Dachuan Xu · Donglei Du
Dingzhu Du (Eds.)

LNCS 9198

Computing and Combinatorics

21st International Conference, COCOON 2015
Beijing, China, August 4–6, 2015
Proceedings

Springer

Dachuan Xu · Donglei Du
Dingzhu Du (Eds.)

Computing and Combinatorics

21st International Conference, COCOON 2015
Beijing, China, August 4–6, 2015
Proceedings

Springer

Editors
Dachuan Xu
Beijing University of Technology
Beijing
China

Dingzhu Du
University of Texas at Dallas
Richardson
USA

Donglei Du
University of New Brunswick
Fredericton
Canada

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-21397-2 ISBN 978-3-319-21398-9 (eBook)
DOI 10.1007/978-3-319-21398-9

Library of Congress Control Number: 2015943451

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media
(www.springer.com)

Preface

The 21st International Computing and Combinatorics Conference (COCOON 2015) was held during August 4-6, 2015, in Beijing, China. COCOON 2015 provided a forum for researchers working in the area of theoretical computer science and combinatorics.

The technical program of the conference included 49 contributed regular papers selected by the Program Committee from full submissions received in response to the call for papers. The accepting rate is 48%. In addition, to increase opportunities for exchange of research ideas, the conference also accepted 11 shorter papers. All the papers were peer reviewed by at least three Program Committee members or external reviewers. The papers cover various topics, including algorithms and data structures, algorithmic game theory, approximation algorithms and online algorithms, automata, languages, logic, and computability, complexity theory, computational learning theory, cryptography, reliability and security, database theory, computational biology and bioinformatics, computational algebra, geometry, number theory, graph drawing and information visualization, graph theory, communication networks, optimization, and parallel and distributed computing. Some of the papers will be selected for publication in special issues of *Algorithmica*, *Theoretical Computer Science* (TCS), and *Journal of Combinatorial Optimization* (JOCO). It is expected that the journal version papers will appear in a more complete form.

We would like to thank the Program Committee members and external reviewers for volunteering their time to review conference papers. We would like to extend special thanks to the publication, publicity, and local organization chairs for their hard work in making COCOON 2015 a successful event. Last but not least, we would like to thank all the authors for presenting their works at the conference.

August 2015

Dachuan Xu
Donglei Du
Dingzhu Du

Conference Organization

Program Chairs

Dingzhu Du	University of Texa at Dallas, USA
Donglei Du	University of New Brunswick, Canada
Dachuan Xu	Beijing University of Technology, China

Publication Chairs

Chenchen Wu	Tianjin University of Technology, China
Fengmin Wang	Beijing University of Technology, China

Publicity Chairs

Gaidi Li	Beijing University of Technology, China
Xuegang Chen	North China Electric Power University
Jianfeng Ren	Qufu Normal University, China

Local Organization Chairs

Dachuan Xu	Beijing University of Technology, China
Xianyuan Zhao	Beijing University of Technology, China

Program Committee

Hee-Kap Ahn	Pohang University of Science and Technology, Korea
Yossi Azar	Tel-Aviv University, Israel
Vladimir Braverman	Jonhs Hopkins University, USA
Zhipeng Cai	Georgia State University, USA
Yixin Cao	Hungarian Academy of Sciences, Hungary
Xi Chen	Columbia University, USA
Zhixiang Chen	University of Texas Pan American, USA
Janos Csirik	University of Szeged, Hungary
Dingzhu Du	University of Texas at Dallas, USA
Donglei Du	University of New Brunswick, Canada
Zachary Friggstad	University of Alberta, Canada
Xiaodong Hu	Chinese Academy of Sciences, China
Tsan-Sheng Hsu	Academia Sinica, Taiwan
Klaus Jansen	University of Kiel, Germany
Iyad Kanj	DePaul University, USA
Ming-Yang Kao	Northwestern University, USA

Donghyun Kim	North Carolina Central University, USA
Piotr Krysta	University of Liverpool, UK
Minming Li	City University of Hong Kong, SAR China
Guohui Lin	University of Alberta, Canada
Julian Mestre	University of Sydney, Australia
Rolf Moehring	Technische Universität Berlin, Germany
Benjamin Moseley	Toyota Technological Institute, Japan
Mitsunori Ogihara	University of Miami, USA
Desh Ranjan	Old Dominion University, USA
Yilin Shen	Samsung Research America, USA
Takeshi Tokuyama	Tohoku University, Japan
Marc Uetz	University of Twente, The Netherlands
Dachuan Xu	Beijing University of Technology, China
Jinhui Xu	State University of New York at Buffalo, USA

Additional Reviewers

Aaronson, Scott
Ahmad, Yanif
Anastasiadis, Eleftherios
Ateniese, Giuseppe
Aydinlioglu, Baris
Bei, Liu
Beisegel, Jesse
Bienkowski, Marcin
Biswas, Abhishek
Bonsma, Paul
Bouland, Adam
Broersma, Hajo
Buchbinder, Niv
Buffett, Scott
Burns, Randal
Byrka, Jarek
Chen, Danyang
Chen, Xujin
Chen, Zihe
Chestnut, Stephen
Cohen, Ilan
Cui, Lei
Dasari, Naga Shailaja
Eden, Alon
Epstein, Leah
Feldman, Michal
Fischer, Felix
Fotakis, Dimitris
Ghorbani, Ali
Gudmundsson, Joachim
Guo, Linke
Han, Xin
Huang, Ziyun
Hwang, Yoonho
Ivkin, Nikita
Jin, Yong
Kaluza, Maren
Kelly, Terence
Kim, Hyunbum
Kim, Min-Gyu
Klavzar, Sandi
Klein, Kim-Manuel
Klimm, Max
Köhler, Ekkehard
Kononov, Alexander
Korman, Matias
Kraft, Stefan
Kwok, Tsz Chiu
Land, Felix
Land, Kati
Levin, Keith
Li,Wenjun
Li, Yuchao
Liang, Dongyue

Liu, Xianliang
Liu, Yangwei
Liu, Zaoxing
Maack, Marten
Mahalanobis, Ayan
Mikkelsen, Jesper W.
Nadeem, Tamer
Nguyen, Nam P.
Nikhil Bansal
Obenshain, Daniel
Oh, Eunjin
Ono, Hirotaka
Pal, Marcin
Papamichail, Dimitris
Park, Dongwoo
Pluhár, András
Poon, Chung Keung
Pruhs, Kirk
Rahman, Md. Saidur
Rau, Malin
Rosenberg, Burton
Roytman, Alan
Sanders, Peter
Son, Junggab
Son,Wanbin
Song,Wei
Tong, Guanmo
Tu, Jianhua
Uma, RN
Vardi, Adi
Vasiliev, Saveliy
Vinar, Tomas
Vorsanger, Greg
Wang,Wei
Wang, Xiangyu
Wang, Yishui
Wang, Yu-Shuen
Wang, Zhenbo
Watrigant, Rémi
Wolff, Alexander
Wong, Prudence
Wu, Chenchen
Wu, Lidong
Wu, Weiwei
Xu, Yicheng
Yamanaka, Katsuhisa
Yang, Lin
Yoon, Sangduk
You, Jie
Yuan, Jing
Yuan, Jinjiang
Zhang, Jialin
Zhang, Jinshan
Zhang, Qiang
Zhao, Liang
Zhao, Yingchao
Zhong, Jiaofei
Zhou, Nanrun
Zhu, Yuqing

Contents

Graph Algorithms I

Mining Preserving Structures in a Graph Sequence

Takeaki Uno[1] and Yushi Uno[2](✉)

[1] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp

[2] Graduate School of Science, Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai 599-8531, Japan
uno@mi.s.osakafu-u.ac.jp

Abstract. In the recent research of data mining, frequent structures in a sequence of graphs have been studied intensively, and one of the main concern is changing structures along a sequence of graphs that can capture dynamic properties of data. On the contrary, we newly focus on "preserving structures" in a graph sequence that satisfy a given property for a certain period, and mining such structures is studied. We bring up two structures of practical importance, a connected vertex subset and a clique that exist for a certain period. We consider the problem of enumerating these structures and present polynomial delay algorithms for the problems. Their running time may depend on the size of the representation, however, if each edge has at most one time interval in the representation, the running time is $O(|V||E|^3)$ for connected vertex subsets and $O(\min\{\Delta^5, |E|^2\Delta\})$ for cliques, where the input graph is $G = (V, E)$ with maximum degree Δ. To the best of our knowledge, this is the first systematic approach to the treatment of this notion, namely, preserving structures.

1 Introduction

Extracting useful information from *graph structured data* has become important in the era of explosive and complex data, and it is often achieved by specifying and/or finding frequent substructures in a graph, that is, *pattern mining* in graphs (or *graph mining*) [1,11,22]. In the case of hyperlink structure on the Web (i.e., the webgraph), for example, a clique is considered to be formed by a community, and finding it may be useful for tracing a social phenomenon on the Web [21]. These observations imply that one of the most promising approaches for graph mining is by *enumeration*, and efficient enumeration of crucial substructures has a rich history. For cliques, a theoretically efficient algorithm is presented in [15], and both [15] and [20] are state-of-the-art algorithms that performs well in practice. Enumerations of paths and matchings are studied in [17]

A part of this research is supported by JST CREST and Grant-in-Aid for Scientific Research (KAKENHI), No. 23500022 and 15H00853.

and [8], respectively, and enumeration of connected components is studied in [2]. Here, notice that all these algorithms work on a single (and thus "static") graph.

In a recent and practical situation, however, it is often the case that graph structures may change over time (i.e., "dynamic"), and such data is collected periodically along a time series. In this setting, graph patterns appearing sequentially could be more important. Along this direction, there are some topics of interest so far. Finding graph patterns that appear periodically in a graph sequence is studied in [9,13]. Graph patterns frequently appear during a certain period are also studied in [6]. On the other hand, some research address the change patterns that appear frequently in a graph sequence composed of graphs with edge insertions/deletions, such as changes between two time periods [3] and changes of subsequences [10]. Furthermore, there are several studies focusing on clustering of vertices by utilizing graph sequences [18,19]. However, these research only concern with the "changes" or their frequency and periodicity.

Objective. Taking these preceding research into account, we propose a new concept of graph mining; finding a part of a graph that satisfies a given property continuously for a long time in a series of dynamically changing graphs, that is, *capturing invariables in change*. More specifically, we consider the problem of enumerating all substructures that satisfy a given property during a prescribed period, i.e., those appearing in a consecutive subsequence of a graph sequence. We call such structures *preserving structures* in a graph sequence, and the problem for enumerating all such structures *preserving structure mining* in general. We consider connected vertex subsets and cliques for such properties. For example, a community on the Web that is active for a long time may correspond to a clique that exists in a consecutive sequence of webgraphs during a certain period. As another example, a group of a species in a wildlife environment may constitute a consecutive sequence of connected vertex subsets in a sequence of graphs that are constructed from its trajectory data [4,12,14]. To the best of our knowledge, this study is the first case in which a "long-lasting" or "unchanging" structure is regarded as the target structure to be captured.

Contributions. In this paper, we first propose a new concept, that is, a preserving structure in a graph sequence. By adopting this notion, we pose two problems of mining preserving structures of practical importance: cliques and connected vertex subsets.

