose similarity number is  $S$ . Then  $|A_S| = d^n \binom{n}{S} (d - 1)^{n - S}$  and  $\mathbb{E}(N^2) = \sum_{S=0}^n |A_S| \Pr(\text{Sat}(a_S, b_S))$ , where  $(a_S, b_S) \in A_S$ . Thus  $\mathbb{E}(N^2) \leq \sum_{S=0}^n d^n \binom{n}{S} (d - 1)^{n - S} \left( p \cdot \left(\frac{S}{n}\right)^k + p^2 \cdot \left(1 - \left(\frac{S}{n}\right)^k\right) \right)^m$ . For  $S = 0$ ,  $\left(\frac{S}{n}\right)^k = 0$ , and  $\sum_{S=0}^n \binom{n}{S} (d - 1)^{n - S} = d^n$ . Thus  $\mathbb{E}(N^2) \leq d^{2n} p^{2m} + \sum_{S=1}^n d^n \binom{n}{S} (d - 1)^{n - S} \left( \left( p \cdot \left(\frac{S}{n}\right)^k + p^2 \cdot \left(1 - \left(\frac{S}{n}\right)^k\right) \right)^m - p^{2m} \right)$ . Finally,  $\frac{1}{\Pr(\text{Sat})} \leq \frac{\mathbb{E}(N^2)}{\mathbb{E}^2(N)} \leq 1 + \sum_{S=1}^n \frac{\binom{n}{S} (d - 1)^{n - S} \left( \left( p + (1 - p) \left(\frac{S}{n}\right)^k \right)^m - p^m \right)}{d^n p^m}$ . By Theorem 3, which will be proven in the below,

$$\lim_{n \rightarrow \infty} \sum_{S=k}^n \frac{\binom{n}{S} (d - 1)^{n - S} \left( \left( p + (1 - p) \left(\frac{S}{n}\right)^k \right)^m - p^m \right)}{d^n p^m} = 0.$$

Thus when  $r < r_{cr}$ ,  $\lim_{n \rightarrow \infty} \Pr(Sat) = 1$ . □

**Theorem 3.** For constants  $\alpha > 0$ ,  $r > 0$ ,  $0 < p < 1$  and integer  $k \geq 2$ , let  $d = n^\alpha$ ,  $m = rn \ln n$  and  $r_{cr} = -\alpha / \ln p$ . When  $\alpha > 1/k$ ,  $p > 1/k$  and  $r < r_{cr}$ , we have  $\lim_{n \rightarrow \infty} \sum_{S=1}^n F_S = 0$ , where

$$F_S = \frac{\binom{n}{S} (d-1)^{n-S} \left( \left( p + (1-p) \left( \frac{S}{n} \right)^k \right)^m - p^m \right)}{d^n p^m}.$$

This theorem is proven by a series of propositions as follows. For convenience, in the following we let

$$x = \frac{S}{n}, \quad t = -\frac{m \ln p}{n \ln d} = -\frac{rn \ln n \ln p}{n \ln n^\alpha} = -\frac{r \ln p}{\alpha}.$$

Then  $\frac{1}{n} \leq x < 1$ ,  $t$  is a positive constant, and  $t < 1$  if and only if  $r < r_{cr}$ . We also let

$$y = x^k, \quad f(y) = \left( 1 + \frac{1-p}{p} y \right)^m.$$

Then  $F_S$  can be rewritten as

$$F_S = \binom{n}{S} \left( \frac{1}{d} \right)^S \left( 1 - \frac{1}{d} \right)^{n-S} (f(y) - 1). \tag{2}$$

**Proposition 1.** For constant  $\lambda \in (0, \min\{1 - \frac{1}{k}, \alpha - \frac{1}{k}\})$ ,  $\lim_{n \rightarrow \infty} \sum_{S=1}^{\frac{n}{n^{1/k+\lambda}}} F_S = 0$ .

*Proof.* Let  $h_n(x) = \ln \ln d + \ln n + \frac{t(1-p)}{-p \ln p} n x^k \ln d + k \ln x$ .

**Lemma 1.**  $f(y) - 1 < \frac{t(1-p)}{-p \ln p} \cdot e^{h_n(x)}$ .

*Proof.* Since  $f(0) = 1$ , by Mean Value Theorem, there exists  $\theta_y \in (0, x^k)$  such that  $f(y) - f(0) = \frac{1-p}{p} m \left( 1 + \frac{1-p}{p} \theta_y \right)^{m-1} y$ . Since  $\theta_y \in (0, x^k)$ , there exists  $\theta_x \in (0, x)$  such that  $\theta_x^k = \theta_y$ , we have  $\left( 1 + \frac{1-p}{p} x^k \right)^m - 1 = \frac{1-p}{p} m \left( 1 + \frac{1-p}{p} \theta_x^k \right)^{m-1} x^k \leq \frac{1-p}{p} m \left( 1 + \frac{1-p}{p} x^k \right)^{m-1} x^k$ . Since  $\ln(1+z) < z$  for all  $z > 0$ ,  $\left( 1 + \frac{1-p}{p} x^k \right)^m - 1 < \frac{1-p}{p} m \cdot e^{m \ln(1 + \frac{1-p}{p} x^k) + k \ln x} < \frac{t(1-p)}{-p \ln p} \cdot e^{h_n(x)}$ . □

**Lemma 2.**  $h_n(x) < -\frac{k\lambda}{2} \ln n$ , for  $\frac{1}{n} < x < \frac{1}{n^{1/k+\lambda}}$ .

*Proof.* Since  $h_n(x)$  is increasing in  $x$ ,  $h_n(x) < h_n\left(\frac{1}{n^{1/k+\lambda}}\right)$ . Since  $d = n^\alpha$ , by definition of  $h_n(x)$ ,  $h_n\left(\frac{1}{n^{1/k+\lambda}}\right)$  is a sum of four terms, where the first and third terms are  $o(\ln n)$  and the fourth term is  $(-1 - k\lambda + o(1)) \ln n$ . Thus  $h_n\left(\frac{1}{n^{1/k+\lambda}}\right) = (-k\lambda + o(1)) \ln n < -\frac{k\lambda}{2} \ln n$ . □

Thus  $f(y) - 1 = o(1)$  by Lemmas 1 and 2. By formula (2),  $\sum_{S=1}^{\frac{n}{n^{1/k+\lambda}}} F_S = \left( \sum_{S=1}^{\frac{n}{n^{1/k+\lambda}}} \binom{n}{S} \left(\frac{1}{d}\right)^S \left(1 - \frac{1}{d}\right)^{n-S} \right) o(1)$ . Since  $\sum_{S=0}^n \binom{n}{S} \left(\frac{1}{d}\right)^S \left(1 - \frac{1}{d}\right)^{n-S} = 1$ , we have  $\sum_{S=1}^{\frac{n}{n^{1/k+\lambda}}} F_S = o(1)$ . The proof of Proposition 1 is finished.  $\square$

Let  $H(x) = -x \ln x - (1-x) \ln(1-x)$  be the natural entropy function. Then  $0 \leq H(x) \leq \ln 2$ ,  $H(1-x) = H(x)$ .

**Proposition 2.** *For some constant  $\eta \in (0, 1)$ ,  $\lim_{n \rightarrow \infty} \sum_{S=\frac{n}{n^{1/k+\lambda}}}^{\eta n} F_S = 0$ .*

*Proof.* Since  $\binom{n}{S} \leq e^{nH(\frac{S}{n})}$ , we get  $\ln(F_S) < nH(x) - nx \ln d + n(1-x) \ln(1-d^{-1}) + \ln(f(y) - 1)$ . Since  $\ln(f(y) - 1) < \ln(f(y)) = \frac{tn \ln d}{-\ln p} \ln\left(1 + \frac{1-p}{p} x^k\right)$  and  $\ln(1+z) < z$  for all  $z > -1$ , we have  $\ln(F_S) < n\left(H(x) + x \ln d^{-1} + \frac{t(1-p)}{-p \ln p} x^k \ln d\right)$ . When  $x \in [\frac{1}{n^{1/k+\lambda}}, \eta]$ ,  $xn \ln d$  is a polynomial of  $n$ . Since  $\frac{1-p}{-p \ln p}$  is decreasing in  $p$  and  $p > 1/k$ , we can choose a sufficiently small constant  $\eta > 0$  such that  $\frac{t(1-p)}{-p \ln p} x^{k-1}$  is sufficiently small. So we need only to prove  $\lim_{n \rightarrow \infty} \frac{H(x)}{x \ln d} - 1 < 0$ , which implies that  $F_S$  is exponentially small with  $S$  in the corresponding interval. It is easy to find that  $\lim_{n \rightarrow \infty} \frac{H(x)}{x \ln d} \leq \frac{1+k\lambda}{\alpha k}$ . By  $\lambda < \alpha - \frac{1}{k}$  (Proposition 1), we have  $\frac{1+k\lambda}{\alpha k} < 1$ .  $\square$

**Lemma 3** (Lemma 3.2 in [12]). *Let constant  $0 < \eta < 1$ . If  $k > 1/p$ , then*

$$g(x) = \frac{\ln\left(1 + \frac{1-p}{p} x^k\right)}{x} \leq -\ln p, \quad \forall x \in [\eta, 1].$$

**Proposition 3.** *For all constants  $0 < \eta < \gamma < 1$ ,  $\lim_{n \rightarrow \infty} \sum_{S=\eta n}^{\gamma n} F_S = 0$ .*

*Proof.* As in Proposition 2,  $\frac{\ln(F_S)}{n} < H(x) - x \ln d + (1-x) \ln(1-d^{-1}) + \frac{t \ln d}{-\ln p} \ln\left(1 + \frac{1-p}{p} x^k\right)$ , where  $H(x) + (1-x) \ln(1-d^{-1})$  is bounded when  $x \in [\eta, \gamma]$ . Note that  $t-1$  is negative and  $\ln d \rightarrow \infty$ , we have  $x \ln d^{-1} + \frac{t \ln d}{-\ln p} \ln\left(1 + \frac{1-p}{p} x^k\right) = x \ln d \left(-1 + \frac{t}{-\ln p} \cdot \frac{\ln\left(1 + \frac{1-p}{p} x^k\right)}{x}\right) \leq x \ln d \left(-1 + \frac{t}{-\ln p} \cdot (-\ln p)\right) = (t-1)x \ln d \leq (t-1)\eta \ln d$ . Since  $\lim_{n \rightarrow \infty} (t-1)\eta \ln d = -\infty$ ,  $F_S$  is exponentially small for  $x \in [\mu, \rho]$ , we have  $\lim_{n \rightarrow \infty} \sum_{S=\eta n}^{\gamma n} F_S = 0$ .  $\square$

**Proposition 4.** *For some constant  $\gamma \in (0, 1)$ ,  $\lim_{n \rightarrow \infty} \sum_{S=\gamma n}^n F_S = 0$ .*

*Proof.* This Proposition is similar to Proposition 3.2 in [12]. Due to its role in our proofs, we sketch a proof as follows. For  $(p + (1-p)x^k)^m - p^m \leq 1 - p^m < 1$ ,  $p^m d^n = d^{(1-t)n}$ , let  $\tau = 1 - t$ . Note that  $\sum_{S=\gamma n}^n F_S \leq nF_S$  and

$$nF_S \leq \frac{n \binom{n}{nx} (d-1)^{n(1-x)}}{d^n p^m} < \frac{ne^{nH(1-x)} (d-1)^{n(1-x)}}{d^{\tau n}}.$$

Since  $\lim_{x \rightarrow 1} H(1-x) = 0$  and  $(d-1)^{n(1-x)} < d^{n(1-x)}$ , there is a constant  $0 < \gamma < 1$ , such that when  $x \in [\gamma, 1]$ , we have  $H(1-x)/\ln d + (1-x) < \tau/2$ . Hence  $\forall x \in [\rho, 1]$ ,

$$\lim_{n \rightarrow \infty} nF_S \leq \lim_{n \rightarrow \infty} \frac{n}{d^{\tau n/2}} \cdot \frac{d^{(H(1-x)/\ln d + (1-x))n}}{d^{\tau n/2}} = 0.$$

□

For Model UB, similar results to Theorem 2 can also be obtained. Here we only sketch a proof of the phase transition of Model UB. First in the Algorithm 1, the ‘If’ statement in line 6 is true with high probability. So the algorithm runs in linear time with high probability. By symmetry, for the  $i$ -th constraint, the probability of a fixed scope is  $1/\binom{n}{k}$ , though the joint distribution of all scopes is not very clear. For both Model NB and UB, the compatible assignments are chosen independently for different constraints, thus for same parameters, the first moment  $\mathbb{E}(N)$  is the same for both model. But the second moment  $\mathbb{E}(N^2)$  is different. In fact, the only difference is that in Model UB,  $\sigma'_{S,i} = \binom{S}{k}/\binom{n}{k}$ . Since  $\sigma'_{S,i} \leq \sigma_{S,i}$  and  $\mathbb{E}(N^2)$  is increasing in  $\sigma_{S,i}$ , the second moment of Model UB is not larger than that of Model NB. Therefore, using the second moment method, we can conclude the exact phase transition of Model UB.

## 5 Experimental Hardness of Model NB/UB

The experiments are conducted on a server with Intel(R) Xeon(R) E7520 1.87GHz (16 cores) and 15.7G RAM under Linux. A complete solver Abscon and a local search solver CSP4j with TABU engine (see <http://www.cril.univ-artois.fr/CPAI08/>).

The experimental parameters are  $d = n^{0.6}$ ,  $k = 2$ ,  $p = 0.7$ ,  $n \in \{40, 50, 60\}$ . When  $t$  goes from 0.1 to 1.9 with step length 0.1,  $m$  is determined by  $-tn \ln d / \ln p$  accordingly. On each parameter point, 50 random instances are generated and averaged on. There are five experiments. The results are shown in the following four figures and three tables.

Table 1 shows the variance of running time, Table 2 shows the minimum and maximum time used by Abscon for  $n = 60$  of unforced instances.

**Table 1.** The average time (s) and the standard deviation of 50 instances around the threshold (Abscon)

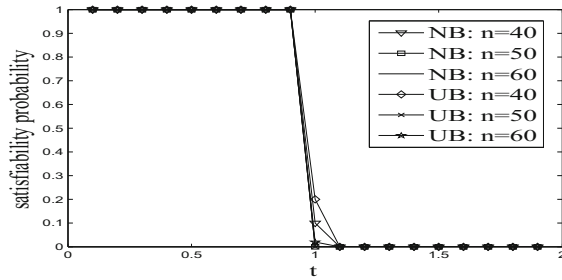
Model	n = 50		n = 60		n = 60, forced sat	
	#avg	StDev	#avg	StDev	#avg	StDev
RB	1.51	0.28	4.18	2.09	2.94	1.54
NB	3.47	0.89	71.5	24.7	35.67	30.75
UB	4.65	1.07	110.7	36.99	53.8	44.5



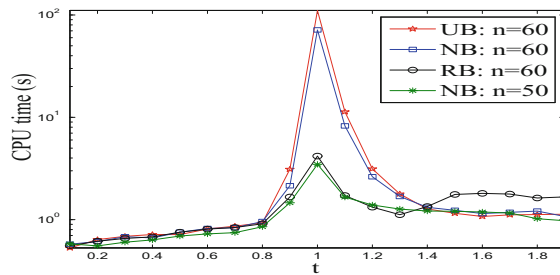
**Table 2.** The minimum and maximum time (s) around the threshold for  $n = 60$  (Abscon)

Model	RB		NB		UB	
	Min	Max	Min	Max	Min	Max
Time	1.61	11.12	29.36	119.17	32.86	197.42

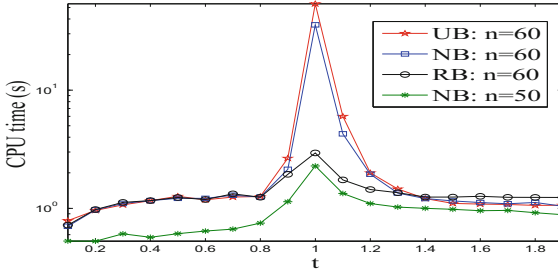
In Fig. 1, Abscon is used to show phase transition of Model NB and UB. There are clear satisfiability phase transitions around the threshold  $t = 1$ . In Fig. 2, Abscon is used to compare hardness between Model RB, NB and UB. Model NB is much harder to solve than Model RB, when  $n = 60$ . The hardest instances locate clearly around the threshold  $t = 1$ . In Fig. 3, Abscon is used to compare hardness of forced satisfiable instances between Model RB, NB and UB. In Fig. 4, CSP4j with TABU engine is used on forced satisfiable instances.



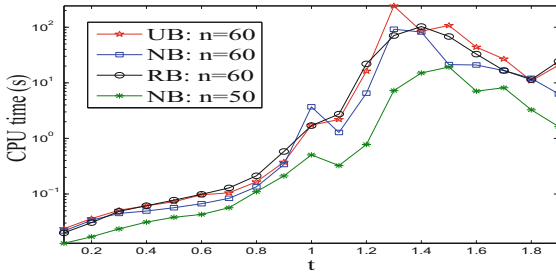
**Fig. 1.** Satisfiability phase transitions around  $t = 1$  (Abscon)



**Fig. 2.** CPU time in a logarithmic scale (Abscon)



**Fig. 3.** CPU time in a logarithmic scale of forced sat instances (Abscon)



**Fig. 4.** CPU time in a logarithmic scale of forced sat instances (CSP4j Tabu)

## 6 Conclusion

In this paper, two model of nearly balanced random constraint satisfaction problems called Model NB and UB are defined. Exact satisfiability thresholds are proven for these models. Experiments on random instances around the thresholds have been conducted. The results show that these balanced models are much harder to solve than their unbalanced counterpart Model RB. Thus, the nearly balanced models are more efficient in generating hard instances than the unbalanced models. Moreover, Model UB is only slightly harder than Model NB, thus being strictly balanced or uniform only helps a little. For practical purposes, being nearly balanced is already enough. We hope that these results are of use for future constructions of more challenging benchmarks.

**Acknowledgments.** We thank Prof. Ke Xu for suggestions on doing this work and for many helpful discussions. This work was partially supported by Natural Science Foundation of China (Grant Nos. 61370052 and 61370156) and Fundamental Research Funds for the Central Universities (Grant No. FRF-TP-16-065A1).

## References

1. Achlioptas, D., Molloy, M., Kirousis, L., Stamatiou, Y., Kranakis, E., Krizanc, D.: Random constraint satisfaction: a more accurate picture. *Constraints* **6**(4), 329–344 (2001)
2. Ansótegui, C., Béjar, R., Fernández, C., Gomes, C.P., Mateu, C.: The impact of balancing on problem hardness in a highly structured domain. In: *Proceedings of AAAI*, pp. 10–15 (2016)
3. Ansótegui, C., Béjar, R., Fernández, C., Gomes, C.P., Mateu, C.: Generating highly balanced sudoku problems as hard problems. *J. Heuristics* **17**(5), 589–614 (2011)
4. Ansótegui, C., Béjar, R., Fernández, C., Mateu, C.: On balanced CSPs with high treewidth. In: *Proceedings of AAAI*, vol. 1, pp. 161–166 (2007)
5. Bollobás, B.: *Random Graphs*. Cambridge Studies in Advanced Mathematics (2001)
6. Cheeseman, P., Kanefsky, R., Taylor, W.: Where the really hard problems are. In: *Proceedings of IJCAI*, pp. 163–169 (1991)
7. Cook, S.A., Mitchell, D.G.: Finding hard instances of the satisfiability problem: a survey. In: *Proceedings of SAT. DIMACS Series*, vol. 35, pp. 1–17 (1997)
8. Creignou, N., Daudé, H.: Random generalized satisfiability problems. In: *Proceedings of SAT*, pp. 17–26 (2002)
9. Creignou, N., Daudé, H.: Generalized satisfiability problems: minimal elements and phase transitions. *Theor. Comput. Sci.* **302**(1–3), 417–430 (2003)
10. Creignou, N., Daudé, H.: Combinatorial sharpness criterion and phase transition classification for random CSPs. *Inf. Comput.* **190**(2), 220–238 (2004)
11. Fan, Y., Shen, J.: On the phase transitions of random  $k$ -constraint satisfaction problems. *Artif. Intell.* **175**(3–4), 914–927 (2011)
12. Fan, Y., Shen, J., Xu, K.: A general model and thresholds for random constraint satisfaction problems. *Artif. Intell.* **193**, 1–17 (2012)
13. Friedgut, E.: Sharp thresholds of graph properties, and the  $k$ -sat problem. *J. Am. Math. Soc.* **12**, 1017–1054 (1998)
14. Friedgut, E.: Hunting for sharp thresholds. *Random Struct. Algorithms* **26**(1–2), 37–51 (2005)
15. Frieze, A.M., Molloy, M.: The satisfiability threshold for randomly generated binary constraint satisfaction problems. *Random Struct. Algorithms* **28**(3), 323–339 (2006)
16. Gao, Y., Culberson, J.: Consistency and random constraint satisfaction problems. *J. Artif. Intell. Res.* **28**, 517–557 (2007)
17. Gent, I.P., Macintyre, E., Prosser, P., Smith, B.M.: Random constraint satisfaction: flaws and structure. *Constraints* **6**, 345–372 (2001)
18. Gomes, C., Walsh, T.: Randomness and structures. *Handbook of Constraint Programming*, pp. 639–664 (2006)
19. Liu, T., Lin, X., Wang, C., Su, K., Xu, K.: Large hinge width on sparse random hypergraphs. In: *Proceedings of IJCAI*, pp. 611–616 (2011)
20. Mitchell, D.G., Selman, B., Levesque, H.J.: Hard and easy distributions of SAT problems. In: *Proceedings of AAAI*, pp. 459–465 (1992)
21. Molloy, M.: Models for random constraint satisfaction problems. *SIAM J. Comput.* **32**(4), 935–949 (2003)
22. Prosser, P.: An empirical study of phase transitions in binary constraint satisfaction problems. *Artif. Intell.* **81**(1–2), 81–109 (1996)

23. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. *Artif. Intell.* **81**(1–2), 17–29 (1996)
24. Smith, B.M., Dyer, M.E.: Locating the phase transition in binary constraint satisfaction problems. *Artif. Intell.* **81**(1–2), 155–181 (1996)
25. Smith, B.: Constructing an asymptotic phase transition in random binary constraint satisfaction problems. *Theor. Comput. Sci.* **265**, 265–283 (2001)
26. Xu, K.: BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems. <http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm>
27. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random constraint satisfaction: easy generation of hard (satisfiable) instances. *Artif. Intell.* **171**, 514–534 (2007)
28. Xu, K., Li, W.: Exact phase transitions in random constraint satisfaction problems. *J. Artif. Intell. Res.* **12**, 93–103 (2000)
29. Xu, K., Li, W.: Many hard examples in exact phase transitions. *Theor. Comput. Sci.* **355**(3), 291–302 (2006)



# Exact Algorithms for Allocation Problems

Sundar Annamalai  and N. S. Narayanaswamy 

Department of Computer Science and Engineering,  
Indian Institute of Technology, Madras, India  
mr.sundu@gmail.com, swamy@cse.iitm.ac.in

**Abstract.** We design new exact exponential time algorithms for the well known NP-hard allocation problems- Makespan minimization, the Santa Claus problem (with and without capacity constraints) and the Bin packing problem. These problems are very well-studied in the paradigm of approximation algorithms. However the best known exact, exponential-time algorithms for all of the above problems has complexity of  $O^*(3^m)$  [6], where  $m$  is the number of jobs except for Bin Packing which has a  $O^*(2^m)$  inclusion exclusion based algorithm (where  $m$  is the number of items) [8]. We introduce a new dynamic programming formulation which helps solve Makespan minimization and Santa Claus problem more efficiently in  $O^*(2^m)$  time and gives a completely different approach with the same time complexity in case of Bin Packing. In addition, Jansen et al. [6] showed that unless the *ETH* (exponential time hypothesis) is false, there is no exact algorithm that runs in time  $2^{o(m)}$ .

## 1 Introduction

We consider allocation problems where the input consists of a set of jobs, a set of machines, and a valuation of machine-job pairs. The valuations we consider are unrestricted and thus we consider the most general form of the problems in question. The Santa Claus Problem is to assign an indivisible set  $J$  of  $m$  jobs to a set  $M$  of  $n$  machines (labeled from 1 to  $n$ ) such that minimum valuation of any machine for the set of jobs assigned to it is maximised (each machine  $i$  has valuation of  $v_i(S)$  for the job set  $S$ ). In other words we need to find an allocation  $T : M \rightarrow 2^J$  such that  $\min_i(v_i(T(i)))$  is maximised. When each machine can be allocated at most  $k$  jobs, where  $k$  is given as part of the input, we get the Santa Claus Problem with capacity constraints. Makespan minimization is the min-max counterpart of Santa Claus problem. It is the problem of assigning an indivisible set  $J$  of  $m$  jobs to a set  $M$  of  $n$  machines (labeled from 1 to  $n$ ) such that maximum valuation of any machine for the set of jobs it gets is minimized. In other words we need to find an allocation  $T$  such that  $\max_i(v_i(T(i)))$  is minimised. Bin packing is the problem of allocating  $m$  objects(given)of different volumes into some number of bins, say  $k$ , such that each bin holds objects of total volume at most  $V$  (given). We need to find the minimum possible value of  $k$  for which such a packing is possible. The definition makes the valuation function in the Bin packing problem inherently additive as volume of a set of

objects is sum of volumes of individual objects. Additive valuation functions are very natural and play an important role in our results. Additive valuation functions are those functions  $f : M * 2^J \rightarrow \mathbb{R}$  ( $2^J$  is power set of  $J$ ) such that  $f(m, S) = \sum_{i \in S} f(m, i)$ : the valuation of a subset  $S$  for a machine  $m$  is equal to the sum of valuations of machine  $m$  for each individual machine in the set  $S$ .

**Related Work.** All the allocation problems we have studied in this paper have been well-studied in the world of approximation algorithms, and relatively recently The Santa Claus problem has been extensively studied [1–4]. While the world of parameterized and exact exponential time algorithms is a very rich area, we find surprisingly few theoretical results related to the area of scheduling and resource allocation, while these areas have many heuristic approaches that are applied extensively in practice. A very basic scheduling problem is the problem of ordering jobs(weighted) into a single machine such that the total penalty incurred by the machine is minimized. Here the total penalty is the sum of penalties of each job. Fomin and Kratsch [5] and Woeginger [8] designed an  $O^*(2^m)$  algorithm for this problem. Lente et al. [7] in their survey had described an  $O^*(3^m)$  time algorithm for a generalization of this problem by considering the variant with multiple machines. Jansen et al. [6] gave an  $O^*(3^m)$  algorithm for makespan minimization, Bin packing and a set of other problems. They also showed that unless the ETH(exponential time hypothesis) is false, there is no algorithm with running time  $2^{o(m)}$ . Their algorithm is very natural, and is the starting point for our work. We describe the algorithm in Sect. 2.

**Our Results.** Our exact algorithms use techniques which are completely different from those used in the approximation algorithms. We observe that the algorithm of Jansen et al. [6] works for the more generic case of non additive valuations which satisfy the monotonicity property. A function  $f$  defined on the power set of a universe satisfies the monotonicity property if  $S' \subseteq S$  implies that  $f(S') \leq f(S)$ . We then present  $O^*(2^m)$  algorithms for the **unrestricted** Santa Claus problem, the Santa Claus problem with capacity constraints, makespan minimization and Bin packing-all under the assumption of additive valuation functions. Our contribution is in identifying for these problems a new optimal substructure that gives a more efficient bottom-up evaluation. This optimal substructure circumvents the seemingly inherent running time of  $3^m$  by algorithms whose basic idea is *nested enumeration* when the jobs are allocated to machines which are arranged in a linear order.

## 2 Santa Claus Problem with Non Additive Valuations

We present the  $O(n * 3^m)$  dynamic programming approach proposed by Jansen et al. [6] for solving the Santa Claus Problem. We do not present the proofs here due to space limitations. Here  $n$  is the number of machines and they are labelled from 1 to  $n$ .  $J$  is the set of  $m$  jobs which are labeled from 1 to  $m$ . Define  $v_i(S), i \in [1, n], S \subseteq J$  to be the valuation of  $i^{th}$  machine for the set  $S$  of jobs. For an allocation  $T$  of jobs to the machines,  $T(i)$  denotes the set of jobs assigned to machine  $i$ .