We then propose efficient algorithms for solving the problems by enumerating all connected vertex subsets or cliques for a certain time period in a given graph sequence. For this purpose, we define a way of representing a graph sequence as the input format. In this model, instead of representing a graph at each time by the difference from the previous one which is used in the dynamic graph model [7], we represent a graph sequence by explicitly associating each edge with its time interval(s) during which it exists. Although there exist similar ways of representing such data (e.g., [5]), our model is new in the sense that it introduces a new parameter, namely the number of time intervals, which will be used to estimate the running time of these algorithms. That is, it would be used as a new measure in the complexity study.

Our enumeration algorithms for preserving connected vertex subsets is based on a recursive graph partition and for preserving cliques is based on the reverse search, which is a framework for efficient enumeration algorithm design. While a straightforward application of maximal clique enumeration may require a long delay per output, our algorithms achieve polynomial delay by exploiting properties of the time intervals of edges. Compared to a naive algorithm, this reduces the time complexity with a factor of the number of edges of an input graph. Although our problem setting is fundamendal, it gives a new perspectives for graphs that change over time, together with a way of data representations and analysis of algorithms, which will pioneer a new research field.

Organization of the Paper. We give definitions and representations of graph sequences and preserving structures together with basic terminology in Sect. 2. In Sect. 3, we deal with the enumeration problem of preserving connected vertex subsets. Then we discuss about the preserving clique enumeration problem in Sect. 4. We conclude this paper in Sect. 5.

2 Preliminaries

2.1 A Graph Sequence and Its Representation

A *graph* G is an ordered pair of a *vertex set* V and an *edge set* E, and is denoted by $G = (V, E)$. We assume that a vertex set V is $\{1, \ldots, n\}$ so that each vertex has an index and can be treated as an integer. The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u \in V \mid \{u, v\} \in E\}$. The *degree* of a vertex v is $|N(v)|$, and is denoted by $\deg(v)$. We use Δ to denote the maximum degree of a graph. For a vertex subset U $(\subseteq V)$, the *induced subgraph* $G[U]$ of G by U is the subgraph whose vertex set is U and edge set is composed of all edges in E that connect vertices in U. For an edge set F, let $V(F)$ denote the set of vertices that are endpoints of some edges in F. Then for an edge subset F $(\subseteq E)$, we define the induced subgraph $G[F]$ of G by F by the subgraph $G[V(F)]$.

A *time stamp* is an integer representing a discrete time, and we denote by $\mathcal{T}$ the ground set of all possible time stamps during which our graph is supposed to exist. We assume $\mathcal{T} = \{1, \ldots, t_{\max}\}$ without loss of generality, and a subset T of $\mathcal{T}$ is called a *time stamp set*. We say that an edge of a graph is *active* at time stamp t if it exists at that moment. The edge set E of a supposed graph consists only of edges that are active at some time stamps. To represent a graph sequence, we associate a time stamp set with each edge on which it is active. We call it an *active time stamp set* of that edge, and is defined by a mapping $\tau : E \to 2^{\mathcal{T}}$. Then we define a *graph sequence* as a pair of a graph $G = (V, E)$ and a mapping τ, that is, (G, τ). Now the active time stamp set of an edge e is $\tau(e)$, and we define the active time stamp set of an edge set F to be $\tau(F) = \bigcap_{e \in F} \tau(e)$. Given a graph sequence (G, τ), we define a *closure graph* G_T of G for a time stamp set T $(\subseteq \mathcal{T})$ as the spanning subgraph in which its edge set consists of edges whose active time stamp sets includes T, that is, $G_T = (V, \{e \mid e \in E, T \subseteq \tau(e)\})$. Especially in case of $T = \{t\}$, a singleton, we sometimes denote the closure graph for T by G_t by convention. Intuitively, G_t represents a snapshot of G at time stamp t. By definition, G_T becomes G if $T = \emptyset$.

A time stamp set is an (*time*) *interval* if it constitutes a single interval $\{t, t+1, \ldots, t+\ell\}$ ($\ell \geq 0$). In this paper, it is sometimes assumed that the active time stamp set of any edge is an interval, and we call this an *interval assumption*. Note that we can assume this without loss of generality, since if an active time stamp set of an edge is composed of multiple time intervals, we can replace it by a set of parallel edges each of which has one of those intervals, respectively. Unlike the existing ones, this way of representing a graph sequence has an advantage in its extendability.

2.2 Preserving Structures

Let (G, τ) be a graph sequence, where $G = (V, E)$ and $\tau : E \to 2^{\mathcal{T}}$ with a ground time stamp set $\mathcal{T}$. We consider preserving structures in a graph sequence, that is, a subgraph that consecutively satisfies certain properties, such as connected vertex subsets and cliques in this paper. Especially, we are interested in maximal ones in some sense, and we use the term "closed" which is usually employed in the pattern mining field [16]; a closed pattern is a maximal pattern that is not included in the other patterns with the same frequency.

A vertex subset U is *connected* if there exists a path between any two vertices of U. In this case we also say that $G[U]$ is connected. A vertex subset U is said to be *connected on a time stamp set* T if U is connected at any time stamp in T. Let $\gamma(U)$ be the set of time stamps at which U is connected. We say that a connected vertex subset U is *closed* if none of its superset U' satisfies $\gamma(U) = \gamma(U')$.

A *clique* is a complete subgraph of a graph. In this paper, we define a clique by its edge set, and thus we do not regard a single vertex as a clique. A clique is called *maximal* if none of its superset becomes a clique. An edge set F is called *active* if $\tau(F) \neq \emptyset$, and $\tau(F)$ equals $\mathcal{T}$ if $F = \emptyset$. An active clique K in a graph sequence is *closed* if no other clique K' such that $K \subset K'$ satisfies $\tau(K) = \tau(K')$.

3 Enumeration of Preserving Connected Vertex Subsets

In this section we study the closed connected vertex subsets in a graph sequence (G, τ), where $G = (V, E)$ and $\tau : E \to 2^{\mathcal{T}}$ with a ground time stamp set $\mathcal{T}$. We start by observing their properties, and then present how they are enumerated.

We first have the following simple observations.

Property 1 (closed under union). For two vertex subsets U and U', if both U and U' are connected on a time stamp set T and $U \cap U' \neq \emptyset$, then $U \cup U'$ is also connected on T.

For two partitions $\mathcal{P}$ and $\mathcal{P}'$ of a universal set, let $\mathcal{P} \wedge \mathcal{P}'$ denote the partition composed of subsets given by the intersection of members of $\mathcal{P}$ and $\mathcal{P}'$, i.e., $\mathcal{P} \wedge \mathcal{P}' = \{I \mid I = H \cap H', H \in \mathcal{P}, H' \in \mathcal{P}'\}$. A *connected component* of G is a maximal vertex subset U such that $G[U]$ is connected. The set of connected components of G gives a partition of the vertex set, and we denoted it by $\mathcal{C}(G)$. For a time stamp set $T = \{t_{i_1}, \ldots, t_{i_k}\}$, let $\mathcal{P}(G, T)$ denote $\bigwedge_{j=1}^{k} \mathcal{C}(G_{t_{i_j}})$, which forms a partition of V.

Property 2 (partition). A connected vertex subset U on a time stamp set T is included in one of vertex subsets of $\mathcal{P}(G, T)$.

Property 3 (subdivision). A connected vertex subset U in W on a time stamp set T is contained in a vertex subset of $\mathcal{P}(G[W], T)$.

We denote the family of all maximal connected vertex subsets of G on a time stamp set T by $\mathcal{C}(G, T)$. Property 1 ensures that $\mathcal{C}(G, T)$ becomes a partition of V. In the subsequent discussions in this subsection, suppose the interval assumption holds for T, and let $T_{t,\ell}$ denote an interval time stamp set $T_{t,\ell} = \{t, t+1, \ldots, t+\ell\}$. In addition, we assume for simplicity that both ends of any interval time stamp set $T_{t,\ell}$ can be examined in $O(1)$ time by some appropriate pre-process and data structures. Then we have the following two lemmas.

Lemma 1. *For an interval time stamp set $T_{t,\ell}$ with a fixed time stamp t, $\mathcal{C}(G, T_{t,\ell})$ for all ℓ (≥ 0) can be computed in $O(|V||E|^2)$ time.*

Lemma 2. *Any member U in $\mathcal{C}(G, T)$ is a closed connected vertex subset of G on an interval time stamp set T.*

Lemma 2 motivates us to compute $\mathcal{C}(G, T)$ for all possible interval time stamp set T to enumerate all closed connected vertex subsets. For each time stamp t, we compute $\mathcal{C}(G, T_{t,\ell})$ for interval time stamp set $T = \{t, t+1, \ldots, t+\ell\}$ for all possible ℓ. From Lemma 1, this computation can be done in $O(|V||E|^2)$ time. Thus we obtain the following theorem, where we use $\ell = O(|E|)$ again.

Theorem 1. *In a graph sequence (G, τ), all closed connected vertex subsets can be enumerated in $O(|V||E|^3)$ time.* □

The correctness of this algorithm relies only on the above three properties, therefore the algorithm can be applied to similar connectivity conditions satisfying these properties, such as strong connectivity of a directed graph and two-edge connectivity of a graph.

Theorem 2. *In a graph sequence (G, τ) in which G is a directed graph, all closed strongly connected vertex subsets can be enumerated in $O(|V||E|^3)$ time.* □

Theorem 3. *In a graph sequence (G, τ), all closed two-edge connected vertex subsets in a graph can be enumerated in $O(|V||E|^3)$ time.* □

In the case of two-vertex connectivity, Property 1 holds only when the intersection size of two components is no less than two. Thus, $\mathcal{C}(G, T)$ could not be a partition of a vertex set. Instead of a vertex set, we represent a connected vertex subset by all vertex pairs included in the subset. Using this representation, when two subsets share at most one vertex, the intersection of their representations is the empty set. Obviously this representation satisfies the other two properties, thus we have the following theorem.

Theorem 4. *In a graph sequence (G, τ), all closed two-vertex connected vertex subsets can be enumerated in $O(|V|^2|E|^3)$ time.* □

4 Enumeration of Closed Active Cliques

This section discusses about the enumeration of all closed active cliques in a graph sequence (G, τ). We first give some additional definitions for further arguments and observe some basic properties of closed active cliques. After that we state a simple output polynomial time algorithm as a warm-up, and then we present a more efficient algorithm based on the reverse search whose time complexity is much smaller than the simple algorithm.

For a time stamp set T, let $N_T(v) = \{w \mid w \in N(v), T \subseteq \tau(\{v, w\})\}$ and $N_T(F) = \bigcap_{v \in V(F)} N_T(v)$ for an edge set F, that is, $N_T(v)$ is the set of vertices adjacent to v at all time stamps in T and $N_T(F)$ is the set of vertices adjacent to *all* vertices in $V(F)$ at any time stamp in T. For an edge set F and a vertex set U, $F \setminus U$ denotes the edge set obtained from F by removing all edges incident to some vertices in U, and $F \cap U$ denotes $F \setminus (V \setminus U)$. For an edge set F and a vertex v, let $M(F, v)$ denote the set of edges connecting v and a vertex in $V(F)$. Let $\Gamma(F)$ be the set of vertices v such that $\tau(F) \subseteq \tau(M(F, v))$.

Now let $F_{\leq i}$ be the edge set obtained from F by removing edges incident to vertices whose index is greater than i. By definition, $F_{\leq i}$ is empty if $i < 1$, and is F if $i \geq n$. A *lexicographic order* on a family of sets is a total order defined in such a way that a set F is smaller than F' when the smallest element in their symmetric difference $F \triangle F'$ belongs to F. For an active clique K in a graph sequence, let $X(K)$ denote the lexicographically smallest closed clique including K among all closed cliques K' such that $\tau(K') = \tau(K)$.