**Optimal Substructure:** The algorithm assigns jobs to machines in the increasing order of the machine labels. The subproblems are denoted by the pair  $(i, S)$  where  $S \subseteq J$  and  $1 \leq i \leq n$ . For a subproblem  $(i, S)$ ,  $F_{i,S}$  denotes the optimal value to the problem of assigning a set  $S$  of jobs to the machines  $[1 \dots i]$ .

- $F_{i,S}$  denotes the highest value  $x$  such that we can assign the set of jobs  $S$  to the machine  $[1 \dots i]$  satisfying the constraint that each machine gets a valuation of at least  $x$ .
- Let  $G_{i,S}(j)$  for  $j < i$ , denote the set of jobs assigned to machine  $j$  in an optimal allocation to the subproblem  $(i, S)$ .
- Let the optimal allocation for the subproblem  $(i, S)$  be  $T^*$  and let  $S' \subseteq S$  denote the set of jobs assigned to machine  $i$  by  $T^*$ .

We now show that we can safely assume that the allocation to the first  $i - 1$  machines is the same as  $F_{i-1, S \setminus S'}$ .

**Lemma 1.** *There exists an optimal  $T^*$  such that  $T^*(i) = S'$  ( $T^*$  assigns  $S'$  to machine  $i$ ) and  $T^*(j) = G_{i-1, S \setminus S'}(j), \forall j < i$  (the allocation of  $T^*$  to machines  $[1 \dots i - 1]$  is the same as some optimal allocation to  $F_{i-1, S \setminus S'}$ ).*

**Dynamic Programming Formulation.** As a consequence of Lemma 1, it follows that if we know  $S' \subseteq S$  that is allocated to  $i$  in an optimum allocation, then the recursive subproblem to solve is to compute  $F_{i-1, S \setminus S'}$ . The value of the allocation thus obtained is  $\min(F_{i-1, S \setminus S'}, v_i(S'))$ , and we iterate this over all  $S' \subseteq S$ . We assume that the values  $v_i(S)$  for all  $i \in [1 \dots n]$  and  $S \subseteq J$  are in a  $n * 2^m$  2D- array with  $V[i][S]$  containing the value  $v_i(S)$ . We then initialise the base cases  $F[1][S] = v_1(S)$  as it is best to assign all the jobs to the only machine. We will then compute  $F_{i,S}$  for each set  $S \subseteq J$  and  $1 \leq i \leq n$  in a bottom-up fashion using the recursive formulation  $F_{i,S} = \min_{S' \subseteq S} (F_{i-1, S \setminus S'}, v_i(S'))$ .

**Running Time Analysis.** The number of such pairs of subsets  $S, S'$  is clearly  $3^m$  (each element in  $[1 \dots m]$  can either be present in none or only  $S$  or both  $S$  and  $S'$ ). So the whole table can be computed in  $O(n * 3^m)$  time.

### 3 A $O^*(2^m)$ Algorithm for the Santa Claus Problem

**Outline.** We first do a binary search on the maximum possible value that can be achieved for each machine. Once we fix this value  $T$ , the algorithm checks if there is an allocation of jobs to machines such that each machine gets a valuation which is greater than or equal to  $T$ . As in [6], we consider the machines in the order from 1 to  $n$  and assume that the jobs are being allocated in that order. Now for a subset  $S$  of jobs, define  $F(S)$  to be a pair  $\{p, q\}$  if, using the subset  $S$  of jobs, there is an allocation of set  $S$  of jobs to machines 1 to  $p$  which allocates a value of at least  $T$  and machine  $p + 1$  gets a value of  $q$  and the pair  $\{p, q\}$  is the maximum possible for the subset  $S$  of jobs. Here the maximum is defined according to the comparison operator,  $(a_1, a_2) > (b_1, b_2) \iff a_1 > b_1$  or  $(a_1 = b_1 \wedge a_2 > b_2)$ . To

compute  $F(S)$  we look at all subsets  $\{S' | S' \subseteq S, |S'| = |S| - 1\}$ . Let us take one such  $S'$  and look at the value  $F(S')$ . Let  $\{p', q'\}$  be the value of  $F(S')$ . Now we add the element which is in  $S$  and not in  $S'$  to the machine  $p' + 1$  and update  $q'$  accordingly. This is one possible answer for  $F(S)$  and we show that doing this for all the possible subsets ensures that we obtain the correct value for  $F(S)$ .

### 3.1 Dynamic Programming Formulation

Let  $w(p, q)$  denote the value of job  $q$  to machine  $p$ .

1. Do a binary search on the value of the optimum allocation (over integer space from minimum possible  $T$  to maximum possible  $T$ ). Let  $T$  denote the candidate value.
2. Check if there is an allocation that achieves the value  $T$  as follows.
  - (a) Consider all subsets  $S$  of the set  $J$  of jobs in the increasing order of their cardinality.
  - (b) For a subset  $S$ , compute the value  $F(S)$  as follows
    - Initialise  $F(S)$  to  $\{0, 0\}$ .
    - Consider all subsets  $S'$  of  $S$  such that  $S' \cup \{x'\} = S$  and  $x' \in S$  and let  $x'$  be the corresponding missing element from  $S'$ .
    - Let  $F(S')$  be  $\{p', q'\}$ .
    - Allocate  $x'$  to machine  $p' + 1$  and update  $q' \leftarrow q' + w(p' + 1, x')$
    - If  $q' \geq T$ , then update  $F(S) \leftarrow \max(F(S), \{p' + 1, 0\})$  (we have satisfied  $p' + 1$  too)
    - If  $q' < T$ , then  $F(S) \leftarrow \max(F(S), \{p', q'\})$ .
  - (c) If  $F(J) = \{n, 0\}$  then  $T$  is possible, otherwise  $T$  is not possible.

The following pseudo-code Santa-Claus( $n, m, V$ ) implements the algorithm and is useful to calculate the running time of the algorithm.

- 1: **function** Santa-Claus( $n, m, V$ )
  - ▷  $n$  is the number of machines,  $m$  is the number of jobs in  $J$
  - ▷  $V$  is the matrix with valuations.  $V[i][j]$  is the valuation of  $i^{th}$  machine to  $j^{th}$  job.
- 2:  $T_{min} \leftarrow 0, T_{max} \leftarrow$  sum of all values in  $V$ .
- 3: Do a binary search for  $T$  in range  $[T_{min}, T_{max}]$  using Check-Validity-Santa-Claus to find the largest value  $T_{opt}$  for which the function returns TRUE.
- 4: **return**  $T_{opt}$
- 5: **end function**
- 6:
- 7: **function** Check-Validity-Santa-Claus( $n, m, V, T$ )
  - ▷ Returns true if each machine can get value at least  $T$
- 8: Let  $F$  be an array, indexed by subsets of  $[1, m]$ , of  $2^m$  pairs with initialised to  $\{0, 0\}$ .
- 9:  $F[\emptyset] = \{0, 0\}$ .
- 10: Each array location in  $F$  corresponds to  $F(S)$  for some  $S \subseteq J$ .



```

11:   for all nonempty  $S \subseteq J = [1 \cdots m]$  in increasing order of  $|S|$  do
12:     for all  $x' \in S$  do
13:        $S' \leftarrow S \setminus \{x'\}$ 
14:        $\{p', q'\} \leftarrow F[S']$ 
15:       if  $q' + V[p' + 1][x'] < T$  then
16:          $F[S] \leftarrow \max(F[S], \{p', q' + V[p' + 1][x']\})$ 
17:       else
18:          $F[S] \leftarrow \max(F[S], \{p' + 1, 0\})$ 
19:       end if
20:     end for
21:   end for
22:   if  $F[J] = \{n, 0\}$  then return TRUE
23:   elsereturn FALSE
24:   end if
25: end function

```

We now prove the correctness of the above algorithm. Due to the additive nature of the valuation function, it is clear that the binary search gives the correct answer. Assume throughout the rest of discussion in this section that we have a fixed value  $T$  and are checking for its feasibility. Let  $X_S$  denote an allocation of the subset  $S$  of jobs to machines. Let  $p + 1$  be the first machine that does not get a value  $T$  in  $X_S$  and let  $p + 1$  get a load of value  $q$  in  $X_S$ . Let  $V(X_S)$  denote  $\{p, q\}$ .

**Lemma 2.** *For a subset  $S$  of  $J$ ,  $F[S]$  computed in the function Check-Validity-Santa-Claus is the maximum for  $T$ . In other words, for each allocation  $X_S$ ,  $F[S] \geq V(X_S)$ .*

*Proof.* We will show that  $\forall X_S V(X_S) \leq F[S]$ . We will show this by induction on the cardinality of the set  $S$ . The base case is when  $S = \emptyset$ . In this case  $F[\emptyset] = \{0, 0\}$  and no machine can get a non-empty set of jobs and machine 1 gets load 0 with the empty set of jobs. The induction hypothesis is that for all  $S$  such that  $|S| < k, k \geq 0$  and any allocation  $X_S$  of jobs from  $S$  to the machines,  $V(X_S) \leq F[S]$ . Given the induction hypothesis, we show that for all  $S$  such that  $|S| \leq k, k \geq 0$  and any allocation  $X_S$  of jobs from  $S$  to the machines  $V(X_S) \leq F[S]$ . Consider a subset  $S$  of jobs with cardinality  $k$ . Consider any allocation  $X_S$  of jobs in  $S$  to the machines. Let  $V(X_S) = \{p, q\}$ . Now there are 2 cases.

Case 1:  $q \neq 0$ .  $X_S$  has at least one job assigned to  $p + 1$ , and let  $x'$  be any one such job. Consider the set  $S' = S \setminus \{x'\}$ , and let  $X_{S'}$  be the allocation induced by  $X_S$  on the subset  $S'$  (removing the allocation of  $x'$  to  $p + 1$  alone). Now, we know by the induction hypothesis that  $F(S') \geq V(X_{S'}) = \{p, q - w(p + 1, x')\}$ , since  $|S'| = k - 1$ . In the function Check-Validity-Santa-Claus the value of  $F(S)$  obtained on adding  $x'$  to  $F(S')$  ensures that  $F(S)$  is at least  $\{p, q\} = V(X_S)$ .

Case2:  $q = 0$ .  $X_S$  has at least one job assigned to  $p$ . Let one such job be  $x'$ . By the definition of  $V(X_S)$ ,  $p$  has a load of at least  $T$ . Therefore, in the allocation

$X_S$  when the job  $x'$  is removed then  $p$  gets a load of at least  $T - w(p, x')$ . Now consider the set  $S' = S \setminus \{x'\}$ , and let  $X_{S'}$  be the allocation induced by  $X_S$  on the subset  $S'$  (removing the allocation of  $x'$  to  $p$ ). By the induction hypothesis we know that  $F(S') \geq V(X_{S'})$ , since  $|S'| = k - 1$ . Now  $V(X_{S'})$  is at least  $\{p - 1, T - w(p, x')\}$  and hence by I.H  $F(S') \geq \{p - 1, T - w(p, x')\}$  and due to the update rule  $F(S) = \max(F(S), g(F(S'), x'))$  we get  $F(S) \geq \{p, 0\} = V(X_S)$ . Therefore, the claim is true for a set  $S$  consisting of  $k$  jobs, given that it is true for any set with less than  $k$  jobs. This completes the proof by induction. Hence the lemma.

**Theorem 1.** *The function  $\text{Santa-Claus}(n, m, V)$  computes an optimum allocation of jobs to machines in time  $O(\log(T_{max})2^m)$ .*

*Proof.* From Lemma 2 it follows that for each  $S \subseteq J$ ,  $F[S]$  computed by the function  $\text{Check-Validity-Santa-Claus}(n, m, V, T)$  is the maximum for each value of  $T$ . Further, the function returns true if and only if  $F[S] = \{n, 0\}$  if it finds an allocation in which each machine gets a value of at least  $T$ . Since the algorithm does a binary search over the range of all possible values for  $T$ , it follows that it computes the value of an optimum allocation, and the execution of the algorithm also yields an optimum valued allocation.

**Running Time Analysis.** The binary search adds a factor which is  $\log(T_{max})$  which is polynomial in the input size. Inside each iteration of binary search, there is a function call to the  $\text{Check-Validity-Santa-Claus}$  function. The slowest steps in the function are in the for-loop in line 11 which contribute to  $2^m$  iterations each with an  $O(n)$  operation. So the overall complexity is  $O(\log(T_{max}) * n * 2^m)$  which is  $O(2^m * \text{poly}(\text{input size}))$ .

## 4 $O^*(2^m)$ Algorithms for Makespan Minimization and Bin Packing

To solve the makespan minimization and Bin packing problems, we modify the algorithm which we described in Sect. 3 for the Santa Claus problem. A common utility for both the problems is the following problem we denote by  $(T, m, n, V)$ : Given each job-machine valuation is given in  $V$ , is it possible to allocate  $m$  jobs (objects) to machines (bins) so that each machine (bin) has a load (volume) of at most  $T$  and number of machines (bins) is  $n$ ? Let us assume that we have this function which tells us whether there is an allocation or not for the given  $(T, m, n, V)$  in  $O^*(2^m)$  time. Then for makespan minimization,  $n$  the number of machines and  $V$  are given in the problem, and performing a binary search on  $T$  to find the smallest  $T$  for which  $(T, m, n, V)$  has an allocation gives us the minimum makespan. This gives us a  $O^*(2^m * \log(ans))$  running time which is still  $O^*(2^m)$ . For Bin packing problem,  $T$  (the capacity of each bin) is given in the problem, and for each job (object), the job-machine evaluation is identical for all machines (bins) as all the bins are identical in case of Bin packing. We find the smallest value of  $n$ , by binary search, for which problem  $(T, m, n, V)$

has an allocation. This gives us a  $O^*(2^m * \log(m))$  running time which is still  $O^*(2^m)$ . So solving the problem  $(T, m, n, V)$  in  $O^*(2^m)$  time gives us a  $O^*(2^m)$  time algorithm for both the problems. We now describe an algorithm to solve this problem.

**Outline.** We now describe the algorithm for makespan minimization, and in the algorithm replacing jobs with objects and machines with bins immediately gives the description for Bin packing. We have a similar dynamic programming approach to the one we used in the Santa Claus problem Sect. 3. Here define  $F(S)$  for a subset  $S$  of jobs to be the pair  $\{p, q\}$  if machines  $[1, p]$  are filled with a value less than or equal to  $T$  (cant be used again) and machine  $p + 1$  is filled upto a value of  $q$  and  $\{p, q\}$  is the minimum possible for the subset  $S$  of jobs. The lesser-than relation  $<$  is defined as follows:  $(a_1, a_2) < (b_1, b_2) \iff a_1 < b_1$  or  $(a_1 = b_1 \wedge a_2 < b_2)$ . To compute  $F(S)$  we consider all subsets  $S'$  which are exactly one element short of  $S$  (i.e.  $\{S' | S' \subseteq S, |S'| = |S| - 1\}$ ). Let  $F(S')$  be  $\{p', q'\}$  for one such  $S' = S \setminus \{x'\}$ . On allocating  $x'$  to machine  $p' + 1$ , check if its load does not exceed  $T$  and update  $F(S)$  accordingly. If the load exceeds  $T$ , then we can not add it to  $p' + 1$ . Since the machines are non-identical, it could happen that the valuation of machine  $p' + 2$  to  $x'$  is more than  $T$ , in which case we will have to look at  $p' + 3$  and so on. If there is a smallest positive integer  $\delta (\delta > 1)$  such that the job can be allocation to machine  $p'' = p' + \delta$ , then one possible value of  $F(S)$  is  $\{p'' - 1, w(p'', x)\}$ . If there does not exist such a  $\delta$ , then  $T$  is an infeasible value for makespan and we proceed to find an allocation for a larger value of  $T$ . In the case of binpacking the the number of bins  $n$  will be inferred as infeasible for the given problem  $(T, m, n, V)$  and we proceed to find an allocation with a larger value of  $n$ .

### 4.1 Dynamic Programming Formulation

Recall that  $J$  is the set of  $m$  jobs, there are  $n$  machines, and  $w(p, q)$  denotes the valuation of job  $p$  to machine  $q$ .

1. We consider all subsets of the set  $J$  of jobs in the increasing order of their cardinality
2. When we consider a subset  $S$ , we compute value of  $F(S)$  as follows.
  - Initialise  $F(S)$  to  $\{\infty, \infty\}$ .
  - Consider all subsets  $S'$  of  $S$  such that  $S' \cup \{x'\} = S$
  - Let  $F(S')$  be  $\{p', q'\}$
  - If  $q' + w(p' + 1, x')$  is less than or equal to  $T$ , then  $F(S) \leftarrow \min(F(S), \{p', q' + w(p' + 1, x')\})$ .
  - Otherwise, let  $p'' (p'' \geq p' + 2)$  be the first machine where  $x'$  can be added with value not exceeding  $T$ . Then update  $F(S) \leftarrow \min(F(S), \{p'' - 1, w(p'', x')\})$ .
3. If  $F(J) \leq \{n - 1, T\}$  then there is an allocation of makespan  $T$ , otherwise there is no allocation of makespan  $T$ .

The following pseudo-code implements the algorithm unambiguously and is useful to calculate the running time of the algorithm.

```

1: function CHECK-SUBROUTINE( $n, m, V, T$ )
  ▷  $n$  machines linearly ordered from 1 to  $n$ ,  $m$  jobs in the set  $J$ 
  ▷  $V$  is the matrix with valuations-  $V[i][j]$  is the valuation of  $i^{th}$  machine to  $j^{th}$  job.
  ▷ Returns true if there is an allocation of all jobs to machines with makespan at most  $T$ 
2:   let  $F$  be an array, indexed by subsets of  $[1, m]$ , of  $2^m$  pairs with initialised to  $\{\infty, \infty\}$ .
3:    $F[\emptyset] \leftarrow \{0, 0\}$ 
4:   for all nonempty  $S \subseteq [1 \cdots m]$  in increasing order of  $|S|$  do
5:     for all  $x' \in S$  do
6:        $S' \leftarrow S \setminus \{x'\}$ 
7:        $\{p', q'\} \leftarrow F[S']$ 
8:       if  $q' + V[p' + 1][x'] \leq T$  then
9:          $F[S] \leftarrow \min(F[S], \{p', q' + V[p' + 1][x']\})$ 
10:      else
11:         $p'' \leftarrow$  smallest index greater than  $p' + 1$  such that  $V[p''] [x'] \leq T$ 
12:         $F[S] \leftarrow \min(F[S], \{p'' - 1, V[p''] [x']\})$ 
13:      end if
14:    end for
15:  end for
16:  if  $F[J] \leq \{n - 1, T\}$  then
17:    return TRUE
18:  else
19:    return FALSE
20:  end if
21: end function

```

**Correctness.** Let  $X_S$  denote an allocation of the subset  $S$  of jobs to machines. Throughout this discussion, we will only look at valid allocations (those which assign a value of at most  $T$  to each machine). Let  $p + 1$  be the last machine (recall that the machines are linearly ordered from 1 to  $n$ ) that gets a non zero value and let the value it gets be  $q$ . Then  $V(X_S)$  is defined to be  $\{p, q\}$ . The proof of the following Lemma and Theorem are similar to the proof of Lemma 2 and we omit the proof in this version of the paper.

**Lemma 3.** *For each subset  $S \subseteq J$ ,  $F[S]$  is at most  $V(X_S)$  for any valid allocation  $X_S$  and in fact the following holds,  $\min_{X_S} V(X_S) = F[S]$ .*

**Theorem 2.** *The function Check-Subroutine correctly checks whether there is an allocation of jobs to the  $n$  machines with makespan at most  $T$  in time  $O^*(2^m)$ .*

*Proof.* From Lemma 3 it follows that for each  $S \subseteq J$ ,  $F[S]$  computed by the function Check-Subroutine( $n, m, V, T$ ) is correct for each value of  $T$ . The function returns true if and only if  $F[S] \leq \{n - 1, T\}$ .

**Running Time Analysis.** The running time is governed by the subproblem for each subset of  $J$ . For each subset  $S$ , the time taken to compute  $F(S)$  accounts

for  $O(m)$  time, which is for each subset  $S' \subset S$  of cardinality one smaller than that of  $S$ . So the complexity of Check-Subroutine( $n, m, V, T$ ) is again  $O^*(2^m)$ .

Therefore, we can binary search on the range of values for  $T$  or  $n$  and minimize makespan or the number of bins, respectively.

## 5 Santa Claus Problem with Capacity Constraints

In this section we present a  $O^*(2^m)$  algorithm to find the optimum max-min allocation of jobs along with the constraint that each machine should get at most  $k$  jobs, where  $k$  is an additional input parameter. Clearly when  $k = m$  we get the Santa Claus problem studied in Sect. 3. As in the previous algorithms the problem of finding an optimum allocation is reduced to checking if there is a feasible allocation of a fixed value  $T$ , and we show in Theorem 3 this can be done in time  $O^*(2^m)$ . Recall that the machines are considered according to a fixed linear order by the algorithm.

**Structure of an Optimum Allocation.** For a subset  $S$  of jobs and  $0 \leq i \leq k$ , let  $X_S^i$  denote an allocation of the set  $S$  of jobs to the machines. For an allocation  $X_S^i$  let us consider the prefix of machines in the linear order that are allocated a non-empty set of jobs. Let  $c(X_S^i)$  denote the number of machines in this prefix. An allocation  $X_S^i$  of the set  $S$  of jobs to the machines is defined to be a valid allocation if it satisfies that following conditions:

- Each machine is allocated at most  $k$  jobs.
- Each of the first  $c(X_S^i) - 1$  machines is allocated a set of jobs of total value at least  $T$ .
- If  $i > 0$ , the  $c(X_S^i)$ -th machine is allocated a set of  $i$  jobs.
- If  $i = 0$ , then the  $c(X_S^i)$ -th machine is allocated a set of jobs of total value at least  $T$ .

The value of a valid allocation  $X_S^i$  denoted by  $V(X_S^i) = \{p, q\}$ , where machines 1 to  $p$  are allocated a set of at most  $k$  jobs of total value at least  $T$  and machine  $p + 1$  is allocated a set of jobs of total value  $q$  and exactly  $i$  jobs.

**Note:** Let us consider an allocation  $X_S^i$  where  $i \geq 1$  and  $V(X_S^i) = \{p, q\}$ . If  $q \geq T$ , then the allocation  $X_S^i$  is also represented as  $X_S^0$  and  $V(X_S^0) = \{p + 1, 0\}$ . This representation is symmetric as follows: Given a valid allocation  $X_S^0$  of value  $\{p, 0\}$ , the  $i$  for which it can be represented as  $X_S^i$  is well-defined- the value of  $i$  is the number of jobs allocated to the machine  $p$ . Let the value of the jobs assigned to  $p$  be  $q$ . The value of the allocation  $X_S^i$  is  $\{p - 1, q\}$ . For an  $X_S^0$  the corresponding  $X_S^i$  as defined here plays a very crucial role in the second case of Lemma 4.

For each  $i \in [0, k]$  and each  $S \subseteq J$ , define  $F(S, i)$ , to be the maximum valued pair  $\{p, q\}$  such that there is valid allocation of jobs in  $S$  with value  $\{p, q\}$  where

machine  $p$  has exactly  $i$  jobs. Recall that the maximum is defined according to the comparison operator where  $(a_1, a_2) > (b_1, b_2)$  if  $a_1 > b_1$  or  $(a_1 = b_1 \wedge a_2 > b_2)$ . At this point, we would like to contrast the  $F(S, i)$  and  $F(S)$  in the Santa Claus problem in Sect. 3 where  $F(S)$  was  $\{p, q\}$ ,  $q$  had to be less than  $T$ . However, in the definition of  $F(S, i)$ , it might happen that  $q$  is more than  $T$ . This is because we need the best pair which corresponds to an allocation in which the last machine has been allocated  $i$  jobs.

We formally define  $F(S, i)$  for each  $S \subseteq J$  and  $0 \leq i \leq k$  as follows. To define  $F(S, 0)$ , we consider the pair  $\{p, q\}$  which is defined to be the maximum of  $F(S, j)$  for all  $j \in [1, k]$ .

$$\begin{aligned} F(S, 0) &= \{0, 0\}, \text{ if } S = \emptyset \\ F(S, 0) &= \{p, 0\}, \text{ if } q < T \\ &= \{p + 1, 0\}, \text{ otherwise.} \end{aligned}$$

For a set  $S'$  such that  $S = S' \cup \{x'\}$ , let  $F(S', i - 1)$  be denoted by  $\{p', q'\}$ . Now for  $S \subseteq J$  and  $1 \leq i \leq k$ , we define the function  $F[S][i]$  recursively as follows,

$$\begin{aligned} F(S, i) &= \{0, 0\}, \text{ if } S = \emptyset \\ F(S, i) &= \max_{S' \cup \{x'\} = S} \max((F(S', i), \{p', q' + w(p' + 1, x')\})) \end{aligned}$$

Finally if  $F(J, 0)$  is  $\{n, 0\}$  then the value  $T$  is possible and otherwise it is not. The following algorithm describes the bottom-up computation of the values of  $F[S][i]$ .

```

1: function CHECK-VALIDITY-CAPACITY-SANTA-CLAUS( $n, m, V, T, k$ )
   ▷  $n$  machines each of capacity  $k$ ,  $m$  jobs in  $J$ .
   ▷  $V$  is the matrix with valuations.  $V[i][j]$  is the valuation of  $i^{\text{th}}$  machine to  $j^{\text{th}}$  job.
   ▷ Returns true if there is an allocation in which each machine is allocated at most  $k$  jobs and at least a load of  $T$ 
2:   let  $F$  be a  $2^m \times (k + 1)$ -array of pairs with initialised to  $\{-\infty, -\infty\}$ .
3:    $F[\emptyset][0 \dots k] \leftarrow \{0, 0\}$ 
4:   for all nonempty  $S \subseteq [1 \dots m]$  in increasing order of  $|S|$  do
5:      $\{p, q\} \leftarrow \{0, 0\}$ 
6:     for all  $i \in [1, k]$  do
7:       for all  $x' \in S$  do
8:          $S' \leftarrow S \setminus \{x'\}$ 
9:          $\{p', q'\} \leftarrow F[S'][i - 1]$ 
10:         $F[S][i] \leftarrow \max(F[S][i], F[S'][i])$ 
11:         $F[S][i] \leftarrow \max(F[S][i], \{p', q' + V[p' + 1][x']\})$ 
12:      end for
13:       $\{p, q\} \leftarrow \max(\{p, q\}, F[S][i])$ 
14:    end for
15:    if  $q < T$  then
16:       $F[S][0] \leftarrow \{p, 0\}$ 

```

```

17:     else
18:          $F[S][0] \leftarrow \{p + 1, 0\}$ 
19:     end if
20: end for
21: if  $F[J][0] = \{n, 0\}$  then
22:     return TRUE
23: else
24:     return FALSE
25: end if
26: end function

```

### Correctness

**Lemma 4.** *For each subset  $S \subseteq J$  and  $0 \leq i \leq k$ ,  $F(S, i)$  computed by function Check-Validity-Capacity-Santa-Claus has a value at least the value of any valid allocation  $X_S^i$  of jobs in  $S$  to the machines.*

*Proof.* We will prove this by induction on the cardinality of  $S \subseteq J$ . The base case is when  $S = \emptyset$ ,  $F(S, 0)$  is  $\{0, 0\}$  which is as good as the value of the only allocation  $X_S^0$  of the empty set which can satisfy 0 machines. Let us assume that the lemma is true for all  $S \subseteq J$  with  $|S| < b$ . Given this assumption, we show that for each set  $S$  with  $b$  jobs and each valid allocation  $X_S^i$ ,  $F(S, i) \geq V(X_S^i)$ .

Case 1:  $i > 0$ . Let  $V(X_S^i)$  be denoted by  $\{p, q\}$ . Now, let  $x$  be any one of the jobs assigned to machine  $p + 1$ . If we remove the allocation of job  $x$  to machine  $p + 1$ , we get a valid allocation  $X_{S'}^{i-1}$  where machine  $p + 1$  has  $i - 1$  jobs with  $V(X_{S'}^{i-1}) = \{p, q - w(p + 1, x)\}$ . By the induction hypothesis, it follows that  $F(S', i - 1) \geq V(X_{S'}^{i-1}) = \{p, q - w(p + 1, x)\}$ . Let  $F(S', i - 1)$  be denoted by  $\{p', q'\}$ . In the function Check-Validity-Capacity-Santa-Claus  $F(S, i)$  is updated by the statement  $F(S, i) \leftarrow \max(F(S, i), \{p', q' + w(p' + 1, x)\})$ . Since  $\{p', q'\} \geq \{p, q - w(p + 1, x)\}$ , we show that  $F(S, i) \geq \{p, q\}$  based on the only two possible cases: when  $p' = p$  and when  $p' > p$ . If  $p' > p$ , then clearly  $F(S, i) \geq \{p', q' + w(p' + 1, x)\}$  is at least  $\{p, q\}$ , due to the definition of the greater-than relation between pairs. In the case when  $p' = p$ , since  $\{p', q'\} \geq \{p, q - w(p + 1, x)\}$ , it follows that  $q' \geq q - w(p + 1, x)$ . Therefore it follows that  $\{p', q' + w(p' + 1, x)\} \geq \{p, q\}$ . Consequently it follows that  $F(S, i) \geq V(X_S^i)$ , thus completing the proof by induction in this case.

Case 2:  $i = 0$ . Let  $V(X_S^0) = \{p, 0\}$ . Let machine  $p$  be allocated  $l$  jobs which have a total value of  $q$ . Since  $X_S^0$  is a valid allocation,  $l \leq k$  and  $q \geq T$ . Clearly, the allocation  $X_S^0$  can be represented as  $X_S^l$  (see Note just after definition of the value of  $X_S^i$ ) and  $V(X_S^l) = \{p - 1, q\}$ . Now, by applying the conclusion in Case 1, it follows that  $F(S, l) \geq V(X_S^l) = \{p - 1, q\} \geq \{p - 1, T\}$ . In the function Check-Validity-Capacity-Santa-Claus  $F(S, 0)$  is updated by taking the maximum value of  $F(S, j)$  for all  $j \in [1, k]$ . Therefore  $F(S, 0) \geq F(S, l)$ . Further, since  $F(S, l) \geq \{p - 1, T\}$  the If-statement in Line 15 ensures that  $F(S, 0) = \{p, 0\} = V(X_S^0)$ . Hence the proof by induction is complete in this case.

**Theorem 3.** *The function Check-Validity-Capacity-Santa-Claus correctly checks whether there is a valid allocation in time  $O^*(2^m)$ .*

*Proof.* From Lemma 4 we know that for each  $S \subseteq J$  and  $1 \leq i \leq k$ , the value for  $F(S, i)$  computed by the algorithm is the maximum value possible by any valid allocation. Further, the function returns true if and only if  $F[S][0] = \{n, 0\}$ . This completes the proof that function correctly checks if there is a valid allocation (that assigns a load of  $T$  to each machine and at most  $k$  jobs to each machine).

**Complexity Analysis.** Each entry in the  $2^m \times (k + 1)$ -array  $F$  is calculated in  $O(m)$  time. Therefore, the function takes time  $O(2^m * k * m)$  and this is  $O^*(2^m)$ . Hence the theorem is proved.

## References

1. Annamalai, C., Kalaitzis, C., Svensson, O.: Combinatorial algorithm for restricted max-min fair allocation. CoRR, abs/1409.0607 (2014). <http://arxiv.org/abs/1409.0607>
2. Asadpour, A., Feige, U., Saberi, A.: Santa Claus meets hypergraph matchings. ACM Trans. Algorithms **8**(3), 24:1–24:9 (2012). <https://doi.org/10.1145/2229163.2229168>
3. Bansal, N., Sviridenko, M.: The Santa Claus problem. In: Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 31–40. ACM, New York (2006). <https://doi.org/10.1145/1132516.1132522>
4. Bezáková, I., Dani, V.: Allocating indivisible goods. ACM SIGecom Exch. **5**(3), 11–18 (2005)
5. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms, 1st edn. Springer, Heidelberg (2010). <https://doi.org/10.1007/978-3-642-16533-7>
6. Jansen, K., Land, F., Land, K.: Bounding the running time of algorithms for scheduling and packing problems. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 439–450. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40104-6\\_38](https://doi.org/10.1007/978-3-642-40104-6_38)
7. Lenté, C., Liedloff, M., Soukhal, A., T'Kindt, V.: Exponential algorithms for scheduling problems. HAL (2014). <https://hal-univ-tours.archives-ouvertes.fr/hal-00944382>
8. Woeginger, G.J.: Exact algorithms for NP-hard problems: a survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization — Eureka, You Shrink!. LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-36478-1\\_17](https://doi.org/10.1007/3-540-36478-1_17). <http://dl.acm.org/citation.cfm?id=885909.885927>





# Exact Algorithms for the Max-Min Dispersion Problem

Toshihiro Akagi<sup>1</sup>, Tetsuya Araki<sup>2</sup>, Takashi Horiyama<sup>3</sup>, Shin-ichi Nakano<sup>1</sup>(✉),  
Yoshio Okamoto<sup>4,5</sup>, Yota Otachi<sup>6</sup>, Toshiki Saitoh<sup>7</sup>, Ryuhei Uehara<sup>8</sup>,  
Takeaki Uno<sup>9</sup>, and Kunihiro Wasa<sup>9</sup>

<sup>1</sup> Gunma University, Kiryu, Japan  
nakano@cs.gunma-u.ac.jp

<sup>2</sup> Tokyo Metropolitan University, Hachioji, Japan

<sup>3</sup> Saitama University, Saitama, Japan

<sup>4</sup> The University of Electro-Communications, Chofu, Japan

<sup>5</sup> RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

<sup>6</sup> Kumamoto University, Kumamoto, Japan

<sup>7</sup> Kyushu Institute of Technology, Kitakyushu, Japan

<sup>8</sup> JAIST, Nomi, Japan

<sup>9</sup> National Institute of Informatics, Tokyo, Japan

**Abstract.** Given a set  $P$  of  $n$  elements, and a function  $d$  that assigns a non-negative real number  $d(p, q)$  for each pair of elements  $p, q \in P$ , we want to find a subset  $S \subseteq P$  with  $|S| = k$  such that  $\text{cost}(S) = \min\{d(p, q) \mid p, q \in S\}$  is maximized. This is the max-min  $k$ -dispersion problem. In this paper, exact algorithms for the max-min  $k$ -dispersion problem are studied. We first show the max-min  $k$ -dispersion problem can be solved in  $O(n^{\omega k/3} \log n)$  time. Then, we show two special cases in which we can solve the problem quickly. Namely, we study the cases where a set of  $n$  points lie on a line and where a set of  $n$  points lie on a circle (and the distance is measured by the shortest arc length on the circle). We obtain  $O(n)$ -time algorithms after sorting.

**Keywords:** Dispersion problem · Algorithm

## 1 Introduction

The facility location problem and many of its variants have been studied extensively [8, 9]. Typically, we are given a set of locations on which facilities can be placed and an integer  $k$ , we want to place  $k$  facilities on some locations in such a way that a given objective is minimized. In the *dispersion* problem, we want to minimize the interference between the placed facilities. As an example scenario, consider that we are planning to open several chain stores in a city. We wish to locate the stores mutually far away from each other to avoid self-competition. Another example is a case where facilities are mutually obnoxious, such as nuclear power plants and oil storage tanks. See more applications, including *result diversification*, in [6, 16, 21, 23].

More specifically, in the *max-min  $k$ -dispersion problem*, we are given a set  $P$  of  $n$  elements which represent possible locations, and a function  $d$  that assigns a non-negative real number  $d(p, q)$  for each pair of elements  $p, q \in P$ . Throughout this paper, we assume that  $d$  is symmetric (i.e.,  $d(p, q) = d(q, p)$  for all  $p, q \in P$ , and  $d(p, p) = 0$  for all  $p \in P$ ), but we do not assume that  $d$  satisfies the triangle inequality. The value  $d(p, q)$  represents the distance between  $p$  and  $q$ . We are also given an integer  $k$  with  $k \leq n$ . Then, we want to find a subset  $S \subseteq P$  with  $|S| = k$  such that  $\text{cost}(S) = \min\{d(p, q) \mid p, q \in S\}$  is maximized.

The max-min  $k$ -dispersion problem was recognized in the early days of research for location theory. At least, Shier [22] wrote and published a paper about  $k$ -dispersion on trees in 1977, and connected the problem with the  $k$ -center problem. The related literature up to the mid 1980's was reviewed by Kuby [15]. Erkut [10] proved the problem is NP-hard even when the triangle inequality is satisfied. A geometric version was studied by Wang and Kuo [25], where points lie in the  $d$ -dimensional space, and the distance is Euclidean. Then, they proved the following: when  $d = 1$ , the problem can be solved in  $O(kn)$  time by dynamic programming after  $O(n \log n)$ -time sorting; when  $d = 2$ , the problem is NP-hard. The running time for  $d = 1$  was recently improved to  $O(n \log \log n)$  (after sorting) [2] by sorted matrix search method [13]. (For a good survey for the sorted matrix search method see [1, Sect. 3.3].)

Ravi et al. [21] proved that the max-min  $k$ -dispersion cannot be approximated within any factor in polynomial time, and cannot be approximated within the factor of two in polynomial time when the distance satisfies the triangle inequality, unless  $P = NP$ . They also gave a polynomial-time algorithm with approximation ratio two when the triangle inequality is satisfied. Thus, the factor of two is tight.

In the *max-sum  $k$ -dispersion problem*, the objective is to maximize the sum of distances between  $k$  facilities. The proof by Erkut [10] can easily be adapted to show that the max-sum  $k$ -dispersion problem is NP-hard. It is not known whether the problem is still NP-hard on the 2-dimensional Euclidean space. Ravi et al. [21] gave an  $O(n \log n + kn)$ -time exact algorithm when the points lie on a line, a polynomial-time factor-four approximation algorithm when the triangle inequality is satisfied, and a polynomial-time factor- $(\pi/2 + \epsilon)$  approximation algorithm for the 2-dimensional Euclidean space (note that  $\pi/2 \approx 1.571$ ). The factor of four was improved to two by Birnbaum and Goldman [4] and Hassin et al. [14]. Fekete and Meijer [11] studied the case for the  $d$ -dimensional space with the  $L_1$  distance, and gave an  $O(n)$ -time exact algorithm when  $k$  is fixed (after sorting the points by  $x$ -coordinates and  $y$ -coordinates), and a polynomial-time approximation scheme when  $k$  is part of the input. Polynomial-time approximation schemes for the Euclidean distance, or more generally for the negative-type metric were given by Cevallos et al. [5, 6]. For other variations, see [3, 7].

In this paper, exact algorithms for the max-min  $k$ -dispersion problem are studied. The main contributions are twofold.

First, we review an intimate relationship between the max-min  $k$ -dispersion problem and the maximum independent set problem. A reduction to prove the NP-hardness of the max-min  $k$ -dispersion problem uses the  $k$ -independent set

problem [21], and we prove the reverse reduction is possible. Namely, we show that if the  $k$ -independent set problem can be solved in  $T(n, k)$  time, then the max-min  $k$ -dispersion problem can be solved in  $O((T(n, k) + n^2) \log n)$  time. Note that the  $k$ -independent set problem can be solved in  $O(n^{\omega k/3})$  time [20], where  $\omega < 2.373$  is the matrix multiplication exponent (See Le Gall [17] for the current best bound on  $\omega$ ). Therefore, our result implies that the max-min  $k$ -dispersion problem can be solved in  $O(n^{\omega k/3} \log n)$  time. We will also discuss some consequences of these reductions.

Then, we turn our attention to two special cases in which we can solve the problem quickly.

We study the case when a set of  $n$  points lie on a line, and obtain an  $O(n)$ -time algorithm after sorting. This is an improvement over the recent  $O(n \log \log n)$ -time algorithm [2]. In our algorithm, we employ the tree partitioning algorithm by Frederickson [13]. Next, we consider the case when a set of  $n$  points lies on a circle on the Euclidean plane, and the distance is measure by the shortest arc length on the circle. Then, we obtain an  $O(n)$ -time algorithm after sorting.

The remainder of this paper is organized as follows. In Sect. 2, we consider a relationship between the max-min  $k$ -dispersion problem and the  $k$ -independent set problem. Section 3 gives an algorithm to solve the max-min dispersion problem when  $P$  is a set of points on a line. Section 4 gives an algorithm to solve the max-min dispersion problem when  $P$  is a set of points on a circle. Finally, Sect. 5 concludes the paper.

## 2 Relationship with the Maximum Independent Set Problem

### 2.1 General Case

First, we reduce the max-min  $k$ -dispersion problem to the  $k$ -independent set problem. Here, we remind the definition of the max-min  $k$ -dispersion problem. In the max-min  $k$ -dispersion problem, we are given a set  $P$  of  $n$  elements, a function  $d$  that assigns a non-negative real number  $d(p, q)$  for each pair of elements  $p, q$  of  $P$ , and an integer  $k$  such that  $k \leq n$ . Then, we want to find a subset  $S \subseteq P$  with  $|S| = k$  that maximizes  $\text{cost}(S) = \min\{d(p, q) \mid p, q \in S\}$ .

In  $k$ -independent set problem, we are given an undirected graph  $G = (V, E)$ , and we want to determine whether there exists a subset  $S \subseteq V$  of  $k$  vertices such that each pair of vertices in  $S$  is non-adjacent, and find such a set  $S$  if exists.

For our reduction, we consider the following question  $Q(r)$  for a given real number  $r$ : does there exist a set of  $k$  locations such that the distance of any two locations is at least  $r$ ?

Observe that the optimal value for the max-min  $k$ -dispersion problem is the maximum of  $r$  such that the answer to  $Q(r)$  is yes. We also observe that the optimal value lies in the set of possible distances  $\{d(p, q) \mid p, q \in P\}$ , and the answers to  $Q(r)$  have the following monotonicity: if the answer to  $Q(r)$  is yes, and  $r' < r$ , then the answer to  $Q(r')$  is also yes. Therefore, if  $Q(r)$  can be answered

correctly for any  $r$ , then we can solve the max-min  $k$ -dispersion problem by performing binary search over the all possible  $O(n^2)$  candidates after sorting the distances  $d(p, q)$ . The sorting takes  $O(n^2 \log n^2) = O(n^2 \log n)$  time.

To answer the question  $Q(r)$  above, we use the  $k$ -independent set problem. To this end, we construct the following undirected graph  $G(r) = (V, E)$ . The vertex set  $V$  is identical to  $P$ . Two locations  $p, q \in P$  are joined by an edge in  $G(r)$  if and only if  $d(p, q) < r$ . Remind that we have the symmetry assumption  $d(p, q) = d(q, p)$  for all  $p, q \in P$ , and thus the undirected graph is well-defined.

Let  $S \subseteq V$  be an independent set of size  $k$  in  $G(r)$ . Then, by definition, every pair of two vertices  $p, q \in S$  satisfies  $d(p, q) \geq r$ . This implies that  $\text{cost}(S) \geq r$ , and the answer to  $Q(r)$  is yes. On the other hand, if the answer to  $Q(r)$  is yes, then there exists a set  $S$  of  $k$  locations such that  $d(p, q) \geq r$  for all  $p, q \in S$ . This means that  $S$  is an independent set in  $G(r)$ . Therefore, it follows that  $G(r)$  has an independent set of size  $k$  if and only if the answer to  $Q(r)$  is yes.

Now we analyze the running time. We assume that the  $k$ -independent set problem can be solved in  $T(n, k)$  time on  $n$ -vertex undirected graphs. Sorting the distances takes  $O(n^2 \log n)$  time as discussed above. Then, we perform binary search to find the maximum  $r$  such that the answer to  $Q(r)$  is yes among the  $O(n^2)$  candidates for  $r$ . The number of iterations is  $O(\log n^2) = O(\log n)$ . For each iteration, we construct the graph  $G(r)$ , which takes  $O(n^2)$  time, and solve the  $k$ -independent set problem, which takes  $T(n, k)$  time. Therefore, the overall running time is  $O(n^2 \log n + (n^2 + T(n, k)) \log n) = O((T(n, k) + n^2) \log n)$ .

The current best bound for  $T(n, k)$  is  $O(n^{\omega k/3})$  [20], where  $\omega$  is the matrix multiplication exponent. Since  $\omega \geq 2$ , we obtain the following theorem.

**Theorem 1.** *The max-min  $k$ -dispersion problem can be solved in  $O(n^{\omega k/3} \log n)$  time.*

On the other hand, the  $k$ -independent set problem can be reduced to the max-min  $k$ -dispersion problem as follows [21]. Let  $G = (V, E)$  be an undirected graph given as an instance of the  $k$ -independent set problem. Then, we construct the following instance of the max-min  $k$ -dispersion problem. The set of locations is  $V$ . The distance  $d(p, q)$  between  $p, q \in V$  is defined as follows:  $d(p, q) = 1$  if  $p$  and  $q$  are adjacent in  $G$ , and  $d(p, q) = 2$  otherwise. Then,  $G$  has an independent set  $S$  of size  $k$  if and only if there exists a set  $S$  of  $k$  locations such that  $\text{cost}(S) = 2$ .

This reduction does not only prove the NP-hardness of the max-min  $k$ -dispersion problem, but also proves the W[1]-hardness of the problem, when  $k$  is a parameter, as the  $k$ -independent set problem is W[1]-hard when  $k$  is a parameter. W[1]-hardness is a concept in parameterized complexity theory. Refer to [12].

In summary, up to a logarithmic-factor overhead in the running time, the max-min  $k$ -dispersion problem is equivalent to the  $k$ -independent set problem.

## 2.2 On the Euclidean Plane

The discussion above also applies to some special cases. We now look at the case when  $P$  is a set of points on the Euclidean plane.

We look at the construction of the graph  $G(r)$  more carefully. Remind that in  $G(r)$ , two vertices  $p, q \in P$  are joined by an edge if and only if  $d(p, q) < r$ . This matches the definition of a *unit disk graph*. Here, we remind the definition of a unit disk graph. A unit disk graph is an undirected graph defined by a set of unit disks. The vertex set is the set of unit disks, and two disks are joined by an edge if and only if the disks intersect. Usually, disks are considered to be closed, but the results below also hold for open disks.

To view  $G(r)$  as a unit disk graph, we consider an open disk of radius  $r/2$  that has a center at each point  $p \in P$ . Then, two such disks centered at  $p, q \in P$  intersect if and only if  $d(p, q) < r$ . If we scale the whole picture by the factor of  $2/r$ , then we obtain  $G(r)$  as a unit disk graph.

It is known that the  $k$ -independent set problem on unit disk graphs can be solved in  $n^{O(\sqrt{k})}$  time [18]. Therefore, from the discussion above, we obtain the following theorem.

**Theorem 2.** *The max-min  $k$ -dispersion problem can be solved in  $n^{O(\sqrt{k})}$  time when  $P$  lies on the Euclidean plane.*

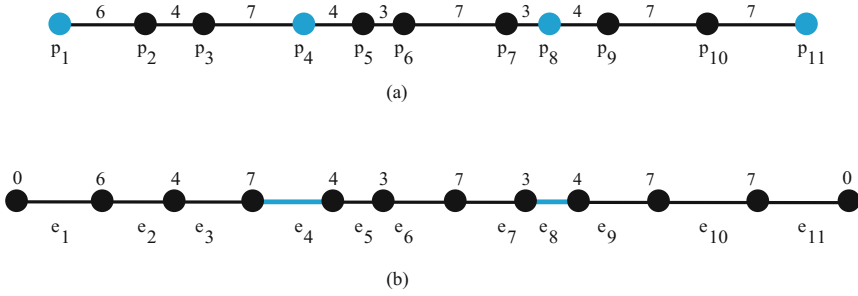
On the other hand, if the optimal value for the max-min  $k$ -dispersion problem is  $r$ , then there exists a set of  $k$  pairwise disjoint open disks of radius  $r/2$  that have their centers in  $P$ . This means that if the max-min  $k$ -dispersion problem can be solved in  $T(n, k)$  time when  $P$  is a set of points on the Euclidean plane, then the  $k$ -independent set problem on unit disk graphs can be solved in  $T(n, k)$  time, too. Since the  $k$ -independent set problem on unit disk graphs cannot be solved in  $n^{o(\sqrt{k})}$  time under the exponential time hypothesis [19], we can also conclude that the max-min  $k$ -dispersion problem on the Euclidean plane cannot be solved in  $n^{o(\sqrt{k})}$  time under the exponential time hypothesis. Thus, the running time in Theorem 2 is essentially optimal.

### 3 Max-Min Dispersion on a Line

In this section we show one can solve the  $k$ -dispersion problem in  $O(n)$  time if  $P$  is a set of points on a line and the order of  $P$  on the line is given. The idea of our algorithm is a reduction to the path partitioning problem [13], which can be solved in  $O(n)$  time.

Let  $T$  be a tree in which each vertex  $v$  has a non-negative weight  $w(v)$ , and  $k$  be an integer. The tree  $k$ -partitioning problem is to delete  $k - 1$  edges in the tree so as to maximize the lightest weight of the remaining subtrees, where the weight of a subtree is defined by the sum of the weights of its vertices. The tree  $k$ -partitioning problem can be solved in  $O(n)$  time [13], where  $n$  is the number of vertices in the tree. If the input tree is a path, then it is the *path  $k$ -partitioning problem*.

Given an instance  $(P, k)$  of the max-min  $k$ -dispersion problem such that  $P$  is a set of points on a line and  $k \geq 3$ , we can transform it to an instance  $(P', k - 1)$  of the path  $(k - 1)$ -partitioning problem as follows. First, we construct a path



**Fig. 1.** (a) A 4-dispersion problem on a line, and (b) the corresponding 3 path-partitioning problem on a line.

$P' = (V', E')$ . Assume  $P = \{p_1, p_2, \dots, p_n\}$  and the points appear in this order on the line from left to right. Define  $V' = \{p'_0, p'_1, \dots, p'_n\}$ ,  $E' = \{e_i = (p'_{i-1}, p'_i) \mid p_i \in V\}$ ,  $w(p'_i) = d(p_i, p_{i+1})$  for each  $i = 1, 2, \dots, n - 1$ , and  $w(p'_0) = w(p'_n) = 0$ . See an example in Fig. 1. A solution of the max-min 4-dispersion problem in Fig. 1(a) is  $\{p_1, p_4, p_8, p_{11}\}$  and its cost is 17. A solution of the path 3-partitioning problem in Fig. 1(b) is  $\{e_4, e_8\}$  and its cost is 17. One can observe that an optimal solution of the max-min  $k$ -dispersion problem contains  $\{p_1, p_n\}$ , and if an optimal solution of the max-min  $k$ -dispersion problem is  $\{p_1, p_n\} \cup \{p_{i_1}, p_{i_2}, \dots, p_{i_{k-2}}\}$ , then a solution of the path  $k - 1$ -partitioning problem is  $\{e_{i_1}, e_{i_2}, \dots, e_{i_{k-2}}\}$ .

Since one can solve the path  $k$ -partitioning problem in  $O(n)$  time [13], one can solve the max-min  $k$ -dispersion problem in  $O(n)$  time.

**Theorem 3.** *The max-min  $k$ -dispersion problem can be solved in  $O(n)$  time when  $P$  is a set of  $n$  points on a line and the order of  $P$  on the line is given.*

### 4 Max-Min $k$ -dispersion on a Circle

In this section, we show one can solve the  $k$ -dispersion problem in  $O(n)$  time if  $P$  is a set of points on a circle and the order of  $P$  on the circle is given. The distance is measured by the arc length of the circle. We assume (1) the length of the circumference, and (2) the length of the clockwise arc from some designated point to each point are given. So, one can compute the central angle corresponding to a given arc in constant time.

Let  $P$  be a set of points on a circle and each point has a non-negative weight, and  $k$  be an integer. The circle partitioning problem deletes  $k$  edges on the circle so as to maximize the lightest weight of the remaining subpath. So, this is the circle version of the path  $k$ -partition problem, explained in Sect. 3. One can solve the circle partition problem in  $O(n)$  time [24].

**Theorem 4.** *One can solve the max-min  $k$ -dispersion problem in  $O(n)$  time when  $P$  is a set of points on a circle and the order of the points on the circle is given.*

The algorithm is rather complicated. Therefore, we seek for a simpler algorithm for  $k = 3$ , which runs  $O(n)$  time.

The outline of the algorithm is as follows. For each point  $p_i \in P$ , we compute the best three points including  $p_i$ , and then output the best three points among them.

---

**Algorithm 1. Find-3-dispersion-on-a-circle( $P, k$ )**

---

```

cost = 0
Ans = ∅
for  $i = 1$  to  $n$  do
    find a set  $S$  of three points including  $p_i$  with maximum  $\text{cost}(S)$ 
    if  $\text{cost}(S) > \text{cost}$  then
        cost =  $\text{cost}(S)$ 
        Ans =  $S$ 
    end if
end for
Output Ans

```

---

Now we introduce some definitions. Given  $p_i \in P$ , let  $p_i^\ell$  and  $p_i^r$  be the points on the circle such that  $p_i, p_i^\ell, p_i^r$  are the three corners of the equilateral triangle. Let  $A_\ell = (p_i, p_i^\ell)$  be the arc of the circle between  $p_i$  and  $p_i^\ell$  with central angle  $120^\circ$ ,  $A_r = (p_i, p_i^r)$  be the arc of the circle between  $p_i$  and  $p_i^r$  with central angle  $120^\circ$ , and  $A_t = (p_i^\ell, p_i^r)$  be the (open) arc of the circle between  $p_i^\ell$  and  $p_i^r$  with central angle  $120^\circ$ .

A set  $S = \{p_i, p_\ell, p_r\}$  is of *type-LR* with respect to  $p_i$  if  $p_\ell \in A_\ell$  and  $p_r \in A_r$ . Similarly,  $S$  is of *type-LL* with respect to  $p_i$  if  $p_\ell \in A_\ell$  and  $p_r \in A_\ell$ , is of *type-RR* with respect to  $p_i$  if  $p_\ell \in A_r$  and  $p_r \in A_r$ , and is of *type-LT* with respect to  $p_i$  if  $p_\ell \in A_\ell$  and  $p_r \in A_t$ , is of *type-TT* with respect to  $p_i$  if  $p_\ell \in A_t$  and  $p_r \in A_t$ , etc.

We have the following lemma.

**Lemma 1.** *When  $P$  is a set of points on a circle, an optimal solution  $S$  of the max-min 3-dispersion problem is of type-LR or type-TT with respect to some  $p_i \in S$ .*

*Proof.* By case analysis. Assume  $S = \{p_i, p_\ell, p_r\}$  and  $p_i, p_\ell$  and  $p_r$  appear in the clockwise order on the circle. If  $S$  is of type-LL with respect to  $p_i$ , then  $S$  is of type-LR with respect to  $p_\ell$  (Fig. 2(a)). If  $S$  is of type-RR with respect to  $p_i$ , then  $S$  is of type-LR with respect to  $p_r$  (Fig. 2(b)). If  $S$  is of type-LT with respect to  $p_i$ , then either (1) the arc of the circle between  $p_\ell$  and  $p_r$  has the central angle less than  $120^\circ$  and  $S$  is of type-LR with respect to  $p_\ell$  (Fig. 2(c)), or (2) the arc of the circle between  $p_\ell$  and  $p_r$  has the central angle at least  $120^\circ$  and  $S$  is of type-TT with respect to  $p_r$  (Fig. 2(d)). If  $S$  is type-TR with respect to  $p_i$ , then we can prove either (1)  $S$  is of type-LR with respect to  $p_r$  (Fig. 2(e)), or (2)  $S$  is of type-TT with respect to  $p_\ell$  (Fig. 2(f)). **Q.E.D.**

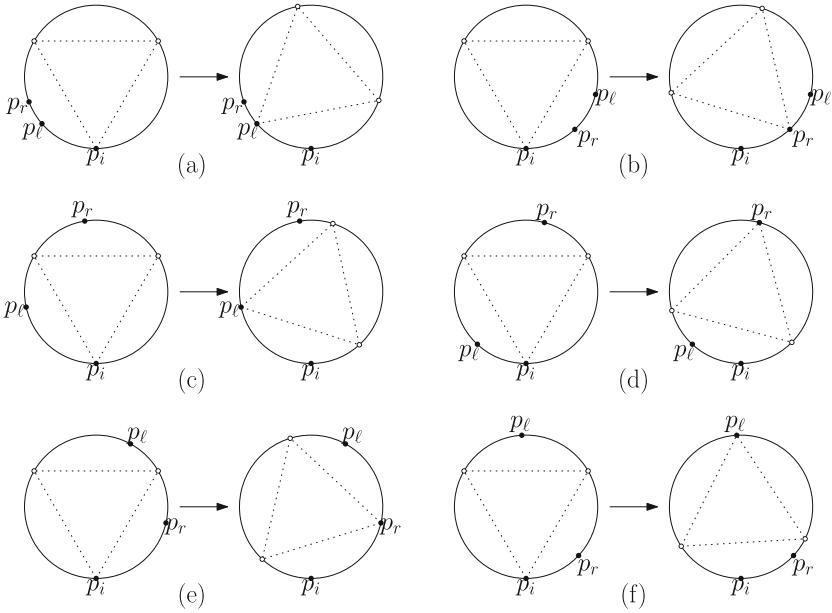


Fig. 2. Proof of Lemma 1.