4.1 A Simple Algorithm

Let (G, τ) be a graph sequence, where $G = (V, E)$ and $\tau : E \to 2^{\mathcal{T}}$ with a ground time stamp set $\mathcal{T}$. We observe a few basic properties of closed active cliques in a graph sequence. Remember that a clique is defined by an edge set in this paper.

Lemma 3. *For any active clique K, $X(K)$ can be computed in $O(\min\{|E|, \Delta^2\})$ time.*

Proof. We can obtain $X(K)$ by iteratively choosing the minimum vertex v in $N_{\tau(K)}(K)$ and adding edges of $M(K, v)$ to K, until $N_{\tau(K)}(K) = \emptyset$. $N_{\tau(K)}(K)$ can be computed in $O(\min\{|E|, \Delta^2\})$ time by scanning all edges adjacent to some edges in K. When we add $N_{\tau(K)}(K)$ to K, $N_{\tau(K)}(K \cup N_{\tau(K)}(K))$ can be computed in $O(\deg(v))$ time by checking whether $\tau(K) \subseteq \tau(\{u, v\})$ or not for each $u \in N_{\tau(K)}(K)$. Therefore the statement holds. □

Lemma 4. *For any time stamp set T, any maximal clique K in G_T is closed.*

Proof. If K is not closed, $G_{\tau(K)}$ includes a clique K' such that $K \subset K'$. Since $T \subseteq \tau(K)$, $T \subseteq \tau(e)$ holds for any edge $e \in K'$. This implies that K' is a clique in G_T, which contradicts the assumption. □

Conversely, we can easily see that any closed active clique K is maximal in the graph $G_{\tau(K)}$. This motivates us to compute all maximal cliques in all closure graphs of possible active time stamp sets for enumerating all closed active cliques.

Lemma 5. *All closed active cliques can be enumerated in $O(|V||E|^3)$ time for each, under the interval assumption.*

Proof. Under the interval assumption, the active time stamp set of any closed active clique is also an interval. These active time stamp sets satisfy that the both ends of the interval are given by the active time sets of some edges, thus their number is bounded by $|E|^2$. Let $\mathcal{K}$ be the family of cliques each of which is a maximal clique in a closure graph of some of those active time stamp sets. Then, from Lemma 4, we can see that $|\mathcal{K}|$ is bounded by the product of $|E|^2$ and the number of closed active cliques. By using the algorithm in [15], the maximal cliques can be enumerated in $O(|V| + |E|)$ time for each, and thus the maximal cliques in $\mathcal{K}$ can be enumerated in $O((|V| + |E|)|\mathcal{K}|)$ time. To check whether an enumerated clique K is closed or not, we compute $X(K)$ in $O(|V| + |E|)$ time. Since a closed active clique can be a maximal clique of G_T for at most $|E|^2$ time stamp sets T, the closed active cliques can be enumerated in $O(|V||E|^3)$ time for each. □

4.2 An Efficient Algorithm Based on the Reverse Search

The reverse search is a scheme for constructing enumeration algorithms, and was originally proposed by Avis and Fukuda [2] for some problems such as enumeration of vertices of a polytope. The key idea of the reverse search is to define an acyclic relation among the objects including the ones to be enumerated. An acyclic relation induces a tree, which results in the so-called a parent-child relation, and we call the tree a *family tree.* Hence enumerating objects is realized by traversing the tree according to the parent-child relation to visit all the objects. In fact, the reverse search algorithm performs a depth-first search on the tree induced by the parent-child relation, and is implemented by a procedure for enumerating all children of a given object. It starts from the root object that has no parent and enumerates its children, and then it recursively enumerates children for each child.

It is easy to see the correctness of the algorithm; that is, the tree induced by the parent-child relation spans all the objects, and the algorithm visits all the vertices of the tree by a depth-first search. When a procedure for enumerating children takes at most $O(A)$ time for each child, the computation time of the reverse search algorithm is bounded by $O(AN)$, where N is the number of objects to be enumerated. Hence, if A is polynomial in terms of the input size, the entire reverse search algorithm takes output polynomial time. In the following, we carefully observe the properties of a graph sequence, and prove that enumeration of children can be done in polynomial time.

Now a more efficient algorithm for enumeration of closed active cliques can be designed based on the reverse search. We start with giving some definitions and fundamental observations. The scheme of the reverse search has already been applied to enumeration maximal cliques [15], and our algorithm for closed active cliques adopts their ideas. For an active clique K, let $i(K)$ be the minimum vertex i satisfying $X(K_{\leq i}) = K$. We define the parent $P(K)$ of closed active clique K by $X(K_{\leq i(K)-1})$, and $P(K)$ is not defined for $K = X(\emptyset)$, which is called the *root* of the family tree.

Lemma 6. *The parent-child relation defined by P is acyclic.*

Proof. Suppose that K is a closed active clique such that $P(K)$ is defined. $P(K)$ is generated by removing vertices one by one from K, and adding vertices so that the active time set does not change, thus $\tau(P(K))$ always includes $\tau(K)$. Since $X(K_{\leq i(K)-1}) \neq K$, $P(K)$ is lexicographically smaller than K when $\tau(P(K)) = \tau(K)$. Thus, either (a) $P(K)$ has a larger active time set than K, or (b) $P(K)$ has the same active time set as K and is lexicographically smaller than K. Therefore the statement holds. □

Lemma 7. *Any vertex in $P(K)_{\leq i(K)} \setminus K$ does not belong to $N_{\tau(K)}(i(K))$, and therefore $K_{\leq i(K)-1} = P(K)_{\leq i(K)} \cap N_{\tau(K)}(i(K))$.*

Proof. Suppose that a vertex v in $P(K)_{\leq i(K)} \setminus K$ belongs to $N_{\tau(K)}(i(K))$. Then, $X(K_{\leq i(K)})$ has to include either v or another vertex $u < v$. It implies that $X(K_{\leq i(K)}) \cap \{1, \ldots, i(K)\} \neq K_{\leq i(K)}$, thereby $X(K_{\leq i(K)}) \neq K$. This contradicts the definition of $i(K)$. □

A subset F of $M(K, v)$ is called *time maximal* if F is included in no other subset F' of $M(K, v)$ satisfying $\tau(F) \cap \tau(K) = \tau(F') \cap \tau(K)$. Let $I(K, v)$ be the set of all time maximal subsets of $M(K, v)$. For a time maximal subset $F \in I(K, v)$, we define $C(K, F) = X(K_{\leq v} \cap V(F) \cup F)$.

Lemma 8. *If K' is a child of non-root closed active clique K, then $K' = C(K, F)$ holds for some vertex v and $F \in I(K, i(K'))$.*

Proof. Let $F = M(K_{\leq i(K')}, i(K'))$. From Lemma 7, $K'_{\leq i(K')-1} = K_{\leq i(K')-1} \cap V(N_{\tau(K')}(i(K')))$ holds, and thus $K = X(K_{\leq i(K')-1} \cap V(F) \cup F)$. We next show that F is a member of $I(K, i(K'))$. Suppose that K' is a child of K, and F does not belong to $I(K, i(K'))$, i.e., F is properly included in an edge subset $F' \in I(K, i(K'))$ such that $\tau(F) = \tau(F')$. Then, the active time set of $K_{\leq i(K')} \cap V(F')$ is same as that of $K'_{\leq i(K')} = K_{\leq i(K')-1} \cap V(F) \cup F$. This implies that $X(K'_{\leq i(K')})$ includes several edges in F', which contradicts the definition of $i(K')$. □

Since $X(K_{\leq v}) \neq K$ holds for any $v < i(K)$, we have the following corollary.

Corollary 1. *$C(K, F)$ is not a child of K for any $F \in I(K, v)$ satisfying $v < i(K)$.*

It is true that any child is $C(K, F)$ for some F. However, $C(K, F)$ cannot always be a child, that is, $C(K, F)$ is a child of K if and only if $P(K) = P(C(K, F))$. This implies that we can check whether $C(K, F)$ is a child or not by computing $P(K)$. Therefore, from Lemma 8, we obtain the following procedure to enumerate children of K. For avoiding the duplicated output of the same child K', we output K' only when K' is generated from $F \in I(K, i(K'))$.

Procedure. EnumChildren(K: non-root closed active clique)

```
1. for each  $F \in I(K, v), v > i(K)$ do
2.     compute $C(K, F)$;
3.     compute $i(C(K, F))$ and $P(C(K, F))$;
4.     if  $K = P(C(K, F))$ and $i(C(K, F)) = v$ then output $C(K, F)$;
5. end for
```

For analyzing the complexity of this procedure, which will later be used as a subroutine of the entire algorithm for enumerating closed active cliques, we show some technical lemmas.

Lemma 9. *$P(K)$ can be computed in $O(|E|)$ time.*

Proof. Suppose that K is not the root, i.e., $P(K)$ is defined. Let K' be initialized to the empty set, and we add vertices of K to K' one by one from the smallest vertices in the increasing order. In each addition, we maintain the change of $\tau(K')$ and $N_{\tau(K)}(K')$. Then, we can find the minimum vertex v satisfying $\tau(K_{\leq v}) = \tau(K)$, and the minimum vertex u satisfying $i = \min\{N_{\tau(K)}(K_{\leq i-1})\}$ for any $i \in K, i \geq u$. We have $i(K) = \max\{u, v\}$, since $X(K_{\leq j}) \neq K$ holds when either $\tau(K) \neq \tau(K_{\leq j})$ or $i \neq \min\{N_{\tau(K)}(K_{\leq i-1})\}$ holds for some $i \in K, i > j$. Under the assumption that both ends of any interval time stamp set can be examined in $O(1)$ time, $\tau(K \cup \{e\})$ can be computed in $O(1)$ time from $\tau(K)$ for any edge e. Thus, we can compute $i(K)$ in $O(\min\{|E|, \Delta^2\})$ time. Together with Lemma 3, the statement holds. □

Lemma 10. *If K is not the root, any child K' of K satisfies that $K_{\leq i(K')} \cap K'_{\leq i(K')} \neq \emptyset$.*

Proof. If $K_{\leq i(K')} \cap K'_{\leq i(K')} = \emptyset$, it holds that $K'_{\leq i(K')-1} \cap K = \emptyset$. Since $K'_{\leq i(K')-1}$ is always included in K, we have $K'_{\leq i(K')-1} = \emptyset$. Therefore, $P(K') = X(\emptyset)$, which implies that $P(K)$ is the root. □

Lemma 11. *If K is not the root, the children of K is enumerated by evaluating at most $\min\{\Delta|E|, \Delta^3\}$ edge sets under the interval assumption.*

Proof. By the interval assumption, the ends of the active time set of any subset F of $I(K, v)$ is given by the ends of some edges in F, and thus $|I(K, v)|$ is bounded from above by Δ^2. Lemma 10 ensures that if K is not the root, Step 2 of EnumChildren does not have to take care of vertices not adjacent to any vertex of $V(K)$. This means that we have to take care only of non-empty maximal subset in $I(K, v)$. Let I be the union of all non-empty subsets of $I(K, v)$. Since each edge in $F \in I(K, v)$ is incident to some vertices in K, we have $|I| \leq \min\{|E|, \Delta^2\}$. It implies that the number of possible choices of two edges from some non-empty $I(K, v)$ is bounded from above by $\Delta \cdot \min\{|E|, \Delta^2\}$. □

By the above lemmas, we can estimate the time complexity of the procedure of enumerating children.