- Lemma 2.** (a) If an optimal solution  $S = \{p_i, p_\ell, p_r\}$  is of type-*TT* with respect to  $p_i$ , then  $p_\ell$  is the first point in  $P$  on the circle after  $p_i^\ell$  in the clockwise order, and  $p_r$  is the first point in  $P$  on the circle after  $p_i^r$  in the counterclockwise order.
- (b) If an optimal solution  $S = \{p_i, p_\ell, p_r\}$  is of type-*LR* with respect to  $p_i$ , then  $p_\ell$  is the first point in  $P$  on the circle after  $p_i^\ell$  in the counterclockwise order, and  $p_r$  is the first point in  $P$  on the circle after  $p_i^r$  in the clockwise order.

*Proof.* (a) Since  $S$  is of type-*TT*,  $\min\{d(p_i, p_\ell), d(p_\ell, p_r), d(p_r, p_i)\} = d(p_\ell, p_r)$  holds. Thus, choosing  $S$  so as to maximize  $d(p_\ell, p_r)$  results in the  $S$  with the maximum cost.

(b) Since  $S$  is of type-*LR*,  $\min\{d(p_i, p_\ell), d(p_\ell, p_r), d(p_r, p_i)\} \neq d(p_\ell, p_r)$  holds. Thus, choosing  $S$  so as to maximize  $d(p_i, p_\ell)$  and  $d(p_i, p_r)$  results in  $S$  with the maximum cost. **Q.E.D.**

We need to compute for each  $p_i$  the first point in  $P$  on the circle after  $p_i^\ell$  in the clockwise order, and it takes  $O(n)$  time in total. Similarly, we can compute for each  $p_i$  the first point in  $P$  on the circle after  $p_i^r$  in the counterclockwise order, and it takes  $O(n)$  time in total. We perform this as preprocessing.

Then, for each  $p_i$  we need  $O(1)$  time to find (1) the set  $S$  of three points of type-*LR* with respect to  $p_i$  with maximum  $\text{cost}(S)$ , and (2) the set  $S$  of three points of type-*TT* with respect to  $p_i$  with maximum  $\text{cost}(S)$ , and then choose the larger one. This is the set  $S$  of three points including  $p_i$  with maximum  $\text{cost}(S)$ .



Thus, we have the following theorem.

**Theorem 5.** *Algorithm 1 solves the max-min 3-dispersion problem in  $O(n)$  time when  $P$  is a set of points on a circle and the order of points on the circle is given.*

## 5 Conclusion

In this paper, we have presented some algorithms to solve the max-min dispersion problems. In general, the max-min  $k$ -dispersion problem can be solved in  $O(n^{\omega k/3} \log n)$  time by reducing the problem to the maximum independent set problem. This method implies that the max-min  $k$ -dispersion problem can be solved in  $n^{O(\sqrt{k})}$  time when the input is a set of points in the Euclidean plane.

We then consider two special cases for which the problem can be solved faster. If  $P$  is a set of points on a line and the ordering of the points on the line is given, one can solve the dispersion problem in  $O(n)$  time. If  $P$  is a set of points on a circle and the ordering of the points on the circle is given, one can solve the  $k$ -dispersion problem in  $O(n)$  time.

One open problem is to solve the problem more efficiently if  $P$  is the set of corner vertices of a convex polygon.

## References

1. Agarwal, P., Sharir, M.: Efficient algorithms for geometric optimization. *Comput. Surv.* **30**, 412–458 (1998)
2. Akagi, T., Nakano, S.: Dispersion on the line, IPSJ SIG Technical reports, 2016-AL-158-3 (2016)
3. Baur, C., Fekete, S.P.: Approximation of geometric dispersion problems. In: Jansen, K., Rolim, J. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 63–75. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0053964>
4. Birnbaum, B., Goldman, K.J.: An improved analysis for a greedy remote-clique algorithm using factor-revealing LPs. *Algorithmica* **50**, 42–59 (2009)
5. Cevallos, A., Eisenbrand, F., Zenklusen, R.: Max-sum diversity via convex programming. In: *Proceedings of SoCG 2016*, pp. 26:1–26:14 (2016)
6. Cevallos, A., Eisenbrand, F., Zenklusen, R.: Local search for max-sum diversification. In: *Proceedings of SODA 2017*, pp. 130–142 (2017)
7. Chandra, B., Halldorsson, M.M.: Approximation algorithms for dispersion problems. *J. Algorithms* **38**, 438–465 (2001)
8. Drezner, Z.: *Facility Location: A Survey of Applications and Methods*. Springer, New York (1995)
9. Drezner, Z., Hamacher, H.W.: *Facility Location: Applications and Theory*. Springer, Heidelberg (2004)
10. Erkut, E.: The discrete  $p$ -dispersion problem. *Eur. J. Oper. Res.* **46**, 48–60 (1990)
11. Fekete, S.P., Meijer, H.: Maximum dispersion and geometric maximum weight cliques. *Algorithmica* **38**, 501–511 (2004)
12. Flum, J., Grohe, M.: *Parameterized Complexity Theory*. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-29953-X>

13. Frederickson, G.: Optimal algorithms for tree partitioning. In: Proceedings of SODA 1991, pp. 168–177 (1991)
14. Hassin, R., Rubinstein, S., Tamir, A.: Approximation algorithms for maximum dispersion. *Oper. Res. Lett.* **21**, 133–137 (1997)
15. Kuby, M.J.: Programming models for facility dispersion: the  $p$ -dispersion and maximum dispersion problems. *Geogr. Anal.* **19**, 315–329 (1987)
16. Lei, T.L., Church, R.L.: On the unified dispersion problem: efficient formulations and exact algorithms. *Eur. J. Oper. Res.* **241**, 622–630 (2015)
17. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of ISSAC 2014, pp. 296–303 (2014)
18. Marx, D., Pilipczuk, M.: Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In: Bansal, N., Finocchi, I. (eds.) *ESA 2015*. LNCS, vol. 9294, pp. 865–877. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48350-3\\_72](https://doi.org/10.1007/978-3-662-48350-3_72)
19. Marx, D., Sidiropoulos, A.: The limited blessing of low dimensionality: when  $1 - 1/d$  is the best possible exponent for  $d$ -dimensional geometric problems. In: Proceedings of SoCG 2014, pp. 67–76 (2014)
20. Nešetřil, J., Poljak, S.: On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae* **26**, 415–419 (1985)
21. Ravi, S.S., Rosenkrantz, D.J., Tayi, G.K.: Heuristic and special case algorithms for dispersion problems. *Oper. Res.* **42**, 299–310 (1994)
22. Shier, D.R.: A min-max theorem for  $p$ -center problems on a tree. *Transp. Sci.* **11**, 243–252 (1977)
23. Sydow, M.: Approximation guarantees for max sum and max min facility dispersion with parameterised triangle inequality and applications in result diversification. *Math. Appl.* **42**, 241–257 (2014)
24. Tsai, K.-H., Wang, D.-W.: Optimal algorithms for circle partitioning. In: Jiang, T., Lee, D.T. (eds.) *COCOON 1997*. LNCS, vol. 1276, pp. 304–310. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0045097>
25. Wang, D.W., Kuo, Y.-S.: A study on two geometric location problems. *Inf. Process. Lett.* **28**, 281–286 (1988)



# Non-orthogonal Homothetic Range Partial-Sum Query on Integer Grids [Extended Abstract]

Yuan Tang<sup>1,2(✉)</sup> and Haibin Kan<sup>1</sup>

<sup>1</sup> School of Computer Science, School of Software,  
Shanghai Key Laboratory of Intelligent Information Processing,  
Fudan University, Shanghai, China  
{yuantang,hbkan}@fudan.edu.cn

<sup>2</sup> State Key Laboratory of Computer Architecture, ICT, CAS, Beijing, China

**Abstract.** Algorithms for range partial sum query on high dimensional integer grids typically focus on orthogonal ranges, which by definition demands fixed right triangles between all adjacent boundary edges. We extend the algorithm to solve 2D homothetic triangular range queries in  $\langle O(N\alpha(N)), O(\alpha^2(N)) \rangle$  ( $\langle$ preprocessing bound, query bound $\rangle$  of both time and space since they are identical.), where  $N$  is the total number of grid points and  $\alpha(\cdot)$  is a functional equivalence of the inverse Ackermann function. This asymmetric bound improves over the existing bound for orthogonal ranges. By the property of homotheticity, we mean that the angles between any two adjacent boundaries are arbitrarily fixed constants. The technique and bounds of our work can be extended to even higher dimensional grids.

**Keywords:** Range partial sum query · Triangular reduction  
Non-orthogonal homothetic range

## 1 Introduction

In this paper, we consider the following range partial sum query problem (range query problem for short). Given a static  $d$ -dimensional integer grid  $A$  of dimension  $n_1 \times n_2 \dots \times n_d$  with each grid point<sup>1</sup> holding a value drawn from a semi-group  $(S, \oplus)$ , how fast can we answer online, if preprocessing is allowed, partial sum queries of a simple shape region  $Q$  (i.e. no self-intersection of boundary edges). For a static grid, we disallow dynamic insertion or deletion of any point values.

$$\text{sum}(Q) = \sum_{(k_1, \dots, k_d) \in Q} A(k_1, \dots, k_d). \quad (1)$$

Y. Tang—Supported in part by the Open Funding (No. CARCH201606).

<sup>1</sup> For convenience, we use the term “grid point” and “point” interchangeably in the rest of paper.

Our problem differs from the classic orthogonal range query problem in that orthogonal range by definition demands right triangles ( $90^\circ$ ) between any two adjacent boundary edges, while we allow arbitrarily fixed angles. We only require that all angles are known to the preprocessing algorithm and can not change in the subsequent queries. We use the general term *triangular query* to stand for *triangular shape range partial sum query in a 2D-grid*, *tetrahedral shape range partial sum query in a 3D-grid*, and/or similar extensions to higher dimensional grids. Since  $(S, \oplus)$  is a semigroup,  $\oplus$  operation is an associative operator, i.e.  $(x \oplus y) \oplus z = x \oplus (y \oplus z)$  holds for all  $x, y, z$  in the semigroup. No other restrictions are imposed on this semigroup as is sometimes done by similar problems in the literature. For instances, we do not assume idempotence, i.e.  $a \oplus a = a, \forall a \in S$ , which is required by the Berkman-Vishkin’s algorithm [1]; We do not assume any partial or total ordering among elements in  $S$  as do RMQ (Range Minimum Query) algorithms; We do not assume the existence of an inverse operation of  $\oplus$ , otherwise a trivial algorithm exists [2].

We use the same arithmetic model for complexity analysis as in Yao [2, 3], Chazelle and Rosenberg [4, 5]. In the arithmetic model, the space bound is given in units of semigroup elements instead of bits. The preprocessing and query time are calculated in number of  $\oplus$  operations, ignoring the time to find the proper memory cells<sup>2</sup>. We use the notation  $\langle O(f(N)), O(g(N)) \rangle$  to denote the complexity bounds of a pair of preprocessing and corresponding query algorithm, respectively. Usually, the space and time bounds of these algorithms are identical so that we do not differentiate.

## Motivation

This research was motivated by our study on the Pochoir stencil compiler [8], where we have to query the properties of an arbitrary  $d$ -dimensional octagonal shape region. The octagonal shape comes from the projection of a  $(d + 1)$ -dimensional hyper-zoid<sup>3</sup> onto a  $d$ -dimensional spatial grid. Apparently, a straightforward way to answer an octagonal range query is to decompose it into a set of rectangular and triangular shape queries. The range query is about partial sum because each grid point may contains various properties for the compiler to collect in order to generate an efficient kernel function for the region.

## Our Contributions

1. We extend the range partial sum query problem on integer grid from orthogonal to non-orthogonal shapes. In particular, we solve the 2D triangular problem in  $\langle O(N\alpha(N)), O(\alpha^2(N)) \rangle$  ( $\langle$ preprocessing bound, query bound $\rangle$ ) of either

<sup>2</sup> In the RAM model, we can locate a proper memory cell in a recursive divide-and-conquer tree by finding the Lowest Common Ancestor (LCA) of the end vertices  $i$  of the query range. The LCA problem can in turn be solved by a  $\pm 1$  RMQ algorithm with linear preprocessing time and  $O(1)$  query time [6, 7].

<sup>3</sup> “Hyper-zoid” is a trapezoidal analogue in a  $(d + 1)$ -dimensional grid, where  $d > 1$ . The  $(d + 1)$ -dimensional grid composes of a  $d$ -dimensional spatial grid plus a 1-dimensional temporal axis.

time or space since they are identical.), where  $N$  is the total number of grid points and  $\alpha(\cdot)$  is a functional equivalence of the inverse Ackermann function [9]. This result improves over the previous result on rectangular problem, i.e. the  $\langle N\alpha^2(N), \alpha^2(N) \rangle$  bounds [4, 5].

2. We make the following algorithmic contributions:
  - (a) We generalize the “dimension reduction” technique introduced by Chazelle and Rosenberg [4, 5] to “triangular data reduction” (“triangular reduction” in short) in Sect. 3;
  - (b) We show that an arbitrary triangular problem can be reduced to an Isosceles Right Triangular (IRT) problem, which can be solved similar to the square problem (Sect. 2) based on the observation that it has only one degree of freedom for scaling;
  - (c) We generalize a recursive algorithmic scheme to get an  $\alpha(\cdot)$  bound in the end of Sect. 2.
3. We conjecture that the optimal bounds of homothetic triangular problem in an arbitrary  $d$ -dimensional integer grid, where  $d > 1$  is a constant, is  $\langle O(N\alpha(N)), O(\alpha^d(N)) \rangle$ , in contrast to the  $\langle O(N\alpha^d(N)), O(\alpha^d(N)) \rangle$  bounds of orthogonal problem.

## 2 Square Shape Range Partial Sum Query Problem

This section considers a simpler problem where the query ranges are of square shape, i.e. the height over width must be a fixed 1 : 1 aspect ratio, the study of which will serve as a warm-up for the later homothetic triangular problem to be discussed in Sect. 3. Intuitively, the square problem can be solved more efficiently because a rectangle can scale independently on either dimension, i.e. having two degrees of freedom, while a square has only one in order to keep the 1 : 1 aspect ratio.

**Lemma 1.** *There is an algorithm of  $\langle O(n_1n_2 \log(n_1 + n_2)), O(1) \rangle$  bounds to solve the 2D square range partial sum query problem on an integer grid of dimensions  $N = n_1 \times n_2$ ,*

*Proof.* We prove the lemma by construction as follows. Without loss of generality, we assume that  $n_1 \geq n_2$ . Referring to Fig. 1, the preprocessing algorithm works as follows.

1. We divide the  $n_1 \times n_2$  grid into a  $2 \times 2$  subgrids, each of dimension  $n_1/2 \times n_2/2$ . We store on each point  $p$  inside a subgrid  $g$  four partial sums reduced from all values contained within the maximum rectangle bounded by  $p$  and one of the four corner vertices of  $g$ . We call this procedure **corner reduction**, respectively the partial sums **corner values**. The corner reduction can accomplish in  $O(n_1n_2)$  time and space using dynamic programming.
2. We recursively divide each subgrid into a  $2 \times 2$  array of subgrids, each of dimension  $n_1/2^2 \times n_2/2^2$ , and perform corner reductions for every points within every subsubgrids. This procedure continues until the size of each subgrid reaches  $O(1)$ .

Apparently, the cost of dynamic programming at each level of recursion is  $O(n_1n_2)$ , and there are  $O(\log n_2)$  levels of recursion. The total preprocessing overhead, for either time or space, thus sums up to  $P_{\square,0}(n_1, n_2) = O(n_1n_2 \log n_2)$ .

Given an arbitrary square query range with dimension  $e_{\square} \in [n_2/2^{k+1}, n_2/2^k)$  for some integer  $k \in [0, \log n_2]$ , i.e.  $0 \leq k \leq \log n_2$ , it can stride at most four intersecting subgrids at recursion level  $k$ , i.e. subgrid of dimension  $n_1/2^k \times n_2/2^k$ , because of the fixed 1 : 1 aspect ratio. In other words, there will be exactly one corner vertex of the four intersecting subgrids sitting inside the square query range. If we assume that locating the intersecting corner vertex as well as the four neighboring subgrids is free (as is true in the arithmetic model or the RAM model, see the footnote in Sect. 1), the partial sum of the query range is then a simple summation of the four corner values stored in the vertices.  $Q_{\square,0}(n_1, n_2) = 3 = O(1)$ . We name the procedure **corner query**.  $\square$

**Theorem 1.** *There is an algorithm of  $\langle O(n_1n_2\alpha(n_1 + n_2)), O(\alpha^2(n_1 + n_2)) \rangle$  bounds to solve the 2D square range partial sum query problem on an integer grid of dimension  $N = n_1 \times n_2$ .*

*Proof.* Referring to Figs. 1 and 2, we prove the theorem by constructing a recursive algorithmic scheme as follows. The inputs to the recursive algorithmic scheme are:

1. A square algorithm of  $\langle P_{\square,k}(n_1, n_2) = O(n_1n_2f(n_1 + n_2)), Q_{\square,k}(n_1, n_2) = O(1) \rangle$  bounds, where the subscript  $k$  indicates the  $k$ -th recursive application to the scheme, and  $f(n) < n - 2$  is a function of problem dimension  $n$ .
2. A 1D algorithm of  $\langle P_{-,k}(n) = O(nf(n)), Q_{-,k}(n) = O(1) \rangle$  [2, 3, 9].

For simplicity, we assume that  $n_1 = \Theta(n_2)$  in the following analysis.

The preprocessing algorithm *recursively* partitions the input grid of dimension  $n_1 \times n_2$  into a 2D  $\frac{n_1}{f(n_1+n_2)} \times \frac{n_2}{f(n_1+n_2)}$  array of subgrids, each of dimension  $f(n_1 + n_2) \times f(n_1 + n_2)$ , until the size of each subgrid reaches  $O(1)$ . At each level of recursion, the preprocessing algorithm  $P_{\square,k+1}$  conducts corner reductions, line reductions, and block reductions on subgrids as follows.

1. **Corner reduction:** The algorithm performs corner reductions for every point with respect to the containing subgrid as in Lemma 1. The preprocessing overhead of each subgrid is  $O(f^2(n_1 + n_2))$  and sums up to  $O(n_1n_2)$  over the entire input grid.
2. **Line reduction:** Firstly, the algorithm reduces each horizontal line segment of length  $f(n_1 + n_2)$  within each subgrid into one single value by the  $\oplus$  operation. That is, a horizontal line of length  $n_2$  is reduced to a 1D array of  $n_2/f(n_1 + n_2)$  reduced values. The algorithm then calculates the partial sums of the  $f(n_1 + n_2)$  reduced values (from horizontal line segments) within each subgrid with respect to the top and bottom boundary edges, respectively. That is, for the input grid with  $n_1$  rows of horizontal lines, there will be  $2n_1$  (one  $n_1$  rows comes from the reduction with respect to the top edge, another

$n_1$  rows comes from the reduction with respect to the bottom edge) arrays, each of  $n_2/f(n_1 + n_2)$  **line values**. Thirdly, the algorithm applies the 1D preprocessing algorithm, i.e.  $P_{-,k}$ , to the  $2n_1$  arrays of horizontal line values. Symmetrically, the algorithm applies the line reductions to all vertices line values. The horizontal line reduction takes  $2n_1 \cdot P_{-,k} \left( \frac{n_2}{f(n_1+n_2)} \right) = O(n_1n_2)$  time and space, which is asymptotically the same as the vertical line reduction. The overall line reductions thus sum up to  $O(n_1n_2)$  space and time as well.

3. **Block reduction:** The algorithm reduces all points within each subgrid into a single partial sum by the  $\oplus$  operation and get a 2D array of  $\frac{n_1}{f(n_1+n_2)} \times \frac{n_2}{f(n_1+n_2)}$  **block values**. We then apply the preprocessing algorithm of  $P_{\square,k}$ , i.e. the input square algorithm, to the array of block values. The block reduction will take  $P_{\square,k}(n_1/f(n_1 + n_2), n_2/f(n_1 + n_2)) = O\left(\frac{n_1}{f(n_1+n_2)} \cdot \frac{n_2}{f(n_1+n_2)} \cdot f\left(\frac{n_1}{f(n_1+n_2)}\right)\right) = O(n_1n_2)$  for either time or space.

**Recursive reduction:** We recursively apply the above corner, line, and block reductions to all  $2 \times 2$  array of subgrids of dimension  $2f(n_1 + n_2) \times 2f(n_1 + n_2)$  with subsubgrids of dimension  $f(f(n_1 + n_2)) \times f(f(n_1 + n_2))$ . By a recursive application to all such  $2 \times 2$  array of subgrids, we cover the case when a lower level (with recursion level  $> k$ ) square query range sits between the boundary of two adjacent subgrids based on the fact that it can not stride more than four subgrids of level- $k$ .

The preprocessing overhead of one round of above reductions can be calculated as follows. If we define  $f^{(i)}(n) = n$  if  $i = 0$  and  $f^{(i)}(n) = f(f^{(i-1)}(n))$  if  $i > 0$ , it's not hard to see that the preprocessing overhead of any 2D array of dimension  $2f^{(k)}(n_1 + n_2) \times 2f^{(k)}(n_1 + n_2)$  with subgrids of dimension  $f^{(k+1)}(n_1 + n_2) \times f^{(k+1)}(n_1 + n_2)$  sum up to  $O((f^{(k)}(n_1 + n_2))^2)$ . Since the entire  $n_1 \times n_2$  grid has  $O\left(\left(\frac{n_1n_2}{f^{(k)}(n_1+n_2)}\right)^2\right)$  such 2D arrays, the overall overheads over the entire grid sum up to  $O(n_1n_2)$  for any recursion level. Since the preprocessing proceeds recursively for  $f^*(n_1 + n_2)$  levels<sup>4</sup>, the overall preprocessing overheads, for either time or space, are  $O(n_1n_2f^*(n_1 + n_2))$ .

The query algorithm works as follows. Assuming that the input square query range is of dimension  $e_{\square} \in [f^{(k+1)}(n_1 + n_2), f^{(k)}(n_1 + n_2))$ , i.e. the square is of dimension  $e_{\square} \times e_{\square}$ , the key observation is that it can stride at most one  $2 \times 2$  array of subgrids of dimension  $2f^{(k)}(n_1 + n_2) \times 2f^{(k)}(n_1 + n_2)$ . Since we have preprocessed all such  $2 \times 2$  array of subgrids, we can answer the query at recursion level  $k$  by that specific  $2 \times 2$  array of subgrids. More specifically, it can be answered by an “**inter-block query**”, i.e. a partial sum on the results returned from the following queries: one square query on the data structure preprocessed by “block reduction”, two horizontal line queries (1D queries), two vertical line queries (1D queries) by “line reduction”, and four corner queries by “corner reduction”. Since we assume that we can locate a proper recursion level and the specific  $2 \times 2$  array of subgrids in  $O(1)$  time and the query time of all

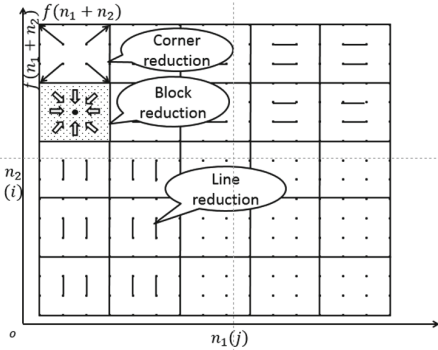
<sup>4</sup> We define  $f^*(n) = 0$  if  $n \leq 1$ , otherwise  $f^*(n) = 1 + f^*(f(n))$ .

input algorithms at any recursion level is  $O(1)$ , the total query overhead sums up to  $O(1)$  as well.

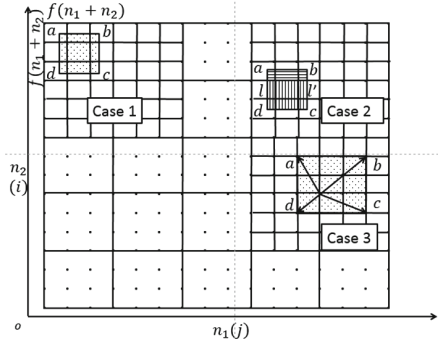
This completes one round of application to the recursive algorithmic scheme for the square problem. The resulting algorithm is of  $\langle O(n_1 n_2 f^*(n_1 + n_2)), O(1) \rangle$  bounds. If we keep supplying the resulting more advanced algorithm into the above recursive algorithmic scheme, we eventually will get the optimal  $\langle O(n_1 n_2$

$\alpha(n_1 + n_2)), O(\alpha^2(n_1 + n_2)) \rangle$  algorithm, where  $\alpha(n) = \min\{k | \log^{* * \dots *}(n) \leq 2\}$  is a functional equivalence of the inverse Ackermann function [9].

We need a bit more explanation on the query bound. The recurrence of query time is  $Q_{\square, k+1}(n_1, n_2) = Q_{\square, k}(n_1/f(n_1 + n_2), n_2/f(n_1 + n_2)) + 4Q_{-, k}(n_2/f(n_1 + n_2)) + 3$  according to the above “inter-block query” procedure. Since the query overhead of 1D algorithm  $Q_{-, \alpha(n)} = O(\alpha(n))$  and the recursive application can continue up to  $\alpha(n_1 + n_2)$  rounds, we have  $Q_{\square, \alpha(n_1 + n_2)}(n_1, n_2) = 4 \sum_{k=0}^{\alpha(n_1 + n_2)} Q_{-, k}(n_1, n_2) + 3\alpha(n_1 + n_2) = O(\alpha^2(n_1 + n_2))$ .  $\square$



**Fig. 1.** This diagram depicts the corner, line, and block reductions.



**Fig. 2.** This diagram demonstrates several possible square shape queries

In summary, our approach, namely the “*recursive algorithmic scheme*”, has the following general framework.

1. We design an initial algorithm  $\mathcal{I}$  that solves the problem correctly, but not necessarily very efficiently.
2. We develop a recursive algorithmic scheme  $\mathcal{M}$ , into which we plug  $\mathcal{I}$  and make the resulting algorithm  $\mathcal{M}(\mathcal{I})$  behave functionally identical to  $\mathcal{I}$ , but with asymptotically different (ideally improved) complexity bound.  $\mathcal{M}$  remains oblivious of the internal structure of  $\mathcal{I}$ , but the knowledge of the asymptotic bounds of  $\mathcal{I}$  may help.
3. We repeat the above two steps for several rounds where in any given iteration  $i > 0$  the input algorithm  $\mathcal{M}^{(i-1)}(\mathcal{I})$  is the resulting algorithm from the  $(i-1)$ -th application of the recursive algorithmic scheme  $\mathcal{M}$ . The goal of repeated self-application is to reach even better bounds (ideally an  $\alpha(\cdot)$  bound).



### 3 Triangular Range Query Problem

This section discusses the 2D triangular problem. We assume that all angles of the triangular shape are fixed constants and known to the preprocessing algorithm. We argue that this is not a weaker version of the classic orthogonal range query problem because an orthogonal range by definition requires fixed right angles between all adjacent boundaries.

**Outline of the high-level idea:**

1. Reducing an arbitrary triangular range query problem to an isosceles right triangular (IRT for short) problem: We have an observation that there is always a smallest *feature triangle* of the query triangular shape, and we can tile up the entire grid by *feature parallelograms* composed of feature triangles and corresponding inverted. A *feature triangle* is a smallest triangle that has the query triangular shape and has all its vertices on the grid points. Similarly, we have the notion of *feature parallelogram*. Referring to Fig. 5, if we reduce every feature triangle and its corresponding feature parallelogram (left-hand side) to two partial sum values ( $\Delta$  and  $\square$  respectively in the figure) and store them as a new grid point, we get a new reduced grid on the right-hand side. By the reduction, an arbitrary triangular range query on the original grid (left-hand side) is equivalent to an IRT query on one of the reduced grids (right-hand side) by aligning its oblique line with one of the tilings of feature parallelograms. To align with all possible input query ranges, we can have up to  $O(|\Delta|)$  different reduced grids, where  $|\Delta|$  is the number of points internal to a feature triangle. In the rest of paper, we use the term “triangle” to stand for IRT unless otherwise specified.
2. Extending the notion of data reduction to triangular prefix/suffix reduction.
3. The key observation is then an input IRT range of dimension  $e_\Delta$  will stride at most one  $3 \times 3$  array of subgrids of dimension  $3(e_\Delta/2) \times 3(e_\Delta/2)$ , where  $e_\Delta/2$  is the dimension of the IRT’s largest embedded square.