Lemma 12. *Procedure EnumChildren enumerates all children of K in $O(\min\{\Delta^5, |E|^2\Delta\})$ time under the interval assumption.*

Proof. The correctness of the procedure comes from Lemma 8. We note that the procedure never output any child more than once, since each child is generated from its unique parent, a maximal subset included in $F \in I(K, i(K))$. We then observe that all non-empty subset $F \in I(K, v), v > i(K)$ can be computed in $O(\min\{|E|, \Delta^2\})$ time by scanning all edges adjacent to some edges in K, and $C(K, F)$ can be computed in $O(\min\{|E|, \Delta^2\})$ time in a straightforward manner. From Lemma 11, the procedure iterates the loop for $\min\{\Delta|E|, \Delta^3\}$ edge sets, and each edge set spends $O(\min\{|E|, \Delta^2\})$ time from Lemma 9. Thus, we conclude the lemma. □

Now we describe our algorithm for enumerating all closed active cliques in a graph sequence based on the reverse search as follows. It is presented in a slightly different form by introducing a threshold σ with respect to the length of active time stamp sets by observing that $\tau(K) \subseteq \tau(P(K))$ always holds. It enumerates all closed active cliques having active time sets larger than σ by giving $X(\emptyset)$ (thus enumerates all when σ is set to be 0).

```
Algorithm. EnumClosedActiveClique($K$: closed active clique)
 1. output  $K$; $prv := nil$;
 2. if $prv = nil$ then $K' :=$ the first clique found by EnumChildren($K$);
     else $K' :=$ the clique found just after $prv$ by EnumChildren($K$);
 3. if there is no such clique $K'$ go to Step 8;
 4. $K := K'$; free up the memory for $K'$;
 5. if $|P(K)| \geq \sigma$ then call EnumClosedActiveClique($K$);
 6. $K := P(K)$;
 7. go to Step 2;
 8. if $K$ is not the root then return;
 9. for each $e \in E$ do
10.    if $e$ is lexicographically minimum in $X(e)$
                              then EnumClosedActiveClique($X(e)$);
11. end for
```

Finally, we can establish the following theorem.

Theorem 5. *Under the interval assumption, Algorithm EnumClosedActiveClique enumerates all closed active cliques in a graph sequence in $O(N \min\{\Delta^5, |E|^2\Delta\})$ time and in $O(|V| + |E|)$ space, where N is the number of closed active cliques in a graph sequence.*

Proof. The correctness of the algorithm is easy to see from the framework of the reverse search and Lemma 6. The computation time of the reverse search is given by the product of the number of objects to be enumerated and the computation time on each object. From Lemma 12, an iteration requires $O(N \min\{\varDelta^5, |E|^2\varDelta\})$ time for non-root closed active cliques. For the root $K = X(\emptyset)$, we can enumerate its children K' satisfying the condition of Lemma 10 in $O(N \min\{\varDelta^5, |E|^2\varDelta\})$ time using procedure EnumChildren. When $K_{\le i(K')} \cap K'_{\le i(K')} = \emptyset$, we have $K'_{\le i(K')-1} \cap K = \emptyset$. This implies that $K'_{\le i(K')}$ is composed of an edge, thus by generating $X(\{e\})$ for all $e \in E$, we can enumerate the children that do not satisfy the condition of Lemma 10, in $O(\min\{|E|^2, |E|\varDelta^2\})$ time. Note that the duplication can be avoided by outputting $X(\{e\})$ only when $e = \arg\min X(\{e\})$. Since $N \ge |E|/\varDelta^2$, it holds that $\min\{|E|^2, |E|\varDelta^2\} \le N \min\{\varDelta^5, |E|^2\varDelta\}$. Therefore the time complexity of the algorithm is as stated.

In a straightforward implementation of the algorithm, each iteration may take $\omega(|V|+|E|)$ space for keeping the intermediate results of the computation in memory, especially for all $C(K, F)$. We can reduce this by restarting the iteration from the beginning. When we find a child K' of K, we immediately generate the recursive call with K', before the termination of the enumeration of the children. After the termination of the recursive call, we resume the enumeration of the children. To save the memory, we restart from the beginning of the iteration, and we pass through the children found before K', and reconstruct all the necessary variables. We note that the time complexity does not change by the restart, since the number of restarts is bounded by the number of recursive calls generated by the algorithm. A child is given by a maximal edge subset, and a maximal edge subset is given by two edges. Thus, we can memorize a child by a constant number of variables. The clique K is constructed by computing $P(K')$, thus it is also not necessary to have K in memory, and can be re-constructed without increasing the time complexity. The iteration with respect to the root takes $O(|V| + |E|)$ space, therefore we have the atatement of the theorem. □

As we stated, since $\tau(K) \subseteq \tau(P(K))$ always holds, we have the following corollary.

Corollary 2. *Under the interval assumption, Algorithm EnumClosedActiveClique enumerates all closed active cliques having active time sets no shorter than a given threshold* σ *in* $O(\min\{\varDelta^5, |E|^2\varDelta\})$ *time for each and in* $O(|V|+|E|)$ *space.*

Note again that the interval assumption can be set without loss of generality, since we can replace an edge with multiple time intervals by parallel edges having a single time interval for each, in their active time stamp sets. However, this transformation increases the degrees of the vertices, thus the time complexity may increase. If we set $\varDelta$ to the maximum degree to the transformed graph, then the results hold.

5 Conclusion

In this paper, we focused on the structures preserved in a sequence of graphs continuously for a long time, which we call "preserving structures". We considered two structures, closed connected vertex subsets and closed active cliques, and proposed efficient algorithms for enumerating these structures preserved during a period no shorter than a prescribed length. An interesting future work is, of course, to develop efficient algorithms for preserving structure mining problems for other graph properties.

References

1. Arimura, H., Uno, T., Shimozono, S.: Time and space efficient discovery of maximal geometric graphs. In: Corruble, V., Takeda, M., Suzuki, E. (eds.) DS 2007. LNCS (LNAI), vol. 4755, pp. 42–55. Springer, Heidelberg (2007)
2. Avis, D., Fukuda, K.: Reverse search for enumeration. Discr. Appl. Math. **65**, 21–46 (1996)
3. Berlingerio, M., Bonchi, F., Bringmann, B., Gionis, A.: Mining graph evolution rules. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part I. LNCS, vol. 5781, pp. 115–130. Springer, Heidelberg (2009)
4. Buchin, K., Buchin, M., van Kreveld, M., Speckmann, B., Staals, F.: Trajectory grouping structure. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 219–230. Springer, Heidelberg (2013)
5. Xuan, B.B., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. Int. J. of Foundations of Computer Science **14**, 267–285 (2003)
6. Borgwardt, K.M., Kriegel, H.P., Wackersreuther, P.: Pattern mining in frequent dynamic subgraphs. In: Proc. 6th IEEE ICDM, pp. 818–822 (2006)
7. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification–A technique for speeding up dynamic graph algorithms. J. ACM **44**, 669–696 (1997)
8. Fukuda, K., Matsui, T.: Finding all the perfect matchings in bipartite graphs. Applied Mathematics Letters **7**, 15–18 (1994)
9. Han, J., Dong, G., Yin, Y.: Efficient mining of partial periodic patterns in time series database. In: Proc. 15th IEEE ICDE, pp. 106–115 (1999)
10. Inokuchi A., Washio, T.: A fast method to mine frequent subsequences from graph sequence data. In: Proc. 8th IEEE ICDM, pp. 303–312 (2008)
11. Mining graph data: A. Inokuchi T. Washio and H. Motoda. Complete mining of frequent patterns from graphs. Machine Learning **50**, 321–354 (2003)
12. Kalnis, P., Mamoulis, N., Bakiras, S.: On discovering moving clusters in spatio-temporal data. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 364–381. Springer, Heidelberg (2005)
13. Lahiri, M. Berger-Wolf, T.Y.: Mining periodic behavior in dynamic social networks. In: Proc. 8th IEEE ICDM, pp. 373–382 (2008)
14. Li, Z., Ding, B., Han, J., Kays, R.: Swarm: Mining relaxed temporal moving object clusters. In: Proc. 36th Int'l Conf. on VLDB, pp. 723–734 (2010)
15. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 260–272. Springer, Heidelberg (2004)
16. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient mining of association rules using closed itemset lattices. J. Information Systems **24**, 25–46 (1999)
17. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks **5**, 237–252 (1975)
18. Sun, J., Faloutsos, C., Papadimitriou, S., Yu, P.S.: GraphScope: Parameter-free mining of large time-evolving graphs. In: Proc. 13th ACM Int'l Conf. on KDD, pp. 687–696 (2007)

19. Tantipathananandh, C., Berger-Wolf, T.: Constant-factor approximation algorithms for identifying dynamic communities. In: Proc. 15th ACM Int'l Conf. on KDD, pp. 827–836 (2009)
20. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theor. Comp. Sci. **363**, 28–42 (2006)
21. Uno, Y., Ota, Y., Uemichi, A.: Web structure mining by isolated cliques. IEICE Transactions on Information and Systems **E90–D**, 1998–2006 (2007)
22. Yan, X., Han, J.: GSPAN: graph-based substructure pattern mining. In: Proc. 2nd IEEE ICDM, pp. 721–724 (2002)

On the Most Imbalanced Orientation of a Graph

Walid Ben-Ameur(✉), Antoine Glorieux, and José Neto

Institut Mines-Télécom, Télécom SudParis, CNRS Samovar UMR 5157,
9 Rue Charles Fourier, 91011 Evry Cedex, France
{walid.benameur,antoine.glorieux,jose.neto}@telecom-sudparis.eu

Abstract. We study the problem of orienting the edges of a graph such that the minimum over all the vertices of the absolute difference between the outdegree and the indegree of a vertex is maximized. We call this minimum *the imbalance* of the orientation, i.e. the higher it gets, the more imbalanced the orientation is. We study this problem denoted by MaxIm. We first present different characterizations of the graphs for which the optimal objective value of MaxIm is zero. Next we show that it is generally NP-complete and cannot be approximated within a ratio of $\frac{1}{2} + \varepsilon$ for any constant $\varepsilon > 0$ in polynomial time unless $\mathtt{P} = \mathtt{NP}$ even if the minimum degree of the graph δ equals 2. Finally we describe a polynomial-time approximation algorithm whose ratio is equal to $\frac{1}{2}$ for graphs where $\delta \equiv 0[4]$ or $\delta \equiv 1[4]$ and $(\frac{1}{2} - \frac{1}{\delta})$ for general graphs.

Introduction and Notation

Let $G = (V, E)$ be an undirected simple graph, we denote by δ_G the minimum degree of the vertices of G. An orientation Λ of G is an assignment of a direction to each undirected edge $\{uv\}$ in E, i.e. any function on E of the form $\Lambda(\{uv\}) \in \{uv, vu\}, \forall \{uv\} \in E$. For each vertex v of G we denote by $d_G(v)$ or $d(v)$ the unoriented degree of v in G and by $d^+_\Lambda(v)$ or $d^+(v)$ (resp. $d^-_\Lambda(v)$ or $d^-(v)$) the outdegree (resp. indegree) of v in G w.r.t. Λ. Graph orientation is a well studied area in graph theory and combinatorial optimization and thus a large variety of constrained orientations as well as objective functions have been considered so far.

Among those arise the popular degree-constrained orientation problems: in 1976, Frank & Gyárfás [12] gave a simple characterization of the existence of an orientation such that the outdgree of every vertex is between a lower and an upper bound given for each vertex. Asahiro et al. in [1–3] proved the NP-hardness of the weighted version of the problem where the maximum outdegree is minimized, gave some inapproximability results, and studied similar problems for different classes of graphs. Chrobak & Eppstein proved that for every planar graph a 3-bounded outdegree orientation and a 5-bounded outdegree acyclic orientation can be constructed in linear time [6].

Other problems involving other criterion on the orientation have been studied such as acyclicity, diameter or connectivity. Robbins' theorem (1939) for example states that the graphs that have strong orientations are exactly the 2-edge-connected graphs [18] and later (1985), Chung et al. provided a linear time

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 16–29, 2015.
DOI: 10.1007/978-3-319-21398-9_2

algorithm for checking whether a graph has such an orientation and finding one if it does [7]. Then in 1960, Nash-Williams generalized Robbin's theorem showing that an undirected graph has a k-arc-connected orientation if and only if it is $2k$-edge-connected [17]. The problem called oriented diameter that consists in finding a strongly connected orientation with minimum diameter was introduced in 1978 by Chv́atal & Thomassen: they proved that the problem is NP-hard for general graphs [8]. It was then proven to be NP-hard even if the graph is restricted to a subset of chordal graphs by Fomin et al. (2004) who gave also approximability and inapproximability results [10].

For an orientation Λ of $G = (V, E)$ and a vertex v we call $|d^+_\Lambda(v) - d^-_\Lambda(v)|$ the *imbalance* of v in G w.r.t Λ and thus we call $\min_{v\in V} |d^+_\Lambda(v) - d^-_\Lambda(v)|$ the *imbalance* of Λ. Biedl et al. studied the problem of finding an acyclic orientation of unweighted graphs minimizing the imbalance of each vertex: they proved that it is solvable in polynomial time for graphs with maximum degree at most three but NP-complete generally and for bipartite graphs with maximum degree six and gave a $\frac{13}{8}$-approximation algorithm [5]. Then Kára et al. closed the gap proving the NP-completeness for graphs with maximum degree four. Furthermore, they proved that the problem remains NP-complete for planar graphs with maximum degree four and for 5-regular graphs [14].

Landau's famous theorem [15] gives a condition for a sequence of non-negative integers to be the score sequence or outdegree sequence of some tournament (i.e. oriented complete graph) and later, Harary & Moser characterized score sequences of strongly connected tournaments [13]. Analogous results for the "imbalance sequences" of directed graphs are were given by Mubayi et al. [16]. In 1962, Ford & Fulkerson characterized the mixed graphs (i.e. partially oriented graphs) which orientation can be completed in a eulerian orientation, that is to say, an orientation for which the imbalance of each vertex equals zero [11]. Many other results related to orientation have been proposed. Some of them are reviewed in [4].

Let us denote by $\overrightarrow{O}(G)$ the set of all the orientations of G, we consider the problem of finding an orientation with maximized imbalance:

$$(\textsc{MaxIm}) \quad \textsc{MaxIm}(G) = \max_{\Lambda\in\overrightarrow{O}(G)} \min_{v\in V} |d^+_\Lambda(v) - d^-_\Lambda(v)|$$

and we call $\textsc{MaxIm}(G)$ the value of $\textsc{MaxIm}$ for G. The minimum degree δ_G of a graph G is a trivial upper bound for $\textsc{MaxIm}(G)$.

The rest of this paper is organized as follows. In the first section, we give several characterizations of the the graphs verifying $\textsc{MaxIm}(G) = 0$. In section 2, we will show that $\textsc{MaxIm}$ is generally NP-complete even for graphs with minimum degree 2 and inapproximable within a ratio $\frac{1}{2} + \varepsilon$ for any constant $\varepsilon > 0$ and then will give an approximation algorithm whose ratio is almost equal to $\frac{1}{2}$. Since the value of $\textsc{MaxIm}$ for a graph is the minimum of the values of $\textsc{MaxIm}$ on its connected component, from here on in, all the graphs we consider are assumed to be connected. For any graph G we will use the notations $V(G)$ and $E(G)$ to refer to the set of vertices of G and the set of edges of G respectively.

1 Characterizing the Graphs for which MaxIm(G) = 0

Now we ask ourselves which are the graphs verifying MaxIm(G) = 0. We will start by unveiling several necessary conditions and properties of such graphs. First we can show that concerning such a graph, we can find an orientation satisfying several additional properties.

Proposition 1. *Let G be a graph such that* MaxIm(G) = 0 *and $u \in V$. Then there exists an orientation $\Lambda \in \overrightarrow{O}(G)$ such that u is the only vertex of G with imbalance equal to zero w.r.t. Λ.*

Proof. Let $\Lambda \in \overrightarrow{O}(G)$ be an orientation minimizing $|\{v \in V / |d^+_\Lambda(v) - d^-_\Lambda(v)| = 0\}|$. We suppose that $|\{v \in V / |d^+_\Lambda(v) - d^-_\Lambda(v)| = 0\}| \geq 2$. We choose two distinct vertices v and w in $\{v \in V / |d^+_\Lambda(v) - d^-_\Lambda(v)| = 0\}$ and a path $p = (v = u_0, \cdots, u_n = w)$ between v and w. If we switch the orientation of the edge $\{u_0u_1\}$, then the imbalance of u_0 becomes positive and necessarily the imbalance of u_1 becomes zero otherwise the resulting orientation would contradict the minimality of Λ. Using the same reasoning, if we switch the orientation of all the edges $\{u_0u_1\}, \cdots, \{u_{n-2}u_{n-1}\}$, we obtain an orientation where both u_{n-1} and u_n have an imbalance equal to zero while the imbalance is positive on all the vertices $u_0, \cdots, u_{n-2}$ and unchanged on all other vertices. So now if we switch the orientation of the edge $\{u_{n-1}u_n\}$ as well, then the resulting orientation contradicts the minimality of Λ. Hence, $|\{v \in V / |d^+_\Lambda(v) - d^-_\Lambda(v)| = 0\}| = 1$.

Now let v be this unique vertex of G such that $|d^+_\Lambda(v) - d^-_\Lambda(v)| = 0$. Let $u \neq v$ be an arbitrary vertex and let $p = (v = u_0, \cdots, u_n = u)$ be a path between v and u. By switching the orientation of all the edges $\{u_0u_1\}, \cdots, \{u_{n-2}u_{n-1}\}$, we obtain an orientation Λ' where u has an imbalance equal to zero while the imbalance is positive for u_0 and unchanged on all other vertices. □

This yields the following necessary condition: if G is a graph such that MaxIm(G) = 0, then G is eulerian. For let $u \in V$, we know there exists $\Lambda \in \overrightarrow{O}(G)$ such that $\{v \in V / |d^+_\Lambda(v) - d^-_\Lambda(v)| = 0\} = \{u\}$. Then $d^+_\Lambda(u) = d^-_\Lambda(u)$, hence $d(u) = d^+_\Lambda(u) + d^-_\Lambda(u) = 2d^+_\Lambda(u)$ is even. The following lemma about eulerian graphs will prove useful for the proof of our characterization.

Lemma 2. *If G is an eulerian graph, then there exists an elementary cycle (*hereafter just called cycle*) C of G such that $G - E(C)$ has at most one connected component that is not an isolated vertex.*

Proof. Being G eulerian and connected, it can be decomposed into edge-disjoint cycles that we can order $C_1, \cdots, C_n$ according to the following condition: $\cup^i_{k=1} C_i$ is connected, $\forall i \in [\![1, n]\!]$. Then C_n is the cycle we are looking for. □

Now let us define a certain family of graphs which will prove to be exactly the graphs for which the optimal objective value of MaxIm is zero. Intuitively they are the graphs for which every block is an odd cycle.

Theorem 3. *We define the class of graphs $\mathscr{C}^{odd}$ as follows: a simple graph G is in $\mathscr{C}^{odd}$ if there exists $C_1, \cdots, C_n$ odd cycles ($n \geq 1$) such that:*

$$\begin{aligned} &\bullet\ \cup_{i=1}^{n} C_i = G, \\ &\bullet\ |V(\cup_{k=1}^{i-1} C_k) \cap V(C_i)| = 1,\ \forall i \in [\![2, n]\!]. \end{aligned} \tag{1}$$

Then for any simple graph G, $\textsc{MaxIm}(G) = 0$ *if and only if* $G \in \mathscr{C}^{odd}$.

Proof.
- $\Leftarrow$ We will work by induction on the number of cycles n contained in the graph. Nothing is required for these cycles except that they must be elementary. If $n = 1$, then our graph is an odd cycle which implies $\textsc{MaxIm}(G) = 0$. Let $n \geq 2$, we assume that all graphs of $\mathscr{C}^{odd}$ with $k \leq n-1$ cycles verify $\textsc{MaxIm}(G) = 0$. Let $G \in \mathscr{C}^{odd}$ with n cycles $C_1, \cdots, C_n$ as in (1). Suppose there exists $\Lambda \in \overrightarrow{O}(G)$ with strictly positive imbalance. Let us call $G' = \cup_{i=1}^{n-1} C_i$ the graph obtained from G after removing C_n and let us take a look at $\Lambda_{|E(G')}$ the orientation of the edges of G' obtained from Λ as its restriction on $E(G')$. As G' is a graph of $n-1$ cycles in $\mathscr{C}^{odd}$, our inductive hypothesis implies that we have a vertex $u \in V(G')$ such that $|d^+_{\Lambda_{|E(G')}}(u) - d^-_{\Lambda_{|E(G')}}(u)| = 0$. Necessarily, $u = V(G') \cap V(C_n)$. Thus $|d^+_{\Lambda}(u) - d^-_{\Lambda}(u)| = |d^+_{\Lambda_{|E(C_n)}}(u) - d^-_{\Lambda_{|E(C_n)}}(u)| > 0$ implying that $\textsc{MaxIm}(C_n) > 0$ which is absurd because C_n is an odd cycle.
- $\Rightarrow$ Since $\textsc{MaxIm}(G) = 0$, we know that G is eulerian. We will again work by induction on the number of cycles n. If $n = 1$, then our graph is eulerian with a unique cycle, hence it is a cycle. Now as $\textsc{MaxIm}(G) = 0$, necessarily it is an odd cycle and is therefore in $\mathscr{C}^{odd}$. Let $n \geq 2$, we assume that all graphs with $k \leq n-1$ cycles verifying $\textsc{MaxIm}(G) = 0$ are in $\mathscr{C}^{odd}$. Let G be a graph with n cycles such that $\textsc{MaxIm}(G) = 0$. Thanks to Lemma 2, there exists an cycle C of G such that $G - E(C)$ has at most one connected component G' that is not an isolated vertex.

  Suppose that $\textsc{MaxIm}(G') > 0$, let $\Lambda \in \overrightarrow{O}(G')$ with strictly positive imbalance. Let $u_0 \in V(G') \cap V(C)$, we name the vertices of C as follows: $u_0, u_1, \cdots, u_k = u_0$. Without loss of generality, we can assume that $d^+_{\Lambda}(u_0) - d^-_{\Lambda}(u_0) > 0$; if it was not the case, replace Λ by its reverse. We complete Λ in an orientation of G by orienting the edges of C: we orient u_0u_1 from u_0 to u_1 and go on as follows:

$$\forall i \in [\![1, k-1]\!], \begin{cases} \text{if } u_i \in V(G'), & \text{we orient } \{u_iu_{i+1}\} \text{ as } \{u_{i-1}u_i\}, \\ \text{otherwise}, & \text{we orient } \{u_iu_{i+1}\} \text{ as } \{u_iu_{i-1}\}. \end{cases}$$

  Where orienting an edge $\{ab\}$ as another edge $\{cd\}$ means orienting it from a to b if $\{cd\}$ was oriented from c to d and from b to a otherwise. Let us have a look at the resulting orientation Λ' (cf Figure 1): when completing Λ in Λ', the imbalance of the vertices in $V(G')\backslash\{u_0\}$ was left unchanged, the imbalance of the vertices in $V(C)\backslash V(G')$ equals 2 and the imbalance of u_0

was either left unchanged or augmented by two. Hence Λ' has strictly positive imbalance which contraditcts $\textsc{MaxIm}(G) = 0$, therefore, $\textsc{MaxIm}(G') = 0$.