**Organization:** We first address how to preprocess all triangular prefix/suffix partial sums in linear time and space by Lemma 2; We then construct an initial algorithm in Theorem 2 and a recursive algorithmic scheme in Theorem 3, respectively.

**Definitions and Conventions throughout the section:**

1. All coordinate systems in the figures of this paper have horizontal axis indexed from left to right (small coordinate on the left and large on the right), and vertical axis from bottom to top (small coordinate on the bottom and large on the top).
2. Referring to Fig. 3, for columns to the left of  $L_v$ , we define the triangular prefix “ $v(i, j).\Delta$  prefix” as the partial sum of the largest IRT bounded between the bottom-left vertex  $(i, j)$  and partition line  $L_v$ , where  $i$  is the horizontal coordinate and  $j$  is the vertical coordinate, respectively.

3. Referring to Fig. 4, for columns to the right of  $L_v$ , we define the triangular suffix “ $v(i, j).\Delta$  suffix” as the partial sum of the largest IRT bounded between the top-right vertex  $(i, j)$  and partition line  $L_v$ .
4. Note that it depends on the orientations of IRT and the partition line whether the reduction to the left, right, upper, lower side of the partition line is for triangular prefix or suffix.
5. We define an IRT’s **core length** ( $e_{\square}$ ) as the dimension of its largest embedded square. We denote an IRT’s dimension, i.e. the horizontal/vertical base, by  $e_{\Delta}$ . Clearly,  $e_{\Delta} = 2e_{\square}$ .

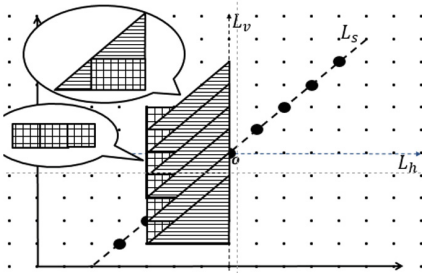


Fig. 3. Triangular prefix

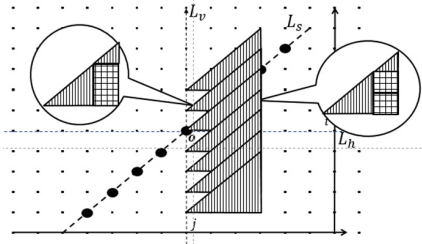


Fig. 4. Triangular suffix

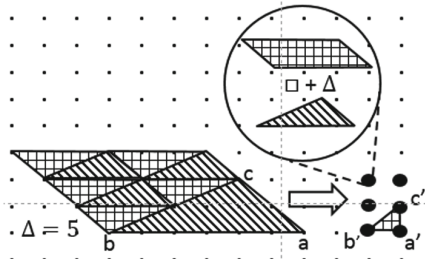


Fig. 5. This diagram shows how to reduce an arbitrary triangular problem to an IRT problem. In this diagram, every grid point on the right-hand side holds two values, i.e. the partial sum of a feature triangle ( $\Delta$ ) and of its feature parallelogram ( $\square$ ).

**Lemma 2.** For an IRT range query on a reduced grid, we can preprocess all triangular prefixes and suffixes with respect to a given partition line in time and space linear to the size of grid.

*Proof.* Referring to Figs. 3 and 4, we assume without loss of generality a vertical partition line  $L_v$ . Initially, each grid point on the reduced grid stores two values, one is the partial sum of a feature triangle ( $\Delta$ ) and the other is the partial sum of the corresponding feature parallelogram ( $\square$ ). We compute the triangular prefixes and suffixes by dynamic programming with respect to  $L_v$  as follows.

1. We can compute all triangular prefixes to the left of  $L_v$  column by column by the recurrences of (2) and (3).

$$v(i, j). \Delta \text{ prefix} = v(i, j). \Delta + v(i + 1, j + 1). \Delta \text{ prefix} + v(i + 1, j). \square \quad (2)$$

$$v(i, j). \square = v(i + 1, j). \square + v(i, j). \square \quad (3)$$

In the recurrences,  $v(i, j). \Delta$  and  $v(i, j). \square$  is the partial sum of feature triangle and corresponding feature parallelogram stored at cell  $(i, j)$ , and  $v(i, j). \square$  denotes the partial sum of one horizontal line segment bounded between coordinate  $(i, j)$  and  $L_v$ . By the dynamic programming recurrences, we can compute all triangular prefixes in time and space linear to the size of grid.

2. We can compute all triangular suffixes to the right of  $L_v$  column by column by the recurrences of (4).

$$v(i, j). \Delta \text{ suffix} = v(i, j). \Delta + v(i - 1, j - 1). \Delta \text{ suffix} + v(i, j - 1). \square \square \quad (4)$$

In the recurrence,  $v(i, j - 1). \square \square$  stands for the partial sum of one vertical line segment bounded between cell  $(i, j - 1)$  and the horizontal base of the largest IRT bounded between  $(i, j)$  and  $L_v$ . In Fig. 4,  $v(i, j - 1). \square \square$  is the vertical line segment shaded by mini-grid.

Observing that the length of  $v(i, j). \square \square$  is always  $i - 1$  for all  $\square \square$  on the same column  $i$ , we denote it by  $d_i = i - 1, i \geq 1$ . We can then compute all  $v(i, j). \square \square$  values column by column by the following simple dynamic programming algorithm.

For the  $i$ -th column to the right of  $L_v$ , we partition it into segments of length  $d_i = i - 1, i \geq 1$ . We then compute by dynamic programming for every point on the column the partial sum from itself to the top and bottom vertices of the holding segment and store them as  $v(i, j). \square$  and  $v(i, j). \sqsupset$ , respectively. Now, it's clear that any  $v(i, j). \square \square$  can be computed in  $O(1)$  time by summing up one  $\square$  and one  $\sqsupset$  value of cell  $(i, j)$  with respect to the top and bottom vertices of the holding segment.

Since the preprocessing time for  $\square \square$  values are amortized  $O(1)$  for every grid point, it's not hard to see that the entire preprocessing overhead of triangular suffixes are linear as well by (4).  $\square$

**Theorem 2.** *There exists an algorithm that can solve the 2D triangular range partial sum query on a grid of dimension  $n_1 \times n_2$  in bounds of  $\langle O(n_1 n_2 \log(n_1 + n_2)), O(1) \rangle$ .*

*Proof.* To simplify the discussion, we assume without loss of generality that  $n_1 \geq n_2$  and  $n_1 = \Theta(n_2)$ . Referring to Figs. 6 and 7, we construct a preprocessing algorithm for 2D triangular range query problem as follows:

1. We divide the  $n_1 \times n_2$  grid into a  $2 \times 2$  array of subgrids, each of dimension  $n_1/2 \times n_2/2$ , ignoring the at most one (1) row or column difference between subgrids to simplify the discussion. More generally, referring to

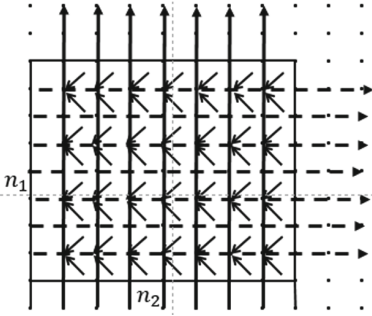


Fig. 6. Perform corner reduction (up to 2 directions) in initial triangular algorithm

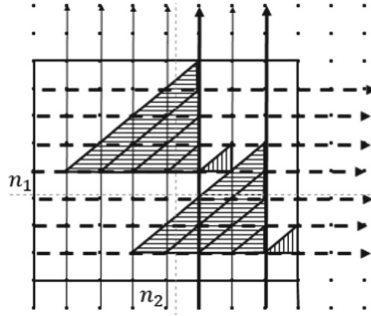


Fig. 7. Perform triangular prefix/suffix (up to 2x) in initial triangular algorithm

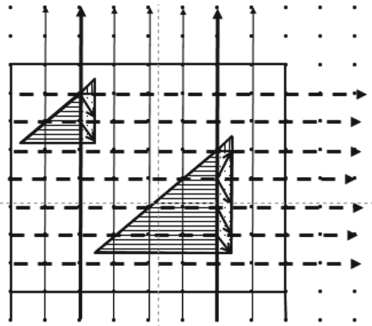


Fig. 8. Query in initial triangular algorithm

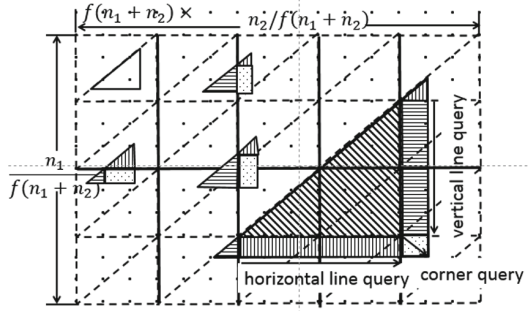


Fig. 9. Query in recursive algorithmic scheme for triangles

Fig. 7, at each recursion level  $k$ , where  $k \in [0, \log n_2]$ , we divide each grid of dimension  $n_1/2^k \times n_2/2^k$  into a  $2 \times 2$  array of subgrids, each of dimension  $n_1/2^{k+1} \times n_2/2^{k+1}$ . Within each subgrid, we perform corner reductions for all points with respect to two vertices, depending on the orientation of the query triangular shape; We also compute triangular prefixes and suffixes for all grid points up to  $2x$  of the subgrid's dimension with respect to every vertical boundaries. The reason behind the  $2x$  triangular prefixes and suffixes computation will become clear in the query algorithm. Since the preprocessing complexity for both corner reduction and triangular prefixes/suffixes are linear, the overall overhead for any recursion level is clearly linear.

- Summing up the overhead over all  $\log n_2$  recursion levels and  $O(|\Delta|)$  different tilings yields the claimed preprocessing bound, where  $|\Delta|$  is a constant for any given query triangular shape.

The query algorithm works as follows. Given an arbitrary input triangular query range, we firstly align its oblique line to one of the  $O(|\Delta|)$  tilings, thus

reduce it to an IRT range query. Analogous to the case of square range query (Lemma 1), we have an observation that an IRT range query with core length  $e_{\square} \in [n_2/2^{k+1}, n_2/2^k)$  can stride at most one  $3 \times 3$  array of subgrids with dimension  $3(n_1/2^k) \times 3(n_2/2^k)$ , thus can be answered by summing up one triangular prefix (up to  $2x$  of the subgrid’s dimension), one triangular suffix, and up to three (3) corner queries as shown in Fig. 8.  $\square$

**Theorem 3.** *There exists a recursive algorithmic scheme that can solve the 2D triangular range partial sum query on a grid of dimension  $n_1 \times n_2$  in bounds of  $\langle O(n_1 n_2 \alpha(n_1 + n_2)), O(\alpha^2(n_1 + n_2)) \rangle$ .*

*Proof.* Referring to Fig. 9, we construct a recursive algorithmic scheme as follows.

We assume an input triangular algorithm with  $\langle P_{\Delta,i}(n_1, n_2) = O(n_1 n_2 f(n_1 + n_2)), Q_{\Delta,i}(n_1, n_2) = O(1) \rangle$  bounds, and an input 1D algorithm with  $\langle P_{-,i}(n) = O(n f(n)), Q_{-,i}(n) = O(1) \rangle$  bounds, where  $f(n) < n - 2$  is a function of the input problem’s dimension and subscript  $i$  specifies the rounds of repeated self-application to the recursive algorithmic scheme as in the case of square problem (proof of Theorem 1). Again, we assume  $n_1 = \Theta(n_2)$  to simplify the discussion.

Analogous to the recursive algorithmic scheme for the square problem (proof of Theorem 1), at recursion level  $k$ , the preprocessing algorithm  $P_{\Delta,i}$  partitions an  $f^{(k)}(n_1 + n_2) \times f^{(k)}(n_1 + n_2)$  grid into a  $\frac{f^{(k)}(n_1 + n_2)}{f^{(k+1)}(n_1 + n_2)} \times \frac{f^{(k)}(n_1 + n_2)}{f^{(k+1)}(n_1 + n_2)}$  array of subgrids, each of dimension  $f^{(k+1)}(n_1 + n_2) \times f^{(k+1)}(n_1 + n_2)$ , and conducts corner reductions, line reductions, triangular prefix/suffix reduction, and block reductions on all subgrids. Recursively, it preprocesses all  $3 \times 3$  subgrids of dimension  $3f^{(k+1)}(n_1 + n_2) \times 3f^{(k+1)}(n_1 + n_2)$  with subsubgrids of dimension  $f^{(k+2)}(n_1 + n_2) \times f^{(k+2)}(n_1 + n_2)$ , and so on. In order to align the oblique line of input (reduced) IRT query range with one of the 2D arrays, we have to preprocess for  $f^{(k+1)}(n_1 + n_2)$  differently aligned 2D arrays of subgrids at level  $k$ . Since the preprocessing overhead for one such array is  $O((f^{(k)}(n_1 + n_2))^2 / f^{(k+1)}(n_1 + n_2))$ , the overhead over all differently aligned arrays is then  $O((f^{(k)}(n_1 + n_2))^2)$ , and sum up to  $O(n_1 n_2)$  over the entire grid.

The query algorithm works as follows. Assuming that the input (reduced) IRT range has core length  $e_{\square} \in [f^{(k+1)}(n_1 + n_2), f^{(k)}(n_1 + n_2))$ , the key observation is that the range can stride at most one  $3 \times 3$  array of subgrids of dimension  $3f^{(k)}(n_1 + n_2) \times 3f^{(k)}(n_1 + n_2)$ . Since we have preprocessed all such  $3 \times 3$  subgrids, we can answer the query by summing up at most the following results, i.e. one triangular query on the data structure preprocessed by “block reduction”, one horizontal and one vertical line queries (1D queries), one corner query, one triangular prefix and one triangular suffix query. The query time is  $O(1)$  following a similar argument as in the proof of Theorem 1.

The claimed final preprocessing and query bounds then follow similarly to that of Theorem 1.  $\square$

### 4 Future Work and Open Problems

We have shown an asymmetric upper bound of  $\langle O(O(N\alpha(N)), O(\alpha^d(N))) \rangle$  for the homothetic triangular range partial sum query problem. The reason behind

asymmetry is that we employ the 1D algorithm [2,9] in our recursive algorithmic scheme. An interesting open problem is whether this bound is tight? A more interesting problem is how to handle general non-orthogonal range partial sum query problem without losing much in complexity bounds, i.e. the angles between adjacent boundaries of input query ranges may change from query to query?

## 5 Related Work

To the best of our knowledge, all previous research on range partial sum query problem assumed orthogonal ranges. Yao [2] devised the first 1D partial sum algorithm with an  $\langle O(N\alpha(N)), O(\alpha(N)) \rangle$  bound. Seidel [9] provided an excellent graphical illustration and simplification of the algorithm. Chazelle and Rosenberg [4] later extended the algorithm to arbitrary  $d$ -dimensional grid with bounds of  $\langle O(N\alpha^d(N)), O(\alpha^d(N)) \rangle$  by the technique of “dimension reduction”. In their later work [5], Chazelle and Rosenberg proved a lower bound of the 1D offline problem, where the queries are known ahead of time. They further conjectured that their multi-dimensional extension is optimal.

Other works in the literature studied variants of the problem, such as the dynamic version that allows insertions and deletions [10], or special cases such as RMQ (Range Minimum Query), where properties such as idempotence and ordering among elements can be utilized [11–18].

Classic orthogonal range searching problem in computational geometry [19] assumes a contiguous space and sparse points, while in our setting we assume an integer grid where every point holds a valid value.

## References

1. Berkman, O., Vishkin, U.: Recursive star-tree parallel data structure. *SIAM J. Comput.* **22**(2), 221–242 (1993)
2. Yao, A.C.: Space-time tradeoff for answering range queries (extended abstract). In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC 1982*, pp. 128–136. ACM, New York (1982)
3. Yao, A.C.C.: On the complexity of maintaining partial sums. *SIAM J. Comput.* **14**(2), 277–288 (1985)
4. Chazelle, B., Rosenberg, B.: Computing partial sums in multidimensional arrays. In: *Proceedings of the Fifth Annual Symposium on Computational Geometry, SCG 1989*, pp. 131–139. ACM, New York (1989)
5. Chazelle, B., Rosenberg, B.: The complexity of computing partial sums off-line. *Int. J. Comput. Geom. Appl.* **1**(1), 33–45 (1991)
6. Demaine, E.D., Landau, G.M., Weimann, O.: On Cartesian trees and range minimum queries. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikolettseas, S., Thomas, W. (eds.) *ICALP 2009*. LNCS, vol. 5555, pp. 341–353. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02927-1\\_29](https://doi.org/10.1007/978-3-642-02927-1_29)
7. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. *J. Algorithms* **57**, 2005 (2005)

8. Tang, Y., Chowdhury, R.A., Kuszmaul, B.C., Luk, C.K., Leiserson, C.E.: The Pochoir stencil compiler. In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, pp. 117–128. ACM, New York (2011)
9. Seidel, R.: Understanding the inverse Ackermann function. In: Invited Talk: 22nd European Workshop on Computational Geometry, Delphi, Greece (2006)
10. Fredman, M.L.: The inherent complexity of dynamic data structures which accommodate range queries. In: FOCS, pp. 191–199 (1980)
11. Amir, A., Fischer, J., Lewenstein, M.: Two-dimensional range minimum queries. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 286–294. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73437-6\\_29](https://doi.org/10.1007/978-3-540-73437-6_29)
12. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.* **40**(2), 465–492 (2011)
13. Fischer, J., Heun, V.: Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 36–48. Springer, Heidelberg (2006). [https://doi.org/10.1007/11780441\\_5](https://doi.org/10.1007/11780441_5)
14. Yuan, H., Atallah, M.J.: Data structures for range minimum queries in multidimensional arrays. In: SODA, pp. 150–160 (2010)
15. Poon, C.K.: Optimal range max datacube for fixed dimensions. *Int. J. Found. Comput. Sci.* **15**(5), 773–790 (2004)
16. Brodal, G.S., Davoodi, P., Lewenstein, M., Raman, R., Srinivasa Rao, S.: Two dimensional range minimum queries and Fibonacci lattices. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 217–228. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-33090-2\\_20](https://doi.org/10.1007/978-3-642-33090-2_20)
17. Brodal, G.S., Davoodi, P., Srinivasa Rao, S.: On space efficient two dimensional range minimum data structures. *Algorithmica* **63**(4), 815–830 (2012)
18. Davoodi, P., Raman, R., Satti, S.R.: Succinct representations of binary trees for range minimum queries. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 396–407. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32241-9\\_34](https://doi.org/10.1007/978-3-642-32241-9_34)
19. Chazelle, B., Edelsbrunner, H.: An optimal algorithm for intersecting line segments in the plane. *J. ACM* **39**(1), 1–54 (1992)



# A Method to Compute the Sparse Graphs for Traveling Salesman Problem Based on Frequency Quadrilaterals

Yong Wang<sup>1(✉)</sup> and Jeffrey Remmel<sup>2</sup>

<sup>1</sup> North China Electric Power University, Beijing 102206, China  
wangyyong100@163.com

<sup>2</sup> University of California, San Diego, CA 92093, USA

**Abstract.** In this paper, an iterative algorithm is designed to compute the sparse graphs for traveling salesman problem (*TSP*) according to the frequency quadrilaterals so that the computation time of the algorithms for *TSP* will be lowered. At each computation cycle, the algorithm first computes the average frequency  $\bar{f}(e)$  of an edge  $e$  with  $N$  frequency quadrilaterals containing  $e$  in the input graph  $G(V, E)$ . Then the  $1/3|E|$  edges with low frequency are eliminated to generate the output graph with a smaller number of edges. The algorithm can be iterated several times and the original optimal Hamiltonian cycle is preserved with a high probability. The experiments demonstrate the algorithm computes the sparse graphs with the  $O(n \log_2 n)$  edges containing the original optimal Hamiltonian cycle for most of the *TSP* instances in the *TSPLIB*. The computation time of the iterative algorithm is  $O(Nn^2)$ .

**Keywords:** Traveling salesman problem · Probability model  
Frequency quadrilateral · Iterative algorithm · Sparse graph

## 1 Introduction

Traveling Salesman Problem (*TSP*) is a well-known *NP*-hard problem in combinatorial optimization. Given the complete graph  $K_n$  on the  $n$  vertices  $\{1, 2, \dots, n\}$ , there is a distance function  $d(u, v) > 0$  between any pairwise vertices  $u, v \in \{1, 2, \dots, n\}$ . For the symmetric *TSP*, we have  $d(u, v) = d(v, u)$ . The objective of *TSP* is to find such a permutation  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$  of the  $n$  vertices  $\sigma_k \in \{1, 2, \dots, n\}$  ( $1 \leq k \leq n$ ) where the total distance  $d(\sigma) = d(\sigma_1, \sigma_n) + \sum_{k=1}^{n-1} d(\sigma_k, \sigma_{k+1})$  is the minimum. Namely, the cycle  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$  with the minimum distance  $d(\sigma)$  is the optimal Hamiltonian cycle (*OHC*) and the other cycles  $\sigma$ s are called the Hamiltonian cycles (*HC*). *TSP* has been proven to be *NP*-complete [1] and there are no exact polynomial-time algorithms unless  $P = NP$ .

*TSP* is the ultimate model of many complex discrete optimization problems, such as the network optimization, VLSI, and machine scheduling, etc. The methods for *TSP* are usually referred to resolve these complicate problems. Thus, *TSP* is extensively studied in combinatorics, operation research and computer science, etc. The algorithms for *TSP* [2] have become one of the prosperous branches in the research. The research



illustrated that the exact algorithms generally need  $O(a^n)$  time to resolve *TSP* where  $a > 1$ . For example, the time complexity of dynamic programming for *TSP* is  $O(n^2 2^n)$  owing to Held and Karp [3], and independently Bellman [4]. The state-of-art branch and bound [5] or cutting plane methods [6, 7] are feasible for *TSP* with thousands of vertices. Due to the exponential number of constraints, the exact methods for *TSP* often need long-time computation to resolve the big scale of *TSP* instances.

Since the computation time of the exact algorithms is hard to reduce, some researchers turn to the approximation algorithms or heuristics for *TSP*. The approximation algorithms mainly depend on the good properties of some special computation models, such as special graphs or trees, to reduce the computation time. The minimum spanning tree-based algorithm [8] and Christofide's algorithm [9] spend the  $O(n^2)$  and  $O(n^3)$  time to produce the 2-approximation and 1.5-approximation for metric *TSP*, respectively. The computation time of the approximation algorithms usually have close relationships with the approximation ratio. The nearer the approximation approaches the optimal solution, the longer the computation time of the approximation algorithms will require [2]. The experiments illustrated that the Lin-Kernighan heuristics (*LKH*) was competitive to generate the approximation [10] within 5% of the optimal solution in nearly  $O(n^{2.2})$  time. One sees the heuristic algorithms are efficient to compute the approximations. On the other hand, they cannot guarantee to find the optimal solution, especially for the large scale of *TSP*.

Besides the above methods for *TSP* on the  $K_n$ , some researchers pursue the methods for *TSP* on the sparse graphs. The sparse graphs include a small number of edges so that the search space of the *OHC* is greatly reduced. For example, the *TSP* on the bounded degree graphs can be resolved in time  $O((2 - \varepsilon)^n)$  [11] where  $\varepsilon$  relies on the maximum degree of a vertex. In the research of approximation algorithm, Gharan and Saberi [12] proposed the polynomial-time approximation schemes based upon the bounded genus graphs where the constant factor is  $22.51(1 + \frac{1}{n})$  for the planar *TSP*. For the metric *TSP* with bounded intrinsic dimension, Bartal et al. [13] have designed the randomized polynomial-time algorithm that is able to compute a  $(1 + \varepsilon)$ -approximation to the optimal solution where  $\varepsilon > 0$ . For the *TSP* on the  $K_n$ , there are no such good results. Whether one pursues the optimal solution or explores the approximations for the *TSP*, he or she will get the better results on the sparse graphs than those on the complete graph  $K_n$ . The question is how to reduce the *TSP* on the  $K_n$  to the *TSP* on the sparse graphs.

This topic is the research of this paper. Given the *TSP* on the  $K_n$ , our objective is to eliminate the number of the irrelevant edges as most as possible and lose the number of the *OHC* edges as least as possible. After many irrelevant edges are trimmed, the sparse graphs will be obtained. If the sparse graph contains the original *OHC*, the exact or approximation algorithms will consume less time to find the *OHC* or approximations in the sparse graphs. Otherwise, if the sparse lose a few *OHC* edges, the exact or approximation algorithms will find the approximations in the sparse graphs.

To eliminate the edges out of the *OHC*, the difference between the *OHC* edges and the other edges will be explored. According to the *k-opt moves*, Hougardy and Schroeder [14] proved some edges cannot belong to the *OHC*. They presented a three-stage combinatorial algorithm to trim a lot of irrelevant edges. The experiments

showed the Concorde *TSP* solver was speeded up 11 times to resolve the Euclidean *TSP* instance *d2103* on the sparse graph.

Our work computes a sparse graph for *TSP* according to the frequency graph. In previous work [15–17], we compute the frequency graph with a set of the optimal 4-vertex paths with given endpoints where the frequency of the *OHC* edges is much higher than the average frequency of all the edges. The results benefit from the good property of these specific optimal paths which have many intersections of edges with the *OHC*. Since the frequency of the *OHC* edges are much bigger than that of most of the other edges, the minimum frequency of the *OHC* edge can be taken as the frequency threshold to eliminate the other edges with low frequency so that it is possible to compute a sparse graph for *TSP*.

In a following paper [18], Wang and Rimmel computed the frequency graphs with the frequency quadrilaterals rather than the specific optimal 4-vertex paths. They listed the six frequency quadrilaterals for a weighted quadrilateral *ABCD* in the  $K_n$ . Based on the six frequency quadrilaterals, they formulated a binomial distribution model to derive the lower bound of the frequency of the *OHC* edge  $e$  as  $\lfloor (\frac{7}{3} + \frac{4}{3(n-3)})N \rfloor$  where  $N$  represents the number of the frequency quadrilaterals containing  $e$ . The probability that an *OHC* edge has the minimum frequency  $\lfloor (\frac{7}{3} + \frac{4}{3(n-3)})N \rfloor$  tends to zero for big  $N$  and  $n$ . In average case, the event that an *OHC* edge  $e$  has the frequency above  $\lfloor (3 + \frac{2}{n-2})N \rfloor$  has the maximum probability. It means that the frequency of the *OHC* edges will be bigger than  $\lfloor (3 + \frac{2}{n-2})N \rfloor$  in most cases. The experiments showed that the actual minimum frequency of the *OHC* edge was bigger than  $\lfloor (3 + \frac{2}{n-2})N \rfloor$  for most *TSP* instances. Moreover, the minimum frequency of the *OHC* edges increases according to  $n$ . Therefore, it is feasible to compute a residual graph using the frequency  $3N$  as a frequency threshold.

Given the  $K_n$ , we first compute the corresponding frequency graph with the frequency quadrilaterals in  $K_n$ . After we eliminate the edges with low frequency according to the minimum frequency of the *OHC* edge, we will obtain the first preserved graph  $G_1$  containing the *OHC*. A natural idea is that we can repeat the procedure for the edges in the  $G_1$  if the edges in the  $G_1$  are included in many quadrilaterals. That is, we compute the frequency graph of  $G_1$  and trim the edges with lower frequency according to the other minimum frequency of the *OHC* edge. Furthermore, if the preconditions in the preserved graphs are sufficient, this procedure can be iterated several times until the final preserved graph is sparse enough. Based on the sparse graphs, the computation time of the algorithms for *TSP* will be greatly reduced. Once the sparse graph has the good properties, such as planarity,  $k$ -edge connected, bounded degree, bounded tree-width or genus, etc., the complexity of *TSP* will be lowered. A first question is how many possible edges we should eliminate at each computation cycle according to the minimum frequency of the *OHC* edge? The second question is how many cycles we can run the procedure to guarantee the preserved graphs to contain the *OHC*. Since the answers to the first question only concerns the number of the deleted edges at each computation cycle, the stop computation cycle must be given to terminate the computation procedure to output the sparse graph for *TSP*.