Suppose $|V(G') \cap V(C)| \geq 2$ and let u and v be 2 distinct vertices in $V(G') \cap V(C))$ such that $u \neq v$. Thanks to proposition 1, we know that there exists an orientation $\Lambda \in \overrightarrow{O}(G')$ such that $\{w \in V/|d^+_\Lambda(w) - d^-_\Lambda(w)| = 0\} = \{v\}$ and without loss of generality, $d^+_\Lambda(u) - d^-_\Lambda(u) > 0$. We name the vertices of C as follows: $u = u_0u_1 \cdots u_k = u_0$, $v = u_l$ and we complete Λ in an orientation of G by orienting the edges of C: we orient $\{u_0u_1\}$ from u_0 and u_1 and go on as follows:

$$\forall i \in [\![1, k-1]\!] \setminus \{l\}, \begin{cases} \text{if } u_i \in V(G'), & \text{we orient } \{u_iu_{i+1}\} \text{ as } \{u_{i-1}u_i\}, \\ \text{otherwise}, & \text{we orient } \{u_iu_{i+1}\} \text{ as } \{u_iu_{i-1}\}. \end{cases}$$

And we orient $\{u_lu_{l+1}\}$ as $\{u_lu_{l-1}\}$. In the resulting orientation Λ', the imbalance of the vertices in $V(G')\backslash\{u, v\}$ was left unchanged, the imbalance of the vertices in $V(C)\backslash V(G')$ equals 2, the imbalance of v was augmented by two and the imbalance of u was either left unchanged or augmented by two. Hence Λ' contradicts $\textsc{MaxIm}(G) = 0$, therefore, $|V(G') \cap V(C)| = 1$.

Suppose C is even. We call $u \in V(G')$ such that $V(G') \cap V(C) = \{u\}$, and $\Lambda \in \overrightarrow{O}(G')$ such that $\{v \in V/|d^+_\Lambda(v) - d^-_\Lambda(v)| = 0\} = \{u\}$. We name the vertices of C as follows: $u = u_0u_1 \cdots u_k = u_0$ and we complete Λ in an orientation of G by orienting the edges of C: we orient $\{u_0u_1\}$ from u_0 to u_1 and $\{u_iu_{i+1}\}$ as $\{u_iu_{i-1}\}$, $\forall i \in [\![1, k-1]\!]$. In the resulting orientation Λ', the imbalance of the vertices in $V(G')\backslash\{u\}$ was left unchanged, the imbalance of the vertices in $V(C)\backslash V(G')$ equals 2 and, C being even, the imbalance of u was augmented by two. Hence Λ' contradicts $\textsc{MaxIm}(G) = 0$, therefore, C is odd.

As G' is a graph with at most $n-1$ cycles verifying $\textsc{MaxIm}(G) = 0$, by induction hypothesis, there exist $C_1, \cdots, C_{n-1}$ odd cycles such that:
- $\cup_{i=1}^{n-1} C_i = G'$,
- $|V(\cup_{k=1}^{i-1} C_k) \cap V(C_i)| = 1, \ \forall i \in [\![2, n-1]\!]$.

Adding the odd cycle $C_n = C$, we directly obtain that $G \in \mathscr{C}^{odd}$.

□

Now in order to widen our perception of those graphs, let us show another characterization.

Theorem 4. *For every simple graph G,*

$$G \in \mathscr{C}^{odd} \Leftrightarrow G \textit{ is eulerian with no even cycle}$$

Proof. • $\Rightarrow$ By construction, every graph in $\mathscr{C}^{odd}$ is eulerian with no even cycle.

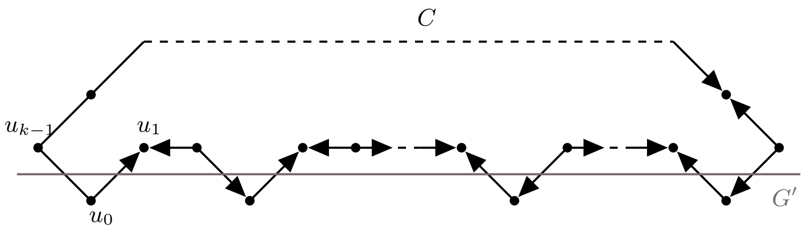


Fig. 1. The vertices of C in G' are left unchanged imbalance-wise, the other vertices of C are set to 2 and in the end $|d^+_{\Lambda'}(u_0) - d^-_{\Lambda'}(u_0)| \geq |d^+_{\Lambda}(u_0) - d^-_{\Lambda}(u_0)| > 0$

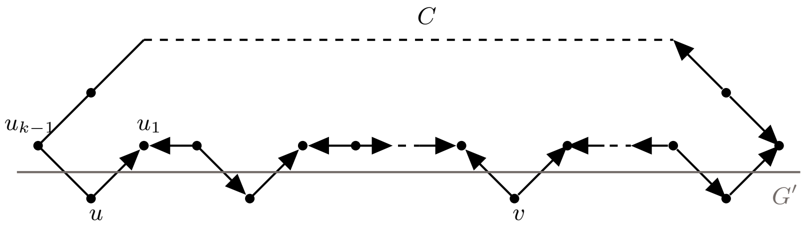


Fig. 2. The vertices of C in G' are left unchanged imbalance-wise except for v which is set to 2, like the other vertices of C and in the end $|d^+_{\Lambda'}(u_0) - d^-_{\Lambda'}(u_0)| \geq |d^+_{\Lambda}(u_0) - d^-_{\Lambda}(u_0)| > 0$

.

- $\Leftarrow$ We will once again work by induction on the number of cycles n.
  If $n = 1$, then our graph is eulerian with a unique odd cycle, hence it is an odd cycle and is therefore in $\mathscr{C}^{odd}$.
  Let $n \geq 2$, we assume that all eulerian graphs with no even cycle and $k \leq n-1$ odd cycles are in $\mathscr{C}^{odd}$. Let G be a graph with no even cycle and n odd cycles. Thanks to Lemma 2, there exists an odd cycle C of G such that $G - E(C)$ has only one connected component G' that is not an isolated vertex. As G' is eulerian and even-cycle-free with $n-1$ odd cycles, by induction hypothesis, $G' \in \mathscr{C}^{odd}$, hence there exist $C_1, \cdots, C_{n-1}$ odd cycles such that:
  - $\cup_{i=1}^{n-1} C_i = G'$,
  - $|V(\cup_{k=1}^{i-1} C_k) \cap V(C_i)| = 1, \ \forall i \in [\![2, n-1]\!]$.

  Suppose there exist u and v ($u \neq v$) belonging to $V(\cup_{k=1}^{n-1} C_k) \cap V(C)$. Since G' is connected, let p be an elementary path in G' between u and v. We can assume that u and v are the only vertices of C contained in p, otherwise we could replace v by the first vertex of C encountered when travelling on p from u. C defines two other vertex-disjoint paths between u and v: one even that we will call p_{even} and one odd that we will call p_{odd}. p being vertex disjoint with either p_{even} or p_{odd}, by concatenating it with the one corresponding to its parity, we obtain an even cycle of G, contradicting our hypothesis on G. This yields that $|V(C) \cap V(G')| = 1$. From that we can conclude
  - $\cup_{i=1}^{n} C_i = G$,
  - $|V(\cup_{k=1}^{i-1} C_k) \cap V(C_i)| = 1, \ \forall i \in [\![2, n]\!]$.

  Hence $G \in \mathscr{C}^{odd}$.

□

2 Complexity, Inapproximability and Approximability

In this section we will prove the NP-completeness and inapproximability of our problem and give an approximation algorithm based on the special case of bipartite graphs.

Concerning the complexity of MaxIm, we will show that the problem is NP-complete. More precisely, that answering if MaxIm(G) equals 2 for a graph G such that $\delta_G = 2$ is NP-complete. For that purpose we will introduce a variant of the satisfiability problem that we will reduce to a MaxIm instance: the not-all-equal at most 3-SAT(3V).

Not-all-equal at most 3-SAT(3V) is a restriction of not-all-equal at most 3-SAT which is itself a restriction of 3-SAT known to be NP-complete [19] where each clause contains at most three literals and in each clause, not all the literals can be true. Since 2-SAT can be solved in polynomial time, we hereafter deal only with formulas having at least one three-literals clause. The added restriction of not-all-equal at most 3-SAT(3V) is that each variable (not literal) appears at most three times in a formula. The resulting problem is still NP-complete.

Lemma 5. *The not-all-equal at most 3-SAT(3V) problem is NP-complete.*

Proof. See Appendix A.

Now we will associate to a not-all-equal at most 3-SAT(3V) instance φ with n variables $\{x_1, \cdots, x_n\}$ and m clauses $\{c_1, \cdots, c_m\}$ a graph G_φ for which the value w.r.t. MaxIm will give the answer to whether φ is satisfiable or not. If a variable x_i occurs only in positive literals (resp. only in negative literals), it follows that a satisfying assignment of the variables of φ must necssarily give the value TRUE (resp. FALSE) to x_i, therefore x_i can be removed from φ with conservation of the satisfiability. Thus, without loss of generality, we can assume that in any not-all-equal at most 3-SAT(3V) formula, every variable occurs at least once as a positive literal and at least once as a negative literal. G_φ consists of gadgets that mimic the variables and the clauses of φ and additional edges that connect them together:

- the gadget corresponding to a variable x_i consists of two vertices labeled x_i and $\neg x_i$ and one edge connecting them;

- the gadget corresponding to a two-literals clause $c_j = (l^1 \vee l^2)$, where l^1 and l^2 are its literals, consists in two vertices labeled $a^j_{l^1}$ and $b^j_{l^2}$ corresponding to l^1 and l^2 respectively (the index "l^k" of the vertices labels stands for the literal they represent, i.e. x_i if l^k is the variable x_i and $\neg x_i$ if l^k is the negation of the variable x_i) and one edge connecting them;

- the gadget corresponding to a three-literals clause gadget consists in six vertices and six edges. For a clause $c_j = (l^1 \vee l^2 \vee l^3)$, where l^1, l^2 and l^3 are its literals (the order is arbitrary), three vertices labeled $a^j_{l^1}$, $b^j_{l^2}$ and $b'^j_{l^3}$ correspond to l^1, l^2 and l^3 respectively. Three additional vertices are labeled

u^j, v^j and w^j and the gadgets' edges are $\{a^j_{l^1} u_j\}$, $\{a^j_{l^1} v_j\}$, $\{u_j w_j\}$, $\{v_j w_j\}$, $\{w_j b^j_{l^2}\}$ and $\{w_j b'^j_{l^3}\}$;

- $\forall i \in [\![1, n]\!]$, the vertex labeled x_i (resp. $\neg x_i$) is connected to all the vertices labeled $a^j_{x_i}$, $b^j_{x_i}$ or $b'^j_{x_i}$ (resp. $a^j_{\neg x_i}$, $b^j_{\neg x_i}$ or $b'^j_{\neg x_i}$), $\forall j \in [\![1, m]\!]$.