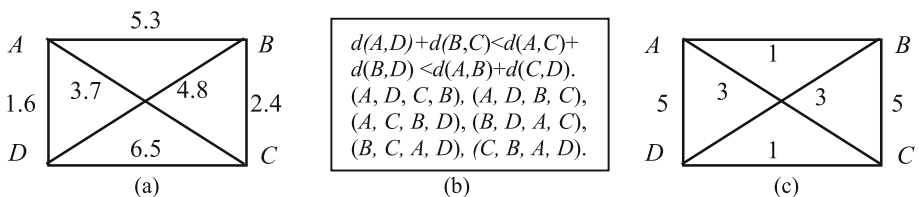
The outline of this paper is given as follows. In Sect. 2, we shall briefly introduce the frequency quadrilaterals and the probability model for the *OHC* edges. A criterion is derived to eliminate how many possible edges whereas the *OHC* edges are kept intact. In Sect. 3, we shall introduce the iterative algorithm to compute a sparse graph based on the frequency quadrilaterals. The maximum computation cycle and the stop computation cycle are also given. The iterative algorithm is tested with tens of real-world *TSP* examples in Sect. 4. The preserved graphs in the computation process will be shown. The conclusions and possible future research are given in the last section.

## 2 The Frequency Quadrilaterals

The frequency quadrilateral is a kind of special frequency graph  $K_i$  where  $i = 4$  [18]. The frequency quadrilateral is computed with the six optimal 4-vertex paths in one corresponding quadrilateral. Here we only consider the frequency quadrilaterals derived from the general weighted quadrilaterals  $K_4$ . Each weighted quadrilateral just includes 6 optimal 4-vertex paths ( $OP^4$ ) and one *OHC*. The  $OP^4$ s with given endpoints in a quadrilateral  $ABCD$  is computed as follows.

Given a quadrilateral  $ABCD$  in  $K_n$ , it includes six edges  $(A, B)$ ,  $(A, C)$ ,  $(A, D)$ ,  $(B, C)$ ,  $(B, D)$  and  $(C, D)$ . The distances of the six edges are  $d(A, B)$ ,  $d(A, C)$ ,  $d(A, D)$ ,  $d(B, C)$ ,  $d(B, D)$  and  $d(C, D)$ , respectively. Appoint two endpoints, such as  $A$  and  $B$ , there are two 4-vertex paths  $P_1 = (A, C, D, B)$  and  $P_2 = (A, D, C, B)$  containing the four vertices  $A, B, C$  and  $D$ . Their distances are computed as  $d(P_1) = d(A, C) + d(C, D) + d(B, D)$  and  $d(P_2) = d(A, D) + d(C, D) + d(B, C)$ . We assume the distances of the two paths are unequal, i.e.,  $d(P_1) \neq d(P_2)$ . One path must be shorter than the other one. We take the shorter path  $P_1$  or  $P_2$  as the  $OP^4$  for the two end vertices  $A$  and  $B$ . Since we have six pairs of endpoints according to the four vertices  $A, B, C$  and  $D$ , there are six  $OP^4$ s in the quadrilateral  $ABCD$ . According to the distances of edges, the 6  $OP^4$ s are computed with the four-vertex and three-line inequality array [19].

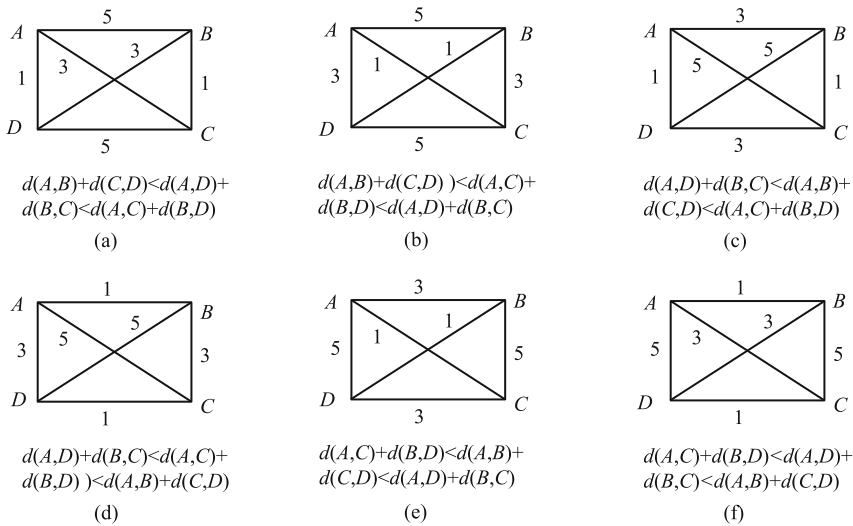
For example in Fig. 1, Fig. 1(a) is the quadrilateral  $ABCD$  where the *OHC* =  $(A, C, B, D, A)$ . Figure 1(b) is the inequality array  $d(A, D) + d(B, C) < d(A, C) + d(B, D) < d(A, B) + d(C, D)$  according to the distances of edges and the six  $OP^4$ s derived based on the inequality array. Figure 1(c) is the frequency quadrilateral  $ABCD$  computed with the 6  $OP^4$ s. It is clear that the 4 edges  $(A, D)$ ,  $(A, C)$ ,  $(B, C)$  and  $(B, D)$  with the top frequencies 5, 5, 3 and 3 belong to the *OHC* =  $(A, C, B, D, A)$ .



**Fig. 1.** The quadrilateral  $ABCD$  (a), the inequality array and the six  $OP^4$ s (b), and the frequency quadrilateral  $ABCD$  (c)

In a frequency quadrilateral  $ABCD$ , the pairwise non-adjacent edges have the same frequency 5, 3 or 1. For example in Fig. 1, the edges  $(A, D)$  &  $(B, C)$  have the frequency 5, the edges  $(A, C)$  &  $(B, D)$  have the frequency 3, and the edges  $(A, B)$  &  $(C, D)$  have the frequency 1, respectively. It is the distances of the pairwise non-adjacent edges that conclude their frequencies. The bigger the summed distance of the two non-adjacent edges is, the smaller their frequency will be. For example in Fig. 1, the summed distances of the three pairs of non-adjacent edges  $(A, D)$  &  $(B, C)$ ,  $(A, C)$  &  $(B, D)$  and  $(A, B)$  &  $(C, D)$  are 4.0, 8.5 and 11.8, respectively. However, their frequencies are 5, 3 and 1, respectively.

The distances of the edges in a quadrilateral  $ABCD$  are various. According to the three summed distances of the three pairs of non-adjacent edges, they will produce six inequality arrays. Each inequality array determines a set of six  $OP^4$ s and a corresponding frequency quadrilateral. Thus, six distinct frequency quadrilaterals  $ABCD$  are computed and shown in Fig. 2(a)–(f) [18]. The summed distance array of the quadrilateral  $ABCD$  is listed below the corresponding frequency quadrilateral  $ABCD$ .



**Fig. 2.** The six frequency quadrilaterals  $ABCD$  for a quadrilateral  $ABCD$

Although there are a number of weighted quadrilaterals, they are classified into six kinds according to the six frequency quadrilaterals in Fig. 2. For each kind of quadrilaterals  $ABCD$ , the distances of the six edges conform to the same inequality array. In addition, their  $OHC$  is determined by the corresponding inequality array. Let's analyze the six frequency quadrilaterals for a quadrilateral  $ABCD$ . Firstly, the frequency of the six edges in each frequency quadrilateral is  $f = 5, 3$  and  $1$ . The frequency of edges in the frequency quadrilaterals composes a stable frequency space  $\{1, 3, 5\}$ . For the three adjacent edges containing a vertex, such as  $AB, AC$  and  $AD$ , they have the different frequency. Therefore, the difference between adjacent edges can be clearly

characterized by their frequency. In addition, the two adjacent edges containing a vertex with the big frequency 5 and 3 are included in the *OHC* whereas another adjacent edge with the frequency 1 is not. Next, the two non-adjacent edges have the equal frequency. If one edge belongs to the *OHC*, the opposite edge must be in the *OHC* and vice versa. If we find two adjacent *OHC* edges in a quadrilateral, the other two *OHC* edges are also concluded.

Secondly, we see the frequency of an edge in the six frequency quadrilaterals. For an edge  $e \in \{(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)\}$ , it has the frequency  $f = 5, 3$  and 1 twice in the six frequency quadrilaterals, respectively. If the frequency  $f \in \{5, 3, 1\}$  of  $e$  is taken as a random variable, we see the probability  $p$  that  $e$  has the frequency  $f = 5, 3$  and 1 is equal to  $1/3$  based on the six frequency quadrilaterals. We note the probability  $p(f = 5) = p(f = 3) = p(f = 1) = 1/3$  for  $e$  whose frequency  $f = 5, 3$  and 1 in a random frequency quadrilateral containing the  $e$ .

Given a *TSP* with  $n$  vertices, there are  $\binom{n}{4}$  weighted quadrilaterals. Each quadrilateral includes six edges. Thus, every edge is included in  $\binom{n-2}{2}$  quadrilaterals. For an edge  $e$  in each of these corresponding frequency quadrilaterals, the possible frequency of  $e$  may be 5, 3 or 1. Given a random frequency quadrilateral containing  $e$ , we assume the frequency quadrilateral has the equal probability to be one of the six frequency quadrilaterals in Fig. 2. Thus, the probability  $p(f = 5) = p(f = 3) = p(f = 1) = 1/3$  that  $e$  has the frequency 5, 3 and 1 in the frequency quadrilaterals. When we compute the frequency  $F(e)$  of  $e$  with  $N$  frequency quadrilaterals containing  $e$ , the frequency  $F(e) = (5p(f = 5) + 3p(f = 3) + p(f = 1))N = 3N$ . It is the average frequency of all edges.

For the *OHC* edges, there are some special frequency quadrilaterals where their frequency  $f = 5, 3$  rather than 1. In the  $K_n$ , each two adjacent *OHC* edges are included in  $n - 3$  quadrilaterals and each two opposite *OHC* edges are contained in a quadrilateral. In the corresponding frequency quadrilateral containing two opposite *OHC* edges, the frequency of the two *OHC* edges are 5 or 3. Otherwise, the two opposite *OHC* edges will be replaced by the other two non-adjacent edges in the quadrilateral. According to the observations, Wang and Rimmel [18] constructed the  $n - 3$  frequency quadrilaterals for an *OHC* edge  $e$  where its frequency is 3 or 5. In the rest frequency quadrilaterals, they assume the frequency 1, 3 and 5 of  $e$  has the equal probability  $1/3$ . Thus, they gave the probability that  $e \in OHC$  has the frequency 1, 3 and 5 in a frequency quadrilateral as  $p(f = 5) = p(f = 3) = \frac{1}{3} + \frac{1}{3(n-2)}$  and  $p(f = 1) = \frac{1}{3} - \frac{2}{3(n-2)}$ . When we compute the frequency of  $e \in OHC$  with  $N$  random frequency quadrilaterals containing  $e$ , the expected frequency  $F(e) = 3N + \frac{2N}{n-2}$ . One sees the expected frequency of an edge  $e \in OHC$  computed with  $N$  random frequency quadrilaterals is bigger than the expected frequency  $3N$  of a common edge.

According to the probability  $p(f = 5) = p(f = 3) = \frac{1}{3} + \frac{1}{3(n-2)}$  and  $p(f = 1) = \frac{1}{3} - \frac{2}{3(n-2)}$ , we know that the probability  $p(f = 3, 5) > \frac{2}{3}$  for an *OHC* edge  $e$  in a frequency quadrilateral in  $K_n$ . When we compute the frequency of an edge with  $N$  frequency quadrilaterals containing  $e$ , the total frequency of an *OHC* edge will be

bigger than  $3N$ . Certainly, the average frequency  $\bar{f}(e)$  of the *OHC* edge  $e$  will be bigger than 3. Therefore, it is necessary to consider the edges  $e$  with the frequency  $F(e) > 3N$  or  $\bar{f}(e) > 3$  for *TSP*. We are interested in how many edges with the frequency  $F(e) > 3N$  or  $\bar{f}(e) > 3$  when each of them is computed with  $N$  frequency quadrilaterals.

Given an edge  $e$  in the six frequency quadrilaterals in Fig. 2, there are four frequency quadrilaterals where  $e$  has the frequency 3 and 5. Thus, the probability  $p(f \geq 3)$  of the case  $f \geq 3$  for an edge  $e$  is equal to  $2/3$ , i.e.,  $p(f \geq 3) = 2/3$ . It says the  $e$  has the probability  $2/3$  that its frequency is bigger than the expected frequency 3 in a random frequency quadrilateral. When we choose  $N$  random frequency quadrilaterals containing  $e$ , there will be  $\lfloor \frac{2N}{3} \rfloor$  frequency quadrilaterals where the frequency is above 3 and  $\lfloor \frac{N}{3} \rfloor$  frequency quadrilaterals where the frequency is below 3. If we take 3 as the frequency threshold to eliminate the edges with smaller frequency, the edge  $e$  will be preserved  $\lfloor \frac{2N}{3} \rfloor$  times. Thus, the probability that  $e$  is preserved is  $\frac{2}{3}$  when we take  $f = 3$  as the frequency threshold to eliminate the edges with lower frequency. In this case, the total frequency  $F(e) = 3N$  and average frequency  $\bar{f}(e) = 3$ . If we take the total frequency  $F = 3N$  or average frequency  $\bar{f} = 3$  as the frequency threshold, the edges  $e$  with the probability  $p(f \geq 3) \geq 2/3$  in each frequency quadrilateral will be preserved.

Considering the  $\binom{n}{2}$  edges in the  $K_n$ , we will preserve at most  $\frac{2}{3} \binom{n}{2}$  edges with the big frequency for *TSP*. Since the *OHC* edge  $e$  has the big frequency  $F(e) > 3N$  or  $\bar{f}(e) > 3$ , the *OHC* edges will be preserved with a big probability. In fact, an *OHC* edge  $e$  has the probability  $p(f \geq 3) \geq 2/3$  in each frequency quadrilateral so it is preserved.

It mentions that we will preserve some edges with the probability  $p(f \geq 3) \geq 2/3$  when we take  $F = 3N$  or  $\bar{f} = 3$  as the frequency threshold. For example, the edges have the probability  $p(f = 5) > 1/2$  and  $p(f = 3) + p(f = 5) < 2/3$ . Thus, more irrelevant edges with the big frequency  $F(e) > 3N$  and average frequency  $\bar{f}(e) > 3$  will be preserved. If the edges in the preserved graph are contained in many frequency quadrilaterals, it is necessary to iterate the computation process to eliminate these edges in the next computation cycles. In the following section, we will give the iterative algorithm to trim these edges step by step until a sparse graph for *TSP* is computed.

### 3 The Iterative Algorithm

We first give the iterative algorithm and then discuss the stop computation cycle. Given an original graph  $G_0(V_0, E_0)$ ,  $|V_0| = n$  and  $|E_0| = \binom{n}{2}$  in general. When the frequency of every edge is computed with  $N$  frequency quadrilaterals in  $G_0(V_0, E_0)$ , we choose  $\lfloor \frac{2}{3}|E_0| \rfloor$  edges with top frequency to compose a second graph  $G_1(V_0, E_1)$  for *TSP* according to the probability  $p(f = 5) = p(f = 3) = \frac{1}{3} + \frac{1}{3(n-2)}$  and  $p(f = 1) = \frac{1}{3} - \frac{2}{3(n-2)}$ . After that, if  $N$  is big for the edges in  $G_1(V_0, E_1)$ , we can use the same method to compute a third graph  $G_2(V_0, E_2)$  with an even smaller number of edges where  $|E_2| = \lfloor (\frac{2}{3})^2 |E_0| \rfloor$ . Furthermore, if  $N$  is still big for the edges in  $G_k(V_0, E_k)$  where  $k > 2$ ,

we can keep executing the computation until a sparse graph containing  $\left[\left(\frac{2}{3}\right)^k |E_0|\right]$  edges is generated. We expect the final sparse graph to include  $O(n \log_2 n)$  edges so that the better polynomial-time algorithms or polynomial-time approximate schemes are designed for *TSP* based on these sparse graphs. It notes that  $N$  should be sufficiently big for the edges in  $G_k(V_0, E_k)$  at each computation cycle where  $k \geq 0$ . Otherwise, the probability  $p_5(e)$  and  $p_3(e)$  of the *OHC* edges will be much smaller than the  $p(f = 5) = p(f = 3) = \frac{1}{3} + \frac{1}{3(n-2)}$  and smaller than that of the other edges. One will wonder the probability  $p(f = 5)$ ,  $p(f = 3)$  of the *OHC* edges will become smaller in the preserved graphs and they will be eliminated in the computation process. We will explore the change of the probability  $p(f = 5)$ ,  $p(f = 3)$  and  $p(f = 1)$  for the *OHC* edges in the computation process in another paper. Here we will focus on the algorithm to compute the sparse graphs for *TSP*. In the next section, we will see the *OHC* edges usually have the bigger average frequency than that of the  $\frac{1}{3} |E_k|$  edges to be eliminated in the preserved graphs for the *TSP* instances. Under the assumption, we give the iterative algorithm in Table 1.

**Table 1.** The iterative algorithm to compute a sparse graph for *TSP*

Step	The pseudo codes of the iterative algorithm
1	Input the initial graph $G_k(V_0, E_k)$ where $k := 0$ , $ V_0  = n$ and $ E_0  = \binom{n}{2}$ .
2	Do {
3	For each edge $e \in E_k$ { Compute the average frequency $\bar{f}(e)$ of $e$ with $N$ frequency quadrilaterals containing $e$ . }
4	Order the $ E_k $ edges according to their $\bar{f}(e)$ s from big to small values.
5	Preserve the previous $\left\lceil \frac{2}{3}  E_k  \right\rceil$ edges to the next graph $G_{k+1}(V_0, E_{k+1})$ .
6	Replace $G_k(V_0, E_k)$ with $G_{k+1}(V_0, E_{k+1})$ and assign $k := k + 1$ .
7	} While $( E_{k+1}  \geq cn)$ .
8	Output the sparse graph with $cn$ edges.

The first step is to input an initial weighted graph  $G_0(V_0, E_0)$  with  $n$  vertices and  $|E_0|$  edges. Generally,  $|V_0| = n$  and  $|E_0| = \binom{n}{2}$ . Assign the initial value of computation cycle  $k = 0$ . In the following steps, the iterative algorithm generates a sparse graph with less than  $cn$  edges where  $c \approx \lceil \log_2 n \rceil$ . At the  $k^{th}$  computation cycle where  $k \geq 0$ , the algorithm begins with a input graph  $G_k(V_0, E_k)$  and outputs the next preserved graph  $G_{k+1}(V_0, E_{k+1})$  with  $|E_{k+1}| = \left[\left(\frac{2}{3}\right)^{k+1} |E_0|\right]$  edges according to the average frequency of edges in the  $G_k(V_0, E_k)$ . We use the average frequency of edges instead of their total frequency to avoid bias to some edges since the edges will be contained in different number of frequency quadrilaterals in the in  $G_k(V_0, E_k)$ . The average frequency  $\bar{f}(e)$  of every edge  $e$  in  $G_k(V_0, E_k)$  is computed as follows. Firstly, the  $N$  quadrilaterals containing the edge  $e$  are chosen and the frequency  $f(e)$  of  $e$  is enumerated from the  $6 OP^4$ s



in each of the quadrilaterals. Given the frequency of  $e$  is  $f_j(1 \leq j \leq N)$  in the  $j^{th}$  frequency quadrilateral, the average frequency of  $e$  is computed as  $\bar{f}(e) = \frac{1}{N} \sum_{j=1}^N f_j$ . In the iterative computation process, the number of the frequency quadrilaterals containing every edge in  $G_k(V_0, E_k)$  will not be equal. It is fair to compare the average frequency of edges rather than their total frequency in  $G_k(V_0, E_k)$ . Therefore, it is rational to keep the  $\lfloor \frac{2}{3} |E_k| \rfloor$  edges with the big average frequency for *TSP*.

After the average frequency  $\bar{f}(e)$ s of the  $|E_k|$  edges are computed, we order them from big to small values to form a frequency sequence at step 4. Given the average frequency of the  $j^{th}$  edge is  $\bar{f}_j(1 \leq j \leq |E_k|)$ , we note the frequency sequence as  $(\bar{f}_1, \bar{f}_2, \dots, \bar{f}_{|E_k|})$  where  $\bar{f}_1$  is the maximum frequency and  $\bar{f}_{|E_k|}$  is the minimum frequency. At the  $5^{th}$  step, the previous  $\lfloor \frac{2}{3} |E_k| \rfloor$  edges are chosen to compose the next graph  $G_{k+1}(V_0, E_{k+1})$ . To continue the recursive computations, we replace the graph  $G_k(V_0, E_k)$  with the graph  $G_{k+1}(V_0, E_{k+1})$  and assign  $k := k + 1$ . That is, the edge set  $E_k$  in the  $G_k$  is substituted with the edge set  $E_{k+1}$  in  $G_{k+1}$ . The iterative algorithms will always be executed until the terminal condition  $|E_{k+1}| \leq cn$  is met. At last, it outputs the final sparse graph with less than  $cn$  edges.

When we run the iterative algorithm based on the frequency quadrilaterals, the preserved graphs will have the smaller and smaller number of edges according to the computation cycle  $k$ . Given through  $k$  iterations, we will obtain a sparse graph with  $cn$  edges. The maximum iterations  $k_{max}$  is given as formula (1). Many researchers take the graphs with  $O(n \log_2 n)$  edges as the sparse graphs. The number  $O(n \log_2 n)$  increases nearly in a linear way according to  $n$ . Thus, we take  $c = \log_2 n$  for general *TSP* instances. The maximum computation cycle becomes  $k_{max} = \left\lceil \log_{\frac{2}{3}} \left( \frac{2 \log_2 n}{n-1} \right) \right\rceil$ . At the  $k_{max}^{th}$  computation cycle, the iterative algorithm will output a sparse graph with  $\lfloor n \log_2 n \rfloor$  edges for general *TSP*.

$$k_{max} = \left\lceil \log_{\frac{2}{3}} \left( \frac{2c}{n-1} \right) \right\rceil \tag{1}$$

If every  $K_4$  includes only one *OHC* and the six  $OP^4$ s, the average frequency of the *OHC* edges will be bigger than the expected frequency 3 whereas the average frequency some of the other edges will be below 3. In this case, we can eliminate  $\frac{1}{3}$  edges with small average frequency. This is the theoretical case. For real-world instances, some  $K_4$ s in the  $K_n$  include more than six  $OP^4$ s as they contain the equal-weight edges. The selection of the right 6  $OP^4$ s becomes hard to compute a unique frequency quadrilateral. If the wrong  $OP^4$ s in these  $K_4$ s are used, the average frequency of some *OHC* edges in the  $K_n$  will become smaller. These *OHC* edges will be eliminated with a big probability. For general *TSP*, the iterative algorithm will work well to compute a sparse graph for *TSP*. Even though for the special *TSP* examples, the experiments showed that the iterative algorithm still works if we add the random small distances to the edges' distances in advance [18].



We are interested in how many *OHC* edges will be lost in the computation process. At the  $(k + 1)^{th}$  computation cycle, we will maintain  $\lfloor \frac{2}{3} |E_k| \rfloor$  edges in the preserved graph. In other words, we will throw away  $\lfloor \frac{1}{3} |E_k| \rfloor$  edges according to the average frequency of edges. In the graph  $G_k(V_0, E_k)$ , the probability that an edge  $e$  is abandoned is  $p(e \notin E_{k+1}) = 1/3$ . The *OHC* includes  $n$  edges. If we do not consider the frequency of edges, the probability that an edge  $e \in OHC$  in  $G_k(V_0, E_k)$  is  $p(e \in OHC) = \frac{n}{|E_k|}$  where  $|E_0| = \binom{n}{2}$ . At the  $k^{th}$  ( $k \geq 1$ ) computation cycle, the number of the edges in the input graph  $G_{k-1}(V_0, E_{k-1})$  is  $\lfloor (\frac{2}{3})^{k-1} |E_0| \rfloor$ . We assume the graph  $G_{k-1}(V_0, E_{k-1})$  includes the *OHC* so that the probability of an edge  $e \in OHC$  is  $p(e \in OHC) = \frac{n}{(\frac{2}{3})^{k-1} |E_0|}$ . Every edge has the probability  $1/3$  to be abandoned at the  $k^{th}$  computation cycle. Thus, the probability that the edge  $e \in OHC$  is abandoned in the next graph  $G_k(V_0, E_k)$  is computed as formula (2).

$$p(e \in OHC \wedge e \notin E_k) = \frac{1}{3} \times \frac{n}{(\frac{2}{3})^{k-1} |E_0|} \tag{2}$$

If one or more *OHC* edges are lost at the  $k^{th}$  computation cycle, we have  $np(e \in OHC \wedge e \notin E_k) \geq 1$ . Therefore, the formula (3) holds if  $|E_0| = \binom{n}{2}$ .

$$k \geq 2 + \left\lceil \log_{\frac{2}{3}} \left( \frac{n}{n-1} \right) \right\rceil \tag{3}$$

As  $n \rightarrow \infty, k \geq 2$ . It means the iterative algorithm can be executed at least 2 times without losing the *OHC* edges. If  $m$  *OHC* edges are lost where  $m$  is a small constant, we can derive the computation cycles  $k \geq 2 + \left\lceil \log_{\frac{2}{3}} \left( \frac{1}{m} \right) \right\rceil$ . The computation cycle becomes bigger as  $m$  rises. It means that we can run the algorithm more times to compute a sparse graph on condition that only a small number of the *OHC* edges are lost. For example, if  $m = \frac{9}{4}$ , we can run the algorithm at least  $k = 4$  times. However, we may lose only  $m < 3$  *OHC* edges.

In fact, the formula (2) is the average probability for an arbitrary edge  $e$  in any given  $n$  edges in  $G_{k-1}(V_0, E_{k-1})$ . In our algorithm, we maintain the edges according to the frequency of edges rather than the random selection. For each of the edges in the *OHC*, their average frequency computed with the  $N$  frequency quadrilaterals in  $G_{k-1}(V_0, E_{k-1})$  will be bigger than the average frequency of all of the edges. In the frequency sequence  $(\bar{f}_1, \bar{f}_2, \dots, \bar{f}_{\lfloor \frac{1}{3} |E_{k-1}| \rfloor})$ , if the number of edges with the average frequency below the 3 is bigger than  $\lfloor \frac{1}{3} |E_{k-1}| \rfloor$ , the probability that the *OHC* edges will be maintained in  $G_k(V_0, E_k)$  tends to 1 but not  $2/3$ . Moreover, the probability that an *OHC* edge is neglected in  $G_k(V_0, E_k)$  is small than  $1/3$  based on the frequency quadrilaterals. Therefore, the edges in the *OHC* have a much bigger probability that they will be maintained to the  $G_k(V_0, E_k)$  as  $N$  is big enough. According to the average frequency of

edges, the probability  $p(e \in OHC \wedge e \notin E_k)$  will be much smaller than that computed with the formula (2) based on the random selection. Thus, we can run the iterative algorithm more times than that restricted by formula (3) for general *TSP*. Meanwhile, we will obtain an even sparser graph containing the *OHC* for *TSP*.

Many incomplete quadrilaterals will appear in the computation process because the  $\lfloor \frac{2}{3}|E_{k-1}| \rfloor$  edges with small frequency are abandoned at the  $k^{th}$  computation cycle. These incomplete quadrilaterals contain less than 6 edges as well as less than 6  $OP^4$ s. If these incomplete quadrilaterals are used to compute the frequency of edges, the average frequency of edges will not equal 3. The probability  $p(f = 5) = p(f = 3) = p(f = 1) = \frac{1}{3}$  will not be right based on these incomplete frequency graphs. The probability that an edge  $e$  has the frequency above 3 is not equal to  $\frac{2}{3}$  in the various incomplete frequency quadrilaterals. Therefore, we should use a different ratio rather than  $\frac{1}{3}$  to discard the number of edges according to their average frequency, especially in the later computation stage.