As an example, for a formula

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_4), \quad (2)$$

the corresponding graph G_φ is represented in Figure 3.

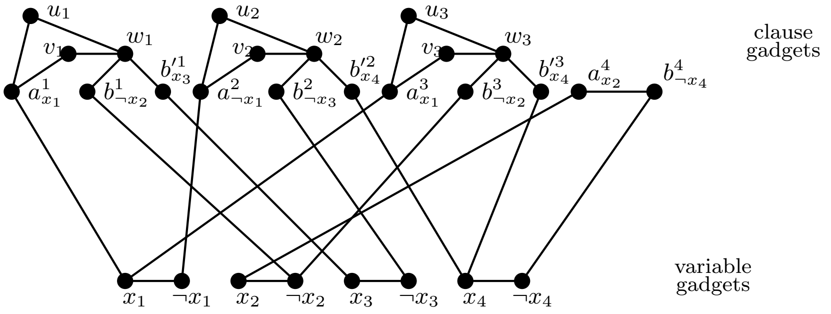


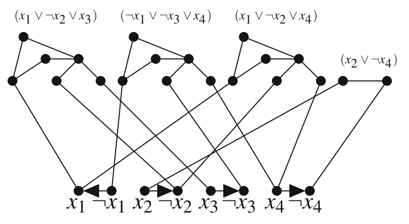
Fig. 3. G_φ for $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_4)$

Theorem 6. *A not-all-equal at most 3-SAT(3V) formula φ is satisfiable if and only if* $\textsc{MaxIm}(G_\varphi) = 2$.

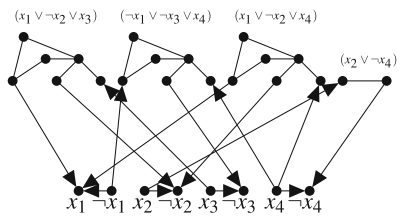
Proof. • $\Rightarrow$ Suppose φ is satisfiable and let $v : \{x_1, \cdots, x_n\} \to \{\text{TRUE}, \text{FALSE}\}$ be a satisfying assignment of $x_1, \cdots, x_n$. We know that $\delta_{G_\varphi} = 2$ which yields $\textsc{MaxIm}(G_\varphi) \leq 2$. So let us build an orientation $\Lambda \in \overrightarrow{O}(G_\varphi)$ which imbalance is greater than or equal to 2. First, we assign an orientation to the edges of the variable gadget:

$$\Lambda(\{x_i \neg x_i\}) = \begin{cases} x_i \neg x_i & \text{if } v(x_i) = \text{TRUE}; \\ \neg x_i x_i & \text{otherwise.} \end{cases}$$

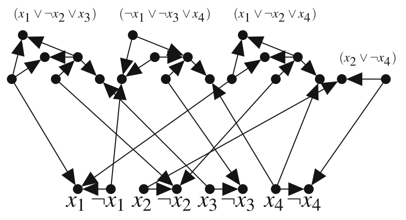
For example, for the formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_4)$ satisfied by the assignment $v(x_1, x_2, x_3, x_4) = (\text{FALSE}, \text{TRUE}, \text{TRUE}, \text{TRUE})$, the edges of the variable gadgets of graph G_φ are oriented as in figure 4(a). Since each variable x_i occurs at least once as a positive literal and at least once as a negative literal, $2 \leq d_{G_\varphi}(x_i) \leq 3$ and $2 \leq d_{G_\varphi}(\neg x_i) \leq 3$, $\forall i \in [\![1, n]\!]$. Then to ensure our objective on the imbalance of Λ, the orientation of the edges connecting vertex gadgets and clause gadgets must be such that $\forall i \in [\![1, n]\!]$, $|d^+_\Lambda(x_i) - d^-_\Lambda(x_i)| = d_{G_\varphi}(x_i)$


$(x_1 \vee \neg x_2 \vee x_3)$
$(\neg x_1 \vee \neg x_3 \vee x_4)$
$(x_1 \vee \neg x_2 \vee x_4)$
$(x_2 \vee \neg x_4)$
x_1 $\neg x_1$ x_2 $\neg x_2$ x_3 $\neg x_3$ x_4 $\neg x_4$

(a) orientation of the edges in the variable gadgets


$(x_1 \vee \neg x_2 \vee x_3)$
$(\neg x_1 \vee \neg x_3 \vee x_4)$
$(x_1 \vee \neg x_2 \vee x_4)$
$(x_2 \vee \neg x_4)$
x_1 $\neg x_1$ x_2 $\neg x_2$ x_3 $\neg x_3$ x_4 $\neg x_4$

(b) orientation of the edges between the variable gadgets and the clause gadgets


$(x_1 \vee \neg x_2 \vee x_3)$
$(\neg x_1 \vee \neg x_3 \vee x_4)$
$(x_1 \vee \neg x_2 \vee x_4)$
$(x_2 \vee \neg x_4)$
x_1 $\neg x_1$ x_2 $\neg x_2$ x_3 $\neg x_3$ x_4 $\neg x_4$

(c) orientation of the edges in the clause gadgets

Fig. 4. G_φ corresponding to $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_4)$ satisfied by $v(x_1, x_2, x_3, x_4) = (\text{FALSE}, \text{TRUE}, \text{TRUE}, \text{TRUE})$

and $|d^+_\Lambda(\neg x_i) - d^-_\Lambda(\neg x_i)| = d_{G_\varphi}(\neg x_i)$. In other words, for $i \in [\![1, n]\!]$, if $v(x_i) =$ TRUE (resp. $v(x_i) =$ FALSE), then the edges adjacent to the vertex x_i are oriented from x_i (resp. to x_i) and the edges adjacent to the vertex $\neg x_i$ are oriented to $\neg x_i$ (resp. from $\neg x_i$), e.g. Figure4(b).
So far, all the edges in the variables gadgets and the edges connecting the vertex gadgets and the clause gadgets have been oriented and the vertices in the variables gadgets have imbalance greater than or equal to 2. In order to complete our orientation Λ we have to orient the edges in the clause gadgets.
Let $c_j = (l^1 \vee l^2)$ be a two-literals clause. Since v satisfies φ, we know that exactly one of the two literals is true w.r.t. v. Which, according to the way we oriented edges so far, means that exactly one of $a^j_{l^1}$ and $b^j_{l^2}$ has one incoming arc from a variable gadget and the other has one outgoing arc to a variable gadget. If $a^j_{l^1}$ is the one with the incoming arc from a variable gadget (meanings that $v(l^1) =$ TRUE), then we assign $\Lambda(\{a^j_{l^1} b^j_{l^2}\}) = (b^j_{l^2} a^j_{l^1})$, otherwise the opposite. We obtain $|d^+_\Lambda(a^j_{l^1}) - d^-_\Lambda(a^j_{l^1})| = |d^+_\Lambda(b^j_{l^2}) - d^-_\Lambda(b^j_{l^2})| = 2$.
Let $c_j = (l^1 \vee l^2 \vee l^3)$ (the order is identical to which was chosen to build the clause gadget, i.e. $d_{G_\varphi}(a^j_{l^1}) = 3$ and $d_{G_\varphi}(b^j_{l^2}) = d_{G_\varphi}(b'^j_{l^3}) = 2$) be at three-literals clause. If the edge connecting $a^j_{l^1}$ to a variable gadget is oriented to $a^j_{l^1}$ (meanings that $v(l^1) =$ TRUE), then we assign $\Lambda(\{a^j_{l^1} u_j\}) = (u_j a^j_{l^1})$, $\Lambda(\{a^j_{l^1} v_j\}) = (v_j a^j_{l^1})$, $\Lambda(\{u_j w_j\}) = (u_j w_j)$ and $\Lambda(\{v_j w_j\}) = (v_j w_j)$. Since $v(l^1) =$ TRUE, either both $v(l^2)$ and $v(l^3)$ are FALSE or exactly one of $v(l^2)$

and $v(l^3)$ is TRUE and one is FALSE. If both are FALSE then $b^j_{l^2}$ and $b'^j_{l^3}$ have an outgoing arc to a variable gadget. In that case, we orient $w_j b^j_{l^2}$ and $w_j b'^j_{l^3}$ to w_j and we obtain $|d^+_\Lambda(a^j_{l^1}) - d^-_\Lambda(a^j_{l^1})| = 3$, $|d^+_\Lambda(b^j_{l^2}) - d^-_\Lambda(b^j_{l^2})| = |d^+_\Lambda((b'^j_{l^3}) - d^-_\Lambda((b'^j_{l^3})| = |d^+_\Lambda(u_j) - d^-_\Lambda(u_j)| = |d^+_\Lambda(v_j) - d^-_\Lambda(v_j)| = 2$ and $|d^+_\Lambda(w_j) - d^-_\Lambda(w_j)| = 4$. If exactly one of $v(l^2)$ and $v(l^3)$ is TRUE and one is FALSE, then exactly one of $b^j_{l^2}$ and $b'^j_{l^3}$ has an incoming arc from a variable gadget and the other an outgoing arc to a variable gadget. If $b^j_{l^2}$ is the one with the incoming arc from a variable gadget (meanings that $v(l^2) =$ TRUE and $v(l^3) =$ FALSE), then we assign $\Lambda(\{w_j b^j_{l^2}\}) = (w_j b^j_{l^2})$ and $\Lambda(\{w_j b'^j_{l^3}\}) = (b'^j_{l^3} w_j)$, otherwise the opposite. We obtain $|d^+_\Lambda(a^j_{l^1}) - d^-_\Lambda(a^j_{l^1})| = 3$ and $|d^+_\Lambda(b^j_{l^2}) - d^-_\Lambda(b^j_{l^2})| = |d^+_\Lambda(b'^j_{l^3}) - d^-_\Lambda(b'^j_{l^3})| = |d^+_\Lambda(u_j) - d^-_\Lambda(u_j)| = |d^+_\Lambda(v_j) - d^-_\Lambda(v_j)| = |d^+_\Lambda(w_j) - d^-_\Lambda(w_j)| = 2$.
If, on the other hand, the edge connecting $a^j_{l^1}$ to a variable gadget is oriented from $a^j_{l^1}$ (meanings that $v(l^1) =$ FALSE), then we assign $\Lambda(\{a^j_{l^1} u_j\}) = (a^j_{l^1} u_j)$, $\Lambda(\{a^j_{l^1} v_j\}) = (a^j_{l^1} v_j)$, $\Lambda(\{u_j w_j\}) = (w_j u_j)$ and $\Lambda(\{v_j w_j\}) = (w_j v_j)$. By symmetry, we conclude in the same way that $|d^+_\Lambda(a^j_{l^1}) - d^-_\Lambda(a^j_{l^1})| = 3$ and $|d^+_\Lambda(b^j_{l^2}) - d^-_\Lambda(b^j_{l^2})| = |d^+_\Lambda(b'^j_{l^3}) - d^-_\Lambda(b'^j_{l^3})| = |d^+_\Lambda(u_j) - d^-_\Lambda(u_j)| = |d^+_\Lambda(v_j) - d^-_\Lambda(v_j)| = |d^+_\Lambda(w_j) - d^-_\Lambda(w_j)| = 2$.
Consequently, the imbalance of the resulting orientation Λ is greater than or equal to 2, e.g. Figure 4(c).