In the later computation process, most of edges in the preserved graphs will only be included in the incomplete quadrilaterals. According to various incomplete quadrilaterals, it is hard to find a suitable ratio to delete the proper number of edges in  $G_k(V_0, E_k)$ . If we use the constant ratio  $\frac{1}{3}$  to abandon the edges with small frequency, some or many *OHC* edges will be neglected, too. To guarantee the *OHC* edges in the last preserved graph, we will find the stop computation cycle  $k_s$  to terminate the iterative algorithm.

If  $N$  is big enough for an edge  $e \in OHC$  in  $G_k(V_0, E_k)$ , the average frequency of  $e$  will be bigger than the average frequency of the total  $|E_k|$  edges based on frequency quadrilaterals. Thus, the average frequency of  $e \in OHC$  will be bigger than 3 when it is computed with  $N$  frequency quadrilaterals. We enumerate the number of edges with the average frequency less than 3 and note it as  $N_{<\bar{f}}$ . When we use the constant ratio  $\frac{1}{3}$  to trim the  $\frac{1}{3}|E_k|$  edges with small frequency, the *OHC* will be maintained in the preserved graph if  $N_{<\bar{f}} > \frac{1}{3}|E_k|$ . It means we just eliminate the  $\frac{1}{3}|E_k|$  edges whose average frequency is below 3. However, we will eliminate some *OHC* edges if we meet  $N_{<\bar{f}} \leq \frac{1}{3}|E_k|$  in the graph  $G_k(V_0, E_k)$ . Therefore, the inequality  $N_{<\bar{f}} \leq \frac{1}{3}|E_k|$  is taken as the restriction to determine the stop computation cycle  $k_s$  and terminate the iterative algorithm. Given a *TSP*, the iterative algorithm can always run until it reaches the stop computation cycle. Once  $N_{<\bar{f}} \leq \frac{1}{3}|E_k|$ , we should be careful to implement the iterative computation. Some *OHC* edges whose average frequency is above but near to the expected frequency 3 will be abandoned at this computation cycle.

In the computation process, the number of edges with the average frequency below 3 will become less and less according to  $k$ . On the other hand, the number of edges with average frequency above 3 will become relatively bigger according to  $k$ . Therefore, we can always find such a preserved graph  $G_k(V_0, E_k)$  where  $N_{<\bar{f}} \leq \frac{1}{3}|E_k|$ . If the minimum average frequency of the *OHC* edge is still bigger than 3 or the average frequency of the *OHC* edges is further beyond that of the  $\frac{1}{3}|E_k|$  edges with small frequency, we may proceed the iterative algorithm one or a few more times even though the  $N_{<\bar{f}} \leq \frac{1}{3}|E_k|$ . In this case, the computation cycle  $k$  will be close to  $k_{max}$  and the residual graph will be

very sparse. However, we cannot guarantee to preserve all of the *OHC* edges in the following preserved graphs for the worst cases of *TSP* once  $k > k_s$ . In the actual computation process, we enumerate the edges with the average frequency below the expected frequency 3 simultaneously. Once  $N_{<\bar{f}} \leq \frac{1}{3}|E_k|$ , it means that we may throw away some *OHC* edges at this computation cycle. It is the time to stop the iterative algorithm and take the output graph with  $|E_{k-1}|$  edges for *TSP*.

It mentions that many incomplete quadrilaterals will be generated in the computation process. Figure 3(a) and (b) shows two kinds of incomplete quadrilaterals we consider in our algorithm, especially at the final stages of the algorithm. Their possible frequency quadrilaterals (1), (2) are shown on their right sides. These incomplete frequency quadrilaterals are computed with the  $OP^4$ s in the two incomplete quadrilaterals. In the frequency quadrilaterals (1) and (2), the numbers on the edges are their frequency. Obviously, the probability  $4/5$  and  $1$  that we preserve the edges based on the two incomplete frequency quadrilaterals is bigger than  $2/3$  according to frequency quadrilaterals. It means we should throw away  $\frac{1}{5}|E_k|$  and  $0$  edges according to their frequency or average frequency computed with the two kinds of incomplete frequency quadrilaterals. It suggests us to keep more edges in the computation process when we use a lot of such incomplete frequency quadrilaterals. In the later computation cycles, we will have many such incomplete quadrilaterals. In this case, we generally cannot use the constant ratio  $\frac{1}{3}$  to eliminate the  $\frac{1}{3}|E_k|$  edges with low frequency to compute the next sparse graph for *TSP*.

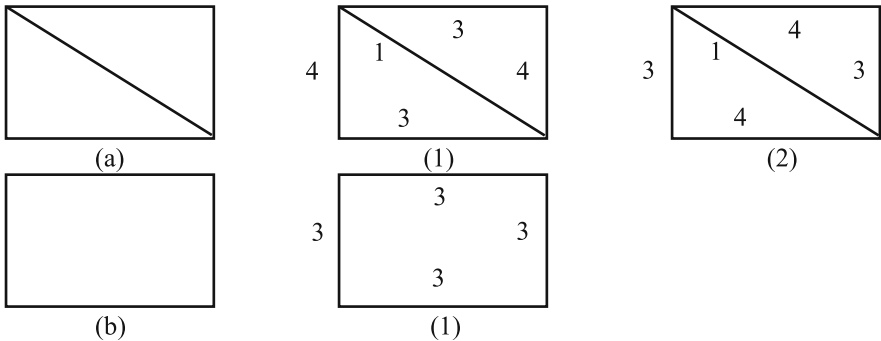


Fig. 3. Two kinds of incomplete quadrilaterals and their corresponding frequency graphs

### 4 The Experiments and Analysis

The experiments occupy 10 pages which exceed the pages limitation of the proceedings. We delete the experimental results for the proceedings. The full version of the paper has been submitted to the <https://arxiv.org/>. The TSP instances are selected from TSPLIB [20]. The optimal solutions of these TSP instances are computed with Concorde online [21].

## 5 Conclusions

We design a heuristic algorithm based on the frequency quadrilaterals to compute the sparse graph for *TSP*. When the frequency of an edge  $e$  is computed with  $N$  frequency quadrilaterals containing  $e$ , the frequency of the *OHC* edges will be bigger than the average frequency  $3N$  of all of edges when  $N$  is big enough. The probability model shows it is likely  $\lceil \frac{2}{3}|E| \rceil$  edges whose frequency is above the average frequency 3. Thus, we can eliminate  $\lceil \frac{1}{3}|E| \rceil$  edges with low frequency so as to compute a preserved graph for *TSP*. We iterate the elimination process until a sparse graph is obtained for *TSP* as  $N$  is big enough in the preserved graphs. In an ideal case, a sparse graph with  $cn$  edges is computed at the  $\left\lceil \log_{\frac{2}{3}}\left(\frac{2c}{n-1}\right) \right\rceil^{\text{th}}$  computation cycles where  $c = \log_2 n$ . We tested the algorithm with tens of various *TSP* instances. The experimental results showed that our probability model works well for general *TSP* instances. The sparse graphs with  $O(n \log_2(n))$  edges are computed for these instances. It says the average frequency of the *OHC* edges is bigger than that of the  $1/3$  edges to be eliminated not only in the  $K_n$ , but also in the preserved graphs that the algorithm computes. Thus, the *OHC* edges are always preserved in the computation process until the stop computation cycle is arrived.

In the near future, the properties of the residual graphs will be analyzed. We expect the sparse graphs have the good properties, such as bounded degree, genus, tree-width and planarity, etc. so that we can design the polynomial-time algorithms or polynomial-time approximation algorithms for *TSP* based on the sparse graphs. In addition, the other terminal conditions will be explored to compute the sparse graphs with good properties.

**Acknowledgement.** The authors acknowledge the anonymous referees for their suggestions to improve the paper. We acknowledge W. Cook, H. Mittelman who created the Concorde and G. Reinelt, et al. who provided the *TSP* data to *TSPLIB*. The authors acknowledge the funds supported by NSFC (No. 51205129) and the Fundamental Research Funds for the Central Universities (No. 2015ZD10). We also thank the support of Beijing Key Laboratory of Energy Safety and Clean Utilization.

## References

1. Karp, R.M.: On the computational complexity of combinatorial problems. *Networks (U.S.A.)* **5**(1), 45–68 (1975)
2. Gutin, G., Punnen, A.P. (eds.): *The Traveling Salesman Problem and Its Variations*. Springer, New York (2007). <https://doi.org/10.1007/b101971>
3. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. *J. Soc. Ind. Appl. Math.* **10**(1), 196–210 (1962)
4. Bellman, R.: Dynamic programming treatment of the travelling salesman problem. *J. ACM* **9**(1), 61–63 (1962)
5. de Klerk, E., Dobre, C.: A comparison of lower bounds for the symmetric circulant traveling salesman problem. *Discret. Appl. Math.* **159**(16), 1815–1826 (2011)

6. Levine, M.S.: Finding the right cutting planes for the TSP. *ACM J. Exp. Algorithmics (JEA)* **5**(6), 1–20 (2000)
7. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W., Espinoza, D.G., Goycoolea, M., Helsgaun, K.: Certification of an optimal TSP tour through 85900 cities. *Oper. Res. Lett.* **37** (1), 11–15 (2009)
8. Thomas, H.C., Charles, E.L., Ronald, L.R., Clifford, S.: *Introduction to Algorithm*, 2nd edn. China Machine Press, Beijing (2006)
9. Hoogeveen, J.A.: Analysis of Christofides' heuristic: some paths are more difficult than cycles. *Oper. Res. Lett.* **10**(5), 291–295 (1991)
10. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic (2012). [www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/](http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/)
11. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: The traveling salesman problem in bounded degree graphs. *ACM Trans. Algorithms* **8**(2), 1–18 (2012)
12. Gharan, S.O., Saberi, A.: The asymmetric traveling salesman problem on graphs with bounded genus. In: *SODA 2011 Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 967–975. ACM, New York (2011)
13. Bartal, Y., Gottlieb, L.A., Krauthgamer, R.: The traveling salesman problem: low-dimensionality implies a polynomial time approximation scheme. In: *STOC 2012: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, pp. 663–672. ACM, New York (2012)
14. Hougardy, S., Schroeder, R.T.: Edge elimination in TSP instances. In: Kratsch, D., Todinca, I. (eds.) *WG 2014. LNCS*, vol. 8747, pp. 275–286. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-12340-0\\_23](https://doi.org/10.1007/978-3-319-12340-0_23)
15. Wang, Y.: A representation model for TSP. In: *15th IEEE International Conference on High Performance Computing and Communications, HPCC 2013 and 11th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 204–209. IEEE, New York (2013)
16. Wang, Y.: An approximate method to compute a sparse graph for traveling salesman problem. *Expert Syst. App.* **42**(12), 5150–5162 (2015)
17. Wang, Y.: Statistical analysis of frequency graph for traveling salesman problem. *J. Intell. Fuzzy Syst.* **28**(3), 1109–1118 (2015)
18. Wang, Y., Remmel, J.B.: A binomial distribution model for the traveling salesman problem based on frequency quadrilaterals. *J. Graph Algorithms App.* **20**(2), 411–434 (2016)
19. Wang, Y.: An approximate algorithm for triangle TSP with a four-vertex three-line inequality. *Int. J. Appl. Metaheuristic Comput.* **6**(1), 35–46 (2015)
20. Reinelt, G. (2016). <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/>
21. Mittelmann, H.: NEOS Server for Concorde (2016). <http://neos-server.org/neos/solvers/co:concorde/TSP.html>



# Optimal Length Tree-Like Refutations of Linear Feasibility in UTVPI Constraints

P. Wojciechowski<sup>1</sup>, K. Subramani<sup>1(✉)</sup>, and Matthew Williamson<sup>2</sup>

<sup>1</sup> LCSEE, West Virginia University, Morgantown, WV, USA

`pwojciec@mix.wvu.edu, ksmani@csee.wvu.edu`

<sup>2</sup> MCIS, Marietta College, Marietta, OH, USA

`williamm@marietta.edu`

**Abstract.** In this paper, we propose two algorithms for determining the optimal length tree-like refutation of linear feasibility in Unit Two Variable Per Inequality (UTVPI) constraints. Given an infeasible UTVPI constraint system (UCS), a refutation certifies its infeasibility. The problem of finding refutations in a UCS finds applications in domains such as program verification and operations research. In general, there exist several **types** of refutations of feasibility in constraint systems. In this paper, we focus on a specific type of refutation called a **tree-like refutation**. Tree-like refutations are **complete**, in that if a system of linear constraints is infeasible, then it must have a tree-like refutation. Associated with a refutation is its length which corresponds to the total number of constraints (including repeats) that are used to establish the infeasibility of the corresponding linear constraint system. Our goal in this paper is to find the optimal (minimum) length tree-like refutation (OTLR) of an infeasible UCS. We show that an OTLR of a UCS can be found in  $O(m \cdot n \cdot k)$  time, where  $m$  is the number of constraints,  $n$  is the number of variables in the system, and  $k$  is the length of an OTLR. We also propose a true-biased, randomized algorithm for this problem. This algorithm runs in  $O(m \cdot n \cdot \log n)$  time, and returns an OTLR with probability  $(1 - \frac{1}{e})$ .

## 1 Introduction

In this paper, we discuss the design and analysis of algorithms for the problem of determining optimal-length tree-like refutations (OTLR) of linear feasibility in a class of constraints called Unit Two Variable per Inequality (UTVPI) constraints. These constraints arise in a number of applications including program verification, abstract interpretation [1] and packing. Our focus in this paper is on

---

P. Wojciechowski was supported in part by the National Science Foundation through Award CCF-1305054, and by NASA through the West Virginia Space Grant.

K. Subramani was supported in part by the Air Force Research Laboratory under US Air Force contract 88ABW-2017-1077.

refutations of linear feasibility in these constraints. In particular, we will focus on tree-like refutations. We shall show that this problem is in **P**.

The principal contributions of this paper are as follows:

1. A deterministic polynomial time algorithm for the problem of finding an OTLR of linear feasibility of a UCS (Sect. 5).
2. A fast, randomized algorithm for the same problem (Sect. 6).

## 2 Constraint Preliminaries

In this section, we discuss the terminology used in this paper. System (1) denotes a system of linear inequalities (or linear program), where **A** has dimensions  $m \times n$ ,  $\mathbf{x} \in \mathbb{R}_n$ , and **b** is an  $m$ -vector.

$$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \tag{1}$$

**Definition 1.** A constraint of the form  $a_i \cdot x_i \leq b_i$  is called an absolute constraint if  $a_i \in \{1, -1\}$ .

**Definition 2.** A constraint of the form  $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$  is called a difference constraint, if  $a_i, a_j \in \{1, -1\}$  and  $a_i = -a_j$ .

A conjunction of difference constraints is called a difference constraint system (DCS).

**Definition 3.** A constraint of the form  $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$  is called a Unit Two Variable per Inequality (UTVPI) constraint, if  $a_i, a_j \in \{1, -1\}$ .

A conjunction of UTVPI constraints is called a UTVPI constraint system (UCS).

In the above definitions,  $b_{ij}$  is called the defining constant of the constraint. For instance,  $x_1 \leq 5$  is an absolute constraint,  $x_1 - x_2 \leq 4$  is a difference constraint, and  $x_1 + x_2 \leq 4$  is a UTVPI constraint.

We are interested in certificates of infeasibility. In particular, we are interested in resolution refutations. In linear programs (systems of linear inequalities), resolution refutations are accomplished via repeated applications of the following rule:

$$\frac{\sum_{i=1}^n a_i \cdot x_i \leq b_1 \quad \sum_{i=1}^n a'_i \cdot x_i \leq b_2}{\sum_{i=1}^n (a_i + a'_i) \cdot x_i \leq b_1 + b_2} \tag{2}$$

Rule (2) is known as the **Addition rule** in the literature.

**Definition 4.** A constraint obtained by the application of the Addition rule to two constraints is called a derived constraint.

Note that when the Addition rule is used, one or both of the constraints involved could themselves be derived constraints.

**Definition 5.** Given a system of linear constraints  $\mathbf{L}$ , a refutation of feasibility is a sequence of constraints such that:

1. Each constraint is either in  $\mathbf{L}$  or can be derived from two preceding constraints by the Addition rule.
2. The last constraint establishes infeasibility in the form of the contradiction  $0 \leq -b, b > 0$ .

*Example 1.* Suppose we are given the following difference constraint system:

$$\begin{aligned} l_1 : x_1 - x_2 \leq -1 & \quad l_2 : x_2 - x_3 \leq -1 & \quad l_3 : x_3 - x_4 \leq 1 \\ l_4 : x_4 - x_1 \leq 1 & \quad l_5 : x_3 - x_1 \leq -1 \end{aligned} \tag{3}$$

The following is a refutation of System (3):

1. Apply the addition rule to  $l_1$  and  $l_2$  to get  $l_6 : x_1 - x_3 \leq -2$ .
2. Apply the addition rule to  $l_5$  and  $l_6$  to get  $0 \leq -3$ .

Clearly, this is a contradiction. Therefore, the constraints  $l_1, l_2$ , and  $l_5$  form a refutation that certifies the infeasibility of System (3).

It is easy to see that refutation using Rule (2) is **sound** in that any assignment satisfying the antecedents **must** satisfy the consequent. Furthermore, it is **complete** in that if System (1) is unsatisfiable, then repeated applications of Rule (2) will result in a contradiction of the form:  $0 \leq -b, b > 0$ . The completeness of the Addition rule was established by Farkas (see [2]). Farkas’ observation is known as Farkas’ Lemma and we state it below. For a proof of the lemma and its implications, see [13].

**Lemma 1.** Let  $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$  denote a system of  $m$  linear constraints over  $n$  variables. Then, either  $\exists \mathbf{x} \ \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$  or (mutually exclusively)  $\exists \mathbf{y} \in \mathbb{R}_m^+ \ \mathbf{y} \cdot \mathbf{A} = \mathbf{0}, \ \mathbf{y} \cdot \mathbf{b} < 0$ .

In Lemma 1, the vector  $\mathbf{y}$  can be chosen from the set of non-negative integers, without affecting its correctness. When  $\mathbf{y}$  is integral, it can be interpreted as representing the number of times each constraint is used in the corresponding refutation. In this paper, we are interested only in cases where the Farkas’ variables,  $\mathbf{y}$ , are integral.

Lemma 1, along with the fact that linear programs must have basic feasible solutions, establishes that the linear programming problem is in the complexity class  $\mathbf{NP} \cap \mathbf{coNP}$ . Farkas’ lemma is one of several lemmata that consider pairs of linear systems in which exactly one element of the pair is feasible. These lemmata are collectively referred to as “Theorems of the Alternative” [10]. The  $\mathbf{y}$  variables are called the Farkas’ variables corresponding to the system  $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$  and they serve as a witness that certifies the infeasibility of this system.

When considering UCSs, we can restrict the Addition rule as follows:

$$\frac{a_i \cdot x_i + a_j \cdot x_j \leq b_{ij} \quad -a_j \cdot x_j + a_k \cdot x_k \leq b_{jk}}{a_i \cdot x_i + a_k \cdot x_k \leq b_{ij} + b_{jk}}$$

This is a restricted version of the addition rule known as the transitive inference rule [17]. Even with this restriction, the transitive inference rule is complete [8].



*Example 2.* Consider the constraints  $x_1 + x_2 \leq 3$  and  $-x_2 + x_3 \leq 4$ . Using the transitive inference rule, we can derive the constraint  $x_1 + x_3 \leq 7$ .

### 2.1 Constraint Network Presentation

The algorithms in this paper utilize the constraint network introduced in [17]. Let  $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$  denote the UCS and let  $\mathbf{X}$  denote the set of all solutions to  $\mathbf{U}$ . Corresponding to this constraint system, we construct the constraint network  $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b} \rangle$  as follows. For each variable  $x_i$ , create a node in  $\mathbf{V}$ . For ease of reference, both a variable and its corresponding node are denoted as  $x_i$ .  $\mathbf{G}$  is stored as an adjacency list. Constraints are represented as edges using the following rules:

- (a) A constraint of the form  $x_i - x_j \leq b_{ij}$  is represented as an undirected “gray” edge,  $(x_j \overset{b_{ij}}{\square} x_i)$ , or  $(x_i \overset{b_{ij}}{\square} x_j)$ , with cost  $b_{ij}$ .
- (b) A constraint of the form  $-x_i - x_j \leq b_{ij}$  is represented by an undirected “black” edge,  $(x_i \overset{b_{ij}}{\blacksquare} x_j)$ , with cost  $b_{ij}$ .
- (c) A constraint of the form  $x_i + x_j \leq b_{ij}$  is represented by an undirected “white” edge,  $(x_i \overset{b_{ij}}{\square} x_j)$ , with cost  $b_{ij}$ .

Finally, we add a node  $x_0$  to the network. This node permits the addition of absolute constraints. Each absolute constraint  $x_i \leq b_i$  is replaced by a pair of constraints  $x_i + x_0 \leq b_i$  and  $x_i - x_0 \leq b_i$ . The corresponding edges  $(x_0 \overset{b_i}{\square} x_i)$  and  $(x_0 \overset{b_i}{\blacksquare} x_i)$  are added to the constraint network. Note that the constraint network has  $(n + 1)$  nodes and  $m' \leq m + 2 \cdot n$  edges. We have that  $m' \leq m + 2 \cdot n$  since each absolute constraint adds two edges instead of one and there are at most  $2 \cdot n$  absolute constraints.

Because we have multiple types of edges, each node  $x_i$  has four lists:

- (1) A list of  $(x_j, b_{ij})$  such that the edge  $(x_i \overset{b_{ij}}{\square} x_j)$  is in  $\mathbf{G}$ .
- (2) A list of  $(x_j, b_{ij})$  such that the edge  $(x_i \overset{b_{ij}}{\blacksquare} x_j)$  is in  $\mathbf{G}$ .
- (3) A list of  $(x_j, b_{ij})$  such that the edge  $(x_i \overset{b_{ij}}{\square} x_j)$  is in  $\mathbf{G}$ .
- (4) A list of  $(x_j, b_{ij})$  such that the edge  $(x_i \overset{b_{ij}}{\blacksquare} x_j)$  is in  $\mathbf{G}$ .

This allows us to easily process all edges involving  $x_i$  and all neighbors of  $x_i$ . Note that the nature of gray edges prevents the adjacency list from being fully symmetric. If  $\mathbf{G}$  has the edge  $(x_i \overset{b_{ij}}{\square} x_j)$ , then  $(x_j, b_{ij})$  appears in list (1) for  $x_i$ , and  $(x_i, b_{ij})$  appears in list (1) for  $x_j$ . However, if  $\mathbf{G}$  has the edge  $(x_i \overset{b_{ij}}{\blacksquare} x_j)$ , then  $(x_j, b_{ij})$  appears in list (3) for  $x_i$ , and  $(x_i, b_{ij})$  appears in list (4) for  $x_j$ .

**Definition 6.** A *k-path* in our constraint network is a sequence of  $(k+1)$  nodes,  $x_1, x_2, \dots, x_{k+1}$ , and  $k$  edges  $e_1, e_2, \dots, e_k$ , such that  $e_i$  is the edge corresponding to one of the constraints between  $x_i$  and  $x_{i+1}$  in the UTVPI constraint system.

**Definition 7.** A  $k$ -path is considered **valid** if it has the following property: For every  $i$  from 2 to  $k$ , the coefficients of  $x_i$  in the constraints corresponding to the edges  $e_i$  and  $e_{i-1}$  have opposite signs.

Note that summing all the constraints corresponding to a valid path results in a UTVPI constraint.

**Definition 8.** The **cost** of a path is the sum of the costs of the edges along that path.

**Definition 9.** A **cycle** is a valid  $k$ -path for which  $x_1 = x_{k+1}$ .

Note that a cycle can consist of edges and nodes that occur more than once. Thus, the notion of a cycle in this paper differs from the traditional notion of a cycle in a directed graph.

We also utilize the concept of edge reductions introduced in [17].

**Definition 10.** An edge reduction is an operation which determines a single edge equivalent to a two-edge path and represents the addition of the two UTVPI constraints which correspond to the edges in question. If this addition results in a UTVPI constraint, the reduction is said to be **valid**.

*Example 3.* Consider the following 2-paths:

1.  $(x_i \overset{b_{ij}}{\square} x_j \overset{b_{jk}}{\square} x_k)$ : The first edge corresponds to the constraint  $x_i + x_j \leq b_{ij}$  and the second edge corresponds to the constraint  $x_j + x_k \leq b_{jk}$ . Summing these constraints produces the non-UTVPI constraint  $x_i + 2x_j + x_k \leq b_{ij} + b_{jk}$ . In other words, this path cannot be reduced to an edge. Thus, this edge reduction and the corresponding path are not valid.
2.  $(x_i \overset{b_{ij}}{\square} x_j \overset{b_{jk}}{\blacksquare} x_k)$ : The first edge corresponds to the constraint  $x_i + x_j \leq b_{ij}$  and the second edge corresponds to the constraint  $-x_j - x_k \leq b_{jk}$ . Summing these constraints produces a UTVPI constraint and hence this path can be reduced to the edge  $(x_i \overset{b_{ij}+b_{jk}}{\blacksquare} x_k)$ . Thus, this edge reduction and the corresponding path are valid.

We use Definition 10 to define paths in the constraint network.

**Definition 11.** We say that a path has type  $t$ , if it can be reduced to a single edge of type  $t$ , where  $t \in \{ \square, \blacksquare, \blacksquare, \blacksquare \}$  by a series of valid edge reductions.

Using this we can define a negative cost gray cycle.

**Definition 12.** A **negative cost gray cycle** is a path which can be reduced to an edge  $(x_i \overset{b_i}{\blacksquare} x_i)$  or  $(x_i \overset{b_i}{\blacksquare} x_i)$  for some node  $x_i$  such that  $b_i < 0$ .

It was shown in [17] that a UCS is infeasible if and only if the corresponding constraint network contains a negative cost gray cycle.

### 3 Statement of Problem

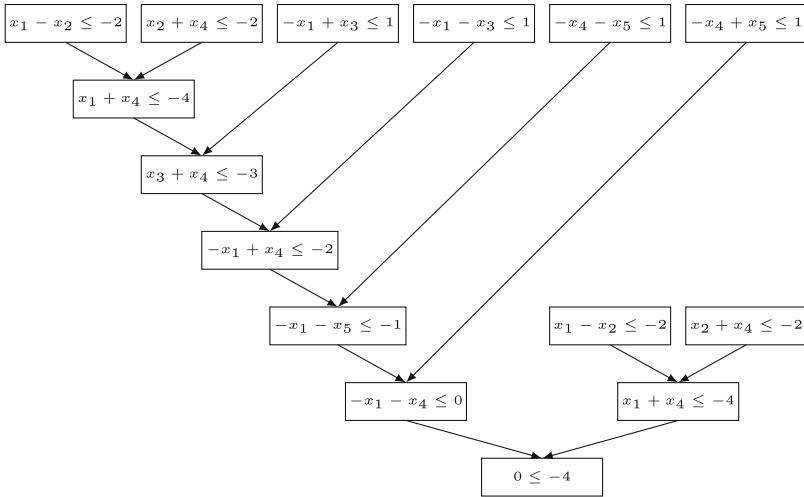
In this section, we utilize Farkas' Lemma to formally define the OTLR problem. We first need to define tree-like refutations and relate them to negative cost gray cycles.

**Definition 13.** A *tree-like refutation* is a refutation in which each derived constraint can be used at most once.

Note that in tree-like refutations, the input constraint system can be used multiple times. Thus, any derived constraint can be derived multiple times.

*Example 4.* Consider the following system:

$$\begin{aligned}
 l_1 : x_1 - x_2 \leq -2 & \quad l_2 : -x_1 + x_3 \leq 1 & \quad l_3 : -x_1 - x_3 \leq 1 \\
 l_4 : x_2 + x_4 \leq -2 & \quad l_5 : -x_4 - x_5 \leq 1 & \quad l_6 : -x_4 + x_5 \leq 1
 \end{aligned}
 \tag{4}$$



**Fig. 1.** Tree-like refutation

Figure 1 depicts the following tree-like refutation of System (4):

1. Apply the addition rule to  $l_1$  and  $l_4$  to get  $l_7 : x_1 + x_4 \leq -4$ .
2. Apply the addition rule to  $l_2$  and  $l_7$  to get  $l_8 : x_3 + x_4 \leq -3$ .
3. Apply the addition rule to  $l_3$  and  $l_8$  to get  $l_9 : -x_1 + x_4 \leq -2$ .
4. Apply the addition rule to  $l_5$  and  $l_9$  to get  $l_{10} : -x_1 - x_5 \leq -1$ .
5. Apply the addition rule to  $l_6$  and  $l_{10}$  to get  $l_{11} : -x_1 - x_4 \leq 0$ .
6. We could derive the contradiction  $0 \leq -4$  by applying the addition rule to  $l_7$  and  $l_{11}$ . However,  $l_7$  is a derived constraint and thus cannot be reused in a tree-like refutation. Thus, it must first be rederived.

7. Apply the addition rule to  $l_1$  and  $l_4$  to get  $l_{12} : x_1 + x_4 \leq -4$ .
8. Apply the addition rule to  $l_{11}$  and  $l_{12}$  to get  $0 \leq -4$ .

Note that the summation of constraints is commutative. Thus, the contradiction resulting from a tree-like refutation  $C$  of a linear system  $\mathbf{U}$  can be obtained by summing (with repeats) the constraints in  $C$  that are also in  $\mathbf{U}$ .

*Example 5.* The contradiction ( $0 \leq -4$ ) obtained by the preceding tree-like refutation of System (4) can be obtained by summing the constraints as follows

$$2 \cdot l_1 + l_2 + l_3 + 2 \cdot l_4 + l_5 + l_6.$$

The contradiction ( $0 \leq -3$ ) obtained by the refutation of System (3) can be obtained by summing the constraints as follows

$$l_1 + l_2 + l_5.$$

Note that this refutation does not need to reuse either original constraints or derived constraints. Thus it is a **read-once** refutation.

As a direct consequence of Farkas' Lemma, tree-like refutations are complete [2].

**Definition 14.** *The length of a tree-like refutation,  $C$ , is equal to the total number of constraints in  $C$  that are in the original system. Each time a constraint is reused, it contributes to the length of  $C$ .*

*Example 6.* Consider the tree-like refutation of System (4). In this refutation the constraints  $l_1$  and  $l_4$  are used twice and the other constraints ( $l_2, l_3, l_5,$  and  $l_6$ ) are used once. Thus, this refutation has length 8.

We formally define an OTLR as follows:

**Definition 15.** *An optimal length tree-like refutation (OTLR) of a system of constraints  $\mathbf{U}$  is a tree-like refutation of  $\mathbf{U}$  with the smallest length.*

From Farkas' Lemma, the OTLR problem can be modeled as the following integer program:

$$\begin{aligned} \min \sum_{i=1}^m y_i & & (5) \\ \mathbf{y} \cdot \mathbf{A} &= \mathbf{0} \\ \mathbf{y} \cdot \mathbf{b} &< 0 \\ \mathbf{y} &\in \mathbb{Z}_m^+ \end{aligned}$$

We now observe that, unlike DCSs, UCSs may require refutations that use constraints more than once. Curiously, every infeasible UCS has a refutation in which each constraint is used at most twice [17].

*Example 7.* Consider the following system:

$$\begin{aligned}
 l_1 : x_1 - x_2 &\leq -3 & l_2 : -x_1 + x_4 &\leq 1 & l_3 : -x_1 - x_4 &\leq 1 \\
 l_4 : x_2 + x_3 &\leq 1 & l_5 : x_2 - x_3 &\leq 1 & & 
 \end{aligned}
 \tag{6}$$

First observe that  $l_1$  is the only constraint with a negative defining constant. Hence, it must be included in **any** refutation. This constraint corresponds to the edge  $(x_1 \overset{-3}{\blacksquare} x_2)$ . Thus, this edge must appear in any cycle corresponding to the refutation.

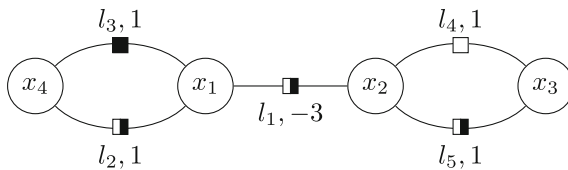
In order to eliminate  $x_1$ , we must include both  $l_2$  and  $l_3$  in the refutation. If we do not include both, then  $x_4$  will not be eliminated. These constraints correspond to the path  $(x_1 \overset{1}{\blacksquare} x_4 \overset{1}{\blacksquare} x_1)$ . Note that this path ends on node  $x_1$ . Thus, this path must precede the edge  $(x_1 \overset{-3}{\blacksquare} x_2)$  in the cycle corresponding to the refutation.

Similarly, to eliminate  $-x_2$ , we must include both  $l_4$  and  $l_5$ . If we do not include both, then  $x_3$  will not be eliminated. These constraints correspond to the path  $(x_2 \overset{1}{\square} x_3 \overset{1}{\blacksquare} x_2)$ . Note that this path starts from node  $x_2$ . Thus, this path must follow the edge  $(x_1 \overset{-3}{\blacksquare} x_2)$  in the cycle corresponding to the refutation.

Summing the constraints used so far results in the constraint  $x_2 - x_1 \leq 1$ . Thus, to finish the refutation we must include the constraint  $l_1$  a second time. This means that the edge  $(x_2 \overset{-3}{\blacksquare} x_1)$  must be added to the end of the cycle.

Thus, the refutation corresponds to cycle  $(x_1 \overset{1}{\blacksquare} x_4 \overset{1}{\blacksquare} x_1 \overset{-3}{\blacksquare} x_2 \overset{1}{\square} x_3 \overset{1}{\blacksquare} x_2 \overset{-3}{\blacksquare} x_1)$  in Fig. 2.

This cycle reduces to the edge  $(x_1 \overset{-2}{\blacksquare} x_1)$ . Therefore, it is a negative cost gray cycle.



**Fig. 2.** Example constraint network (without node  $x_0$ )

From the preceding example we see a relationship between negative cost gray cycles and tree-like refutations. This relationship is substantiated by the following theorem.

**Theorem 1.** *A UCS  $\mathbf{U}$  has a tree-like refutation of length  $l$  if and only if the corresponding constraint network  $\mathbf{G}$  has a negative cost gray cycle of length  $l$ .*

*Proof.* In the journal version of the paper.

## 4 Motivation and Related Work

Unit Two Variable Per Inequality (UTVPI) constraints arise in a number of problem domains, including but not limited to program verification [7], abstract interpretation [1, 8], real-time scheduling [3] and operations research.

The focus of this paper is on proofs of linear infeasibility in UCSs. Such proofs are very important from the perspective of designing certifying algorithms. In a certifying algorithm, both positive and negative answers must be accompanied by “certificates” which attest to the validity of the answer. For general linear programs, strong duality (Farkas’ lemma) enables us to derive proofs of infeasibility.

Proofs of infeasibility are also referred to as refutations. There exist a number of refutation types, depending upon how the input constraints can be used in the construction of a proof of infeasibility. Our focus is on a class of refutations called resolution refutations. In resolution refutations, there is only one inference rule, viz., the transitive inference rule. The three major types of (resolution) refutations are **read-once**, **tree-like** and **dag-like** [4, 5]. As already established in the previous section, read-once proofs are not **complete** for the purpose of refuting linear feasibility in UCSs. However, both tree-like and dag-like proof systems are complete. In this paper, we will focus exclusively on tree-like refutations.

Optimal length proofs (refutations) of various types and for various constraint systems have been studied extensively in the literature. In [15], optimal-length tree-like proofs were studied for 2CNF formulae. In [16], it was established that read-once, tree-like, and dag-like proofs coincide for difference constraints systems.

In [11], a randomized algorithm was proposed for the optimal length refutation problem in difference constraints. This paper generalizes that algorithm for a larger class of constraints.

We also note that [8] proposes an alternate network representation for UCSs. However, this network cannot handle absolute constraints.

## 5 A Path Following Approach

In this section, we exploit the observations in Sect. 2.1 to design a simple, path following algorithm for the OTLR problem. From Theorem 1, a negative cost gray cycle in  $\mathbf{G}$  corresponds to a tree-like refutation of the original UCS. Thus, the shortest such cycle in  $\mathbf{G}$  corresponds to an OTLR of the original UCS. The following observations result in Algorithms 5.1 and 5.2.

1. Let  $d_i^{(k,t)}(j)$  denote the length of the shortest path of type  $t$  from node  $x_i$  to node  $x_j$  with at most  $k$  edges.
2. Let  $d_i^{(k)}()$  contain  $d_i^{(k,t)}(j)$  for all  $j = 1 \dots n$  and  $t \in \{ \square, \blacksquare, \blacksquare \cdot \blacksquare \}$ .
3. We initially set  $d_i^{(0,t)}(i) = 0$  and  $d_i^{(0,t)}(j) = \infty$  for each  $t \in \{ \square, \blacksquare, \blacksquare \cdot \blacksquare \}$  and  $j \neq i$ .

4. From [17], we have that,

$$\begin{aligned}
 d_i^{(k+1, \square)}(j) &= \min \begin{cases} d_i^{(k, \square)}(r) + b(x_r \square x_j), r \text{ is a neighbor of } j \\ d_i^{(k, \blacksquare)}(r) + b(x_r \square x_j), r \text{ is a neighbor of } j \end{cases} \\
 d_i^{(k+1, \blacksquare)}(j) &= \min \begin{cases} d_i^{(k, \blacksquare)}(r) + b(x_r \blacksquare x_j), r \text{ is a neighbor of } j \\ d_i^{(k, \blacksquare)}(r) + b(x_r \square x_j), r \text{ is a neighbor of } j \end{cases} \\
 d_i^{(k+1, \blacksquare)}(j) &= \min \begin{cases} d_i^{(k, \blacksquare)}(r) + b(x_r \blacksquare x_j), r \text{ is a neighbor of } j \\ d_i^{(k, \blacksquare)}(r) + b(x_r \blacksquare x_j), r \text{ is a neighbor of } j \end{cases} \\
 d_i^{(k+1, \blacksquare)}(j) &= \min \begin{cases} d_i^{(k, \blacksquare)}(r) + b(x_r \blacksquare x_j), r \text{ is a neighbor of } j \\ d_i^{(k, \square)}(r) + b(x_r \blacksquare x_j), r \text{ is a neighbor of } j \end{cases}
 \end{aligned} \tag{7}$$

5. If we have a negative cost gray cycle of length  $k$  centered around an arbitrary node  $x_i$ , then  $d_i^{(k, \blacksquare)}(i) < 0$ . Observe that any path which reduces to the edge  $(x_i \blacksquare x_i)$  also reduces to the edge  $(x_i \square x_i)$  since these are the same edge. This means that  $d_i^{(k, \blacksquare)}(i) = d_i^{(k, \square)}(i)$ . Thus, it is only necessary to check one of these values.

**Function** SHORTEST-PATH-UTVPI( $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b} \rangle, d_i^{(k)}(), d_i^{(k-1)}()$ )

```

1: for ( $j = 0$  to  $n$ ) do
2:   for ( $t \in \{ \square, \blacksquare, \blacksquare, \blacksquare \}$ ) do
3:      $d_i^{(k,t)}(j) \leftarrow \infty$ .
4:   for (each edge  $(x_r \blacksquare x_j) \in \mathbf{E}$ ) do
5:      $d_i^{(k, \square)}(j) \leftarrow \min\{d_i^{(k, \square)}(j), d_i^{(k-1, \square)}(r) + b(x_r \blacksquare x_j)\}$ 
6:      $d_i^{(k, \blacksquare)}(j) \leftarrow \min\{d_i^{(k, \blacksquare)}(j), d_i^{(k-1, \blacksquare)}(r) + b(x_r \blacksquare x_j)\}$ 
7:   for (each edge  $(x_r \square x_j) \in \mathbf{E}$ ) do
8:      $d_i^{(k, \square)}(j) \leftarrow \min\{d_i^{(k, \square)}(j), d_i^{(k-1, \blacksquare)}(r) + b(x_r \square x_j)\}$ 
9:      $d_i^{(k, \blacksquare)}(j) \leftarrow \min\{d_i^{(k, \blacksquare)}(j), d_i^{(k-1, \blacksquare)}(r) + b(x_r \square x_j)\}$ 
10:  for (each edge  $(x_r \blacksquare x_j) \in \mathbf{E}$ ) do
11:     $d_i^{(k, \blacksquare)}(j) \leftarrow \min\{d_i^{(k, \blacksquare)}(j), d_i^{(k-1, \blacksquare)}(r) + b(x_r \blacksquare x_j)\}$ 
12:     $d_i^{(k, \blacksquare)}(j) \leftarrow \min\{d_i^{(k, \blacksquare)}(j), d_i^{(k-1, \blacksquare)}(r) + b(x_r \blacksquare x_j)\}$ 
13:  for (each edge  $x_r \blacksquare x_j \in \mathbf{E}$ ) do
14:     $d_i^{(k, \blacksquare)}(j) \leftarrow \min\{d_i^{(k, \blacksquare)}(j), d_i^{(k-1, \blacksquare)}(r) + b(x_r \blacksquare x_j)\}$ 
15:     $d_i^{(k, \blacksquare)}(j) \leftarrow \min\{d_i^{(k, \blacksquare)}(j), d_i^{(k-1, \square)}(r) + b(x_r \blacksquare x_j)\}$ 

```

**Algorithm 5.1:** Shortest Path Computation for UTVPI Constraints

**Function** SHORTEST-NEGATIVE-GRAY-CYCLE( $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b} \rangle$ )

- 1: **for** ( $k = 1$  **to**  $(2 \cdot n + 2)$ ) **do**
- 2:     **for** ( $i = 0$  **to**  $n$ ) **do**
- 3:         SHORTEST-PATH-UTVPI( $\mathbf{G}, d_i^{(k)}(), d_i^{(k-1)}()$ ).
- 4:         **if** ( $(d_i^{(k, \blacksquare)}(i) < 0)$ ) **then**
- 5:             **return** ( $k$ ).
- 6: **return** ( $-1$ ). {No negative gray cycle was detected.}

**Algorithm 5.2:** Deterministic Algorithm for UTVPI Constraints

### 5.1 Resource Analysis

We first analyze the running time of Algorithm 5.1.

**Lemma 2.** *Given  $d_i^{(k-1)}()$ , Algorithm 5.1 computes  $d_i^{(k)}()$  in  $O(m')$  time, where  $m'$  is the number of edges in the constraint network.*

*Proof.* Observe that Algorithm 5.1 implements the recurrence relation defined by System (7). Computing  $d_i^{(k,t)}$  is accomplished by relaxing all of the edges in  $\mathbf{G}$ . Relaxing an edge takes  $O(1)$  time, and  $\mathbf{G}$  has  $m'$  edges. Therefore, the running time of Algorithm 5.1 is  $O(m')$ . □

We now analyze the running time of Algorithm 5.2. Let  $T(n, m')$  denote the running time of Algorithm 5.2 on a network with  $(n + 1)$  nodes and  $m'$  edges. Also let  $q \in O(1)$  denote the time for a single edge relaxation. Observe that the **for** loop in lines 2 to 5 has  $O(n)$  iterations. From Lemma 2, we know that line 3 takes  $O(m')$  time. Therefore, the total running time is:

$$T(n, m') \leq \sum_{i=1}^k \sum_{j=0}^n q \cdot m' = q \cdot m' \cdot (n + 1) \cdot k \in O(m' \cdot n \cdot k) = O(m \cdot n \cdot k).$$

### 5.2 Correctness

We now prove the correctness of Algorithm 5.2. First we need the following lemma from [17].

**Lemma 3.** *If the UCS  $\mathbf{U}$  is infeasible, then the corresponding constraint network  $\mathbf{G}$  has a negative cost gray cycle with at most  $(2 \cdot n + 2)$  edges.*

**Theorem 2.** *Algorithm 5.2 always returns the length of an OTLR of the UCS  $\mathbf{U}$  corresponding to the input network  $\mathbf{G}$  or  $-1$  if  $\mathbf{U}$  is feasible.*

*Proof.* We first address the correctness of Algorithm 5.1. As stated in Lemma 2, the algorithm is an implementation of System (7). From [17], System (7) correctly calculates  $d_i^{(k+1)}()$  from  $d_i^{(k)}()$ .



If Algorithm 5.2 returns  $k \neq -1$ , then for some node  $x_i$ ,  $d_i^{(k, \blacksquare)}(i) < 0$ . This means that  $x_i$  is located on a negative cost gray cycle of length  $k$ . From Theorem 1, this means that  $\mathbf{U}$  has a tree-like refutation of length  $k$ .

We also know that for all nodes  $x_j$ ,  $d_j^{(l, \blacksquare)}(j) \geq 0$  for all  $0 < l < k$ . Thus,  $\mathbf{G}$  has no negative cost gray cycles of length less than  $k$ . From Theorem 1, this means that  $\mathbf{U}$  has no tree-like refutations of length less than  $k$ . Thus  $k$  is the length of an OTLR of  $\mathbf{U}$ .

If Algorithm 5.2 returns  $-1$ , then for all nodes  $x_j$ ,  $d_j^{(l, \blacksquare)}(j) \geq 0$  for all  $0 < l \leq (2 \cdot n + 2)$ . Thus,  $\mathbf{G}$  has no negative cost gray cycles with  $(2 \cdot n + 2)$  or fewer edges. By Lemma 3,  $\mathbf{U}$  must be feasible.  $\square$

## 6 A Randomized Approach

In this section, we propose a randomized algorithm for an OTLR problem in UCSs. This algorithm is a generalization of the randomized algorithm for finding shortest negative cost cycles in a directed graph [11].

In each iteration, the algorithm (Algorithm 6.1) processes a randomly chosen node. Let  $v_r$  denote the node chosen by the  $r^{th}$  iteration of this process. The algorithm proceeds by generating the shortest paths from  $v_r$  to every node in the network  $\mathbf{G}$  having at most  $\lceil \frac{2 \cdot n + 2}{r} \rceil$  edges (see Lemma 3).

Algorithm 6.1 represents our strategy to find the shortest negative cost cycle in a UTVPI constraint network with arbitrarily costed edges.

**Function** SHORTEST-NEGATIVE-GRAY-CYCLE( $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b} \rangle$ )

```

1: if ( $\mathbf{G}$  has a negative cost gray cycle) then
2:   Let  $l$  be the number of edges in the cycle.
3: else
4:   return  $(-1)$ .
5: for ( $r = 1$  to  $(n + 1)$ ) do
6:   Let  $x_i$  be a node in  $\mathbf{V}$  chosen uniformly and at random.
7:   for ( $k = 1$  to  $\lceil \frac{2 \cdot n + 2}{r} \rceil$ ) do
8:     SHORTEST-PATH-UTVPI( $\mathbf{G}, d_i^{(k)}(), d_i^{(k-1)}()$ ).
9:     if ( $d_i^{(k, \blacksquare)}(i) < 0$ ) and ( $k < l$ ) then
10:       $l \leftarrow k$ .
11: return  $(l)$ .
```

**Algorithm 6.1:** Randomized Algorithm for UTVPI Constraints

### 6.1 Resource Analysis

Let  $T(n, m')$  denote the running time of Algorithm 6.1 on a network with  $(n + 1)$  nodes and  $m'$  edges. Also let  $q_1 \in O(1)$  denote the amount of time taken by a

single edge relaxation. The check on line 1 can be accomplished in  $O(m' \cdot n)$  time [17]. Indeed, some negative cost gray cycle (not necessarily the one having the fewest number of edges) is returned by the linear feasibility algorithm in [17]. Let  $q_2 \cdot m' \cdot (n + 1)$  be the amount of time taken by this process. The **for** loop on lines 5 has  $O(n)$  iterations. From Lemma 2, it follows that each iteration of the **for** loop on line 7 takes  $O(m')$  time. Thus, we have that:

$$\begin{aligned} T(n, m') &\leq q_2 \cdot m' \cdot (n + 1) + \sum_{r=1}^{n+1} \sum_{k=1}^{\lceil \frac{2 \cdot n + 2}{r} \rceil} q_1 \cdot m' \\ &= q_2 \cdot m' \cdot (n + 1) + q_1 \cdot m' \cdot \left( \sum_{r=1}^{n+1} \sum_{k=1}^{\lceil \frac{2 \cdot n + 2}{r} \rceil} 1 \right) \\ &= q_2 \cdot m' \cdot (n + 1) + q_1 \cdot m' \cdot \left( \sum_{r=1}^{n+1} \left\lceil \frac{2 \cdot n + 2}{r} \right\rceil \right) \\ &\leq q_2 \cdot m' \cdot (n + 1) + q_1 \cdot m' \cdot 2 \cdot (n + 1) \cdot (H_{n+1}) \\ &\in O(m' \cdot n \cdot \log n) \\ &= O(m \cdot n \cdot \log n) \end{aligned}$$

where  $H_n$  is the  $n^{\text{th}}$  harmonic number.

### 6.2 Correctness

We now establish that Algorithm 6.1 returns an OTLR of  $\mathbf{G}$  with high probability.

**Theorem 3.** *Algorithm 6.1 returns the OTLR with probability at least  $(1 - \frac{1}{e})$ .*

*Proof.* If  $\mathbf{G}$  has no negative cost gray cycles, then Algorithm 6.1 returns  $-1$ . Thus, Algorithm 6.1 always returns the correct answer in this case.

If  $\mathbf{G}$  has a negative cost gray cycle, then let  $C$  denote an OTLR of  $\mathbf{G}$ . Let  $N_C$  be the number of nodes in  $C$  and  $|C|$  be the length of  $C$ . Note that  $|C| \leq 2 \cdot N_C$  [17].

Let us compute the probability that  $C$  is discovered during the  $r^{\text{th}}$  iteration of the **for** loop on line 3. If  $r \leq \lceil \frac{n+1}{N_C} \rceil$ , then  $r \leq \lceil \frac{2 \cdot n + 2}{2 \cdot N_C} \rceil \leq \lceil \frac{2 \cdot n + 2}{|C|} \rceil$ . This means that  $|C| \leq \lceil \frac{2 \cdot n + 2}{r} \rceil$ . Thus,  $C$  will be discovered if  $x_i$  lies on  $C$ . This has a probability of  $\frac{N_C}{n+1}$ .

Let  $E_r$  be the event that the  $r^{\text{th}}$  node processed by the algorithm is not a node of  $C$ . Note that these events are independent. We have that  $C$  not being discovered corresponds to the event  $\bigcap_{r=1}^n E_r$  [9]. Thus, the probability that  $C$  is not discovered by Algorithm 6.1 is

$$P \left( \bigcap_{r=1}^n E_r \right) \leq P \left( \bigcap_{r=1}^{\lceil \frac{n+1}{N_C} \rceil} E_r \right) = \prod_{r=1}^{\lceil \frac{n+1}{N_C} \rceil} P(E_r) \leq \left( 1 - \frac{N_C}{n+1} \right)^{\lceil \frac{n+1}{N_C} \rceil} \leq \frac{1}{e}.$$

It follows that Algorithm 6.1 succeeds with probability at least  $(1 - \frac{1}{e}) = 0.632$ . □

Clearly, we can boost the probability of success by running the same procedure multiple times. Indeed, the expected number of runs before finding an OTLR is 2.

## 7 Conclusion

In this paper, we studied the problem of determining an OTLR of a UCS. In particular, we designed two polynomial time algorithms for this problem. The first of these algorithms is deterministic and is based on a path-following approach in a constraint network. The second algorithm is also a polynomial time path-following approach, but it uses randomization and has a non-zero probability of returning a non-optimal refutation.

From our perspective, there are two open problems that are worth pursuing:

1. Implementing and studying the algorithms in this paper in the context of SMT solvers such as Yices [14].
2. Establishing the computational complexities of determining optimal length read-once and dag-like refutations in UCSs.

## References

1. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977
2. Farkas, G.: Über die Theorie der Einfachen Ungleichungen. *Journal für die Reine und Angewandte Mathematik* **124**(124), 1–27 (1902)
3. Gerber, R., Pugh, W., Saksena, M.: Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computers* **44**(3), 471–479 (1995)
4. Iwama, K.: Complexity of finding short resolution proofs. *Lecture Notes in Computer Science* **1295**, 309–319 (1997)
5. K. Iwama and E. Miyano. Intractability of read-once resolution. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (SCTC '95)*, pages 29–36, Los Alamitos, CA, USA, June 1995. IEEE Computer Society Press
6. B. Korte and J. Vygen. *Combinatorial Optimization*. Number 21 in Algorithms and Combinatorics. Springer-Verlag, New York, 4<sup>th</sup> edition, 2010
7. S. K. Lahiri and M. Musuvathi. An Efficient Decision Procedure for UTVPI Constraints. In *Proceedings of the 5<sup>th</sup> International Workshop on the Frontiers of Combining Systems, September 19–21, Vienna, Austria*, pages 168–183, New York, 2005. Springer
8. Miné, A.: The octagon abstract domain. *Higher-Order and Symbolic Computation* **19**(1), 31–100 (2006)
9. Motwani, R., Raghavan, P.: *Randomized Algorithms*. Cambridge University Press, Cambridge, England (1995)
10. Nemhauser, G.L., Wolsey, L.A.: *Integer and Combinatorial Optimization*. John Wiley & Sons, New York (1999)
11. James, B., Orlin, K.: Subramani, and Piotr Wojciechowski. Randomized algorithms for finding the shortest negative cost cycle in networks. *Discrete Applied Mathematics* **236**(1), 387–394 (2018)

12. Toniann Pitassi and Alasdair Urquhart. The complexity of the Hajós calculus. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24–27 October 1992*, pages 187–196, 1992
13. Schrijver, A.: *Theory of Linear and Integer Programming*. John Wiley and Sons, New York (1987)
14. SRI International. *Yices: An SMT solver*. <http://yices.csl.sri.com/>
15. Subramani, K.: Optimal length tree-like resolution refutations for 2sat formulas. *ACM Transactions on Computational Logic* **5**(2), 316–320 (2004)
16. Subramani, K.: Optimal length resolution refutations of difference constraint systems. *Journal of Automated Reasoning (JAR)* **43**(2), 121–137 (2009)
17. Subramani, K., Piotr, J.: Wojciechowski. A combinatorial certifying algorithm for linear feasibility in UTVPI constraints. *Algorithmica* **78**(1), 166–208 (2017)

## Author Index

- Akagi, Toshihiro 263  
Annamalai, Sundar 251  
Araki, Tetsuya 263  
Arora, Pranav 126  
Atulya, M. S. 29
- Banik, Aritra 126  
Bao, Xiaoguang 81
- Cao, Yixin 140  
Chang, Jou-Ming 3  
Chen, Ke 15
- Das, Gautam K. 68  
Das, Gautam 112  
De, Minati 112  
Deng, Xiaotie 183  
Dumitrescu, Adrian 15
- Feng, Qilong 99, 169
- Gargano, Luisa 197
- Horiyama, Takashi 263  
Huang, Neng 169
- Jagalmohanam, Mohith 154  
Jallu, Ramesh K. 68  
Jena, Sangram K. 68
- Kan, Haibin 273  
Kobayashi, Yusuke 54
- Li, Chih-Yu 43  
Li, Songhua 212  
Li, Wenjun 140  
Liu, Gaoang 224  
Liu, Tian 238  
Liu, Xiuying 224  
Liu, Zhaohui 81
- Nakano, Shin-ichi 263  
Nandy, Subhas C. 68, 112  
Narayanaswamy, N. S. 251
- Okamoto, Yoshio 263  
Otachi, Yota 263
- Pai, Kung-Jui 3  
Pal, Manjish 112  
Paliwal, Vijay Kumar 126  
Purohit, Nidhi 29
- Raman, Venkatesh 126  
Rani, M. R. 154  
Rommel, Jeffrey 286  
Rescigno, Adele A. 197
- Saitoh, Toshiki 263  
Subashini, R. 154  
Subramani, K. 300
- Takayama, Koki 54  
Tan, Guanlan 169  
Tang, Yuan 273  
Tripathi, Nitesh 112
- Uehara, Ryuhei 263  
Uno, Takeaki 263
- Vaccaro, Ugo 197  
Vaishali, S. 29
- Wang, Biing-Feng 43  
Wang, Chaoyi 238  
Wang, Jianxin 99, 169  
Wang, Yong 286  
Wasa, Kunihiro 263  
Williamson, Matthew 300  
Wojciechowski, P. 300  
Wu, Ro-Yu 3

Xu, Wei 238

Xu, Yinfeng 212

Yang, Jinn-Shyong 3

Ye, Jhih-Hong 43

Ye, Junjie 140

You, Jie 99

Yu, Wei 81

Zhu, Keyu 183

Zhuo, Beilin 169