- $\Leftarrow$ Now we assume that $\textsc{MaxIm}(G_\varphi) = 2$, let $\Lambda \in \overrightarrow{O}(G_\varphi)$ with optimal imbalance. Since all the vertices in the variable gadgets have degree at most 3, each vertex x_i (or $\neg x_i$) is necessarily adjacent to only incoming arcs or only outgoing arcs w.r.t. Λ. We will show that the assignment $v : \{x_1, \cdots, x_n\} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ of $x_1, \cdots, x_n$ defined by

$$v(x_i) = \begin{cases} \text{TRUE} & \text{if } d^+_\Lambda(x_i) > d^-_\Lambda(x_i); \\ \text{FALSE} & \text{otherwise;} \end{cases}$$

satisfies φ. Suppose φ doesn't satisfy a clause c_j, $j \in [\![1, m]\!]$. If c_j is a two-literals clause $(l^1 \vee l^2)$ then either $v(l^1) = v(l^2) =$ TRUE or $v(l^1) = v(l^2) =$ FALSE, i.e. either both $a^j_{l^1}$ and $b^j_{l^2}$ have an incoming arc from a variable gadget or both have an outgoing arc to a variable gadget and in both cases, whichever is the orientation assigned to $a^j_{l^1} b^j_{l^2}$ by Λ, either $a^j_{l^1}$ or $b^j_{l^2}$ have a zero imbalance which contradicts our assumption. So c_j is a three-literals clause $(l^1 \vee l^2 \vee l^3)$ (the order is identical to which was chosen to build the clause gadget, i.e. $d_{G_\varphi}(a^j_{l^1}) = 3$ and $d_{G_\varphi}(b^j_{l^2}) = d_{G_\varphi}(b'^j_{l^3}) = 2$). Then either $v(l^1) = v(l^2) = v(l^3) =$ TRUE or $v(l^1) = v(l^2) = v(l^3) =$ FALSE, i.e. either all $a^j_{l^1}$, $b^j_{l^2}$ and $b'^j_{l^3}$ have an incoming arc from a variable gadget or they all have an outgoing arc to a variable gadget. In the first case, it implies $\Lambda(\{a^j_{l^1} u_j\}) = (u_j a^j_{l^1})$, $\Lambda(\{a^j_{l^1} v_j\}) = (v_j a^j_{l^1})$, $\Lambda(\{u_j w_j\}) = (u_j w_j)$, $\Lambda(\{v_j w_j\}) = (v_j w_j)$, $\Lambda(\{w_j b^j_{l^2}\}) = (w_j b^j_{l^2})$ and $\Lambda(\{w_j b'^j_{l^3}\}) = (w_j b'^j_{l^3})$, and we obtain $|d^+_\Lambda(w_j) - d^-_\Lambda(w_j)| = 0$ which contradicts the optimality of Λ. Similarly, in the second case it implies that the orientations assigned to the edges of the clause gadgets are the opposite from the previous ones and we obtain the same contradiction.

So we can conclude that v does satisfy φ.

□

Corollary 7. MaxIm *is* NP*-complete and inapproximable within* $\frac{1}{2} + \varepsilon$ *where* $\varepsilon \in \mathbb{R}_+^*$*, unless* P = NP.

Proof. Let $\varepsilon \in \mathbb{R}_+^*$, suppose that there existed a polynomial approximation algorithm giving $val \geq (\frac{1}{2} + \varepsilon)\,\textsc{MaxIm}(G)$ for an input graph G. Let φ be a not-all-equal at most 3-SAT(3V) formula and G_φ its associated graph. Since G_φ contains at least one three-literals clause gadget, we know that G_φ contains an even cycle and $\delta_{G_\varphi} = 2$. This leads to $\textsc{MaxIm}(G_\varphi) \in \{1, 2\}$ and since $(\frac{1}{2} + \varepsilon)\,\textsc{MaxIm}(G_\varphi) \leq val \leq \textsc{MaxIm}(G_\varphi)$, if the polynomial approximation algorithm returns a value less than or equal to 1 then

$$(\frac{1}{2} + \varepsilon)\,\textsc{MaxIm}(G_\varphi) \leq 1 \Rightarrow \textsc{MaxIm}(G_\varphi) < 2 \Rightarrow \textsc{MaxIm}(G_\varphi) = 1;$$

and if it returns a value greater than 1, then $\textsc{MaxIm}(G_\varphi)$ is greater than 1 hence equal to 2. In other words the polynomial approximation algorithm output answers whether φ is satisfiable or not which is absurd unless P = NP. □

Now we consider the case of bipartite graphs: if $G = (V1 \bigsqcup V_2, E)$ is a bipartite graph, the orientation that consists in assigning to each edge in E the orientation from its extremity in V_1 to its extremity in V_2 has an imbalance equal to δ_G, i.e. optimal. This simple case permits us to obtain the following lower bound:

Theorem 8. *For every graph* G*,*

$$\textsc{MaxIm}(G) \geq \lceil \frac{\delta_G}{2} \rceil - 1.$$

Proof. Let (V_1, V_2) be a partition of V corresponding to a cut $C \subset E$ such that we have $|\delta(\{v\}) \cap C| \geq \lceil \frac{d(v)}{2} \rceil$, $\forall v \in V$. Such a cut exists: for example a maximum cardinality cut verifies this property, otherwise we could find a higher cardinality cut by switching a vertex $v \in V$ s.t. $|\delta(\{v\}) \cap C| < \lceil \frac{d(v)}{2} \rceil$ from V_1 to V_2 (or the contrary). Moreover, if we iterated this process starting from a random cut, we would converge in polynomial time time to a such a cut. Now we define $\Lambda \in \overrightarrow{O}(G)$ as follows. We begin by orienting all edges in C from V_1 to V_2. Then for any $i \in \{1, 2\}$, we orient the edges of the induced subgraph $G[V_i]$. We add a new vertex v_0 and an edge between v_0 and each vertex with an odd degree in $G[V_i]$ if it isn't eulerian and we consider a decomposition of its edges into edge-disjoint cycles. we orient each of these cycles as a directed cycle. Removing v_0 if necessary, the imbalance of each vertex in $G[V_i]$ is now in $\{-1, 0, 1\}$ which implies that $\forall v \in V$ we have $|d_\Lambda^+(v) - d_\Lambda^-(v)| \geq \lceil \frac{d(v)}{2} \rceil - 1$, hence, $\textsc{MaxIm}(G) \geq \lceil \frac{\delta_G}{2} \rceil - 1$. □

Since the imbalance of the orientation defined in that proof is at least $\frac{\lceil \frac{\delta_G}{2} \rceil - 1}{\delta_G} \cdot$ $\textsc{MaxIm}(G)$, we can derive a polynomial $(\frac{1}{2} - \frac{1}{\delta_G})$-approximation algorithm. It is also easy to see that when $\delta_G \equiv 0[4]$ then $\textsc{MaxIm}(G) \geq \lceil \frac{\delta_G}{2} \rceil$ while $\textsc{MaxIm}(G) \geq \lceil \frac{\delta_G + 1}{2} \rceil$ when $\delta_G \equiv 1[4]$. This leads to a $(\frac{1}{2})$-approximation algorithm when either $\delta_G \equiv 0[4]$ or $\delta_G \equiv 1[4]$.

3 Further Research

While computing the most imbalanced orientation of a graph is generally difficult, the problem turns out to be easy for cactus graphs. It may be the same for other graph classes, characterizing such graph classes would be interesting.

We are currently looking for efficient mathematical programming formulations to solve the problem for large size graphs. Details will follow in the extended version of the paper. One can also study the weighted version of the problem.

Appendix A Proof of Lemma 5

Let φ be a not-all-equal at most 3-SAT formula with n variables $\{x_1, \cdots, x_n\}$ and m clauses $\{c_1, \cdots, c_m\}$ and for all $i \in [\![1, n]\!]$, let $k_i \in \mathbb{N}$ be the number of occurences of x_i in φ. We assume that there is at least one variable x_i that has at least 4 occurences in φ (otherwise φ is already a not-all-equal at most 3-SAT(3V) formula) and we will build from φ a not-all-equal at most 3-SAT(3V) φ' such that φ and φ' are equisatisfiable as follows.

- For all $i \in [\![1, n]\!]$, if $k_i \geq 4$ then we introduce k_i new variables $\{x_i^1, \cdots, x_i^{k_i}\}$ and for $l \in [\![1, k_i]\!]$ we replace the l-th occurence of x_i in φ with x_i^l.
- For all $i \in [\![1, n]\!]$, if $k_i \geq 4$ then we add k_i new clauses $\{c_{x_i}^1, \cdots, c_{x_i}^{k_i}\}$ where for $l \in [\![1, k_i - 1]\!]$, $c_{x_i}^l = (x_i^l \vee \neg x_i^{l+1})$ and $c_{x_i}^l = (x_i^l \vee \neg x_i^1)$.

Suppose there exists an assignment $v : \{x_1, \cdots, x_n\} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ of $x_1, \cdots, x_n$ satisfying φ. Then

$$v' : \begin{array}{ll} x_i \mapsto v(x_i) & \forall i \in [\![1, n]\!] \text{ s.t. } k_i \leq 3; \\ x_i^l \mapsto v(x_i) & \forall i \in [\![1, n]\!] \text{ s.t. } k_i \geq 4 \text{ and } \forall l \in [\![1, k_i]\!]; \end{array}$$

is an assignment of the variables x_i and x_i^l satisfying φ' for

- $\forall j \in [\![1, m]\!]$, the values of the literals of c_j w.r.t. v and v' are piecewise equal so $v'(c_j) = v(c_j) = \text{TRUE}$ and v' is not-all-equal for c_j as well as v is;
- $\forall i \in [\![1, n]\!]$ s.t. $k_i \geq 4$, $\forall l \in [\![1, k_i - 1]\!]$, $v'(x_i^l) = v'(x_i^{l+1}) = v(x_i)$ and $v'(x_i^{k_i}) = v'(x_i^1) = v(x_i)$ so we directly have $\forall l \in [\![1, k_i - 1]\!]$, $v'(c_{x_i}^l) = \text{TRUE}$ and $v'(c_{x_i}^{k_i}) = \text{TRUE}$ and v' is not-all-equal for each of these clauses since they all consist of two literals having opposite values w.r.t. v'.

As an example, for a formula

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee x_3),$$

where x_1 occurs five times and x_3 four so we add nine new variables x_1^1, x_1^2, x_1^3, x_1^4, x_1^5, x_3^1, x_3^2, x_3^3 and x_3^4 and nine new clauses:

$$\begin{aligned}\varphi' =&(x_1^1 \vee \neg x_2 \vee x_3^1) \wedge (\neg x_1^2 \vee \neg x_3^2 \vee x_4) \wedge (x_1^3 \vee \neg x_2) \wedge (\neg x_1^4 \vee \neg x_3^3 \vee \neg x_4) \wedge (x_1^5 \vee x_3^4)\\ &\wedge (x_1^1 \vee \neg x_1^2) \wedge (x_1^2 \vee \neg x_1^3) \wedge (x_1^3 \vee \neg x_1^4) \wedge (x_1^4 \vee \neg x_1^5) \wedge (x_1^5 \vee \neg x_1^1)\\ &\wedge (x_3^1 \vee \neg x_3^2) \wedge (x_3^2 \vee \neg x_3^3) \wedge (x_3^3 \vee \neg x_3^4) \wedge (x_3^4 \vee \neg x_3^1).\end{aligned}$$

Now suppose there exists an assignment v' of the x_i and x_i^l satisfying φ' and let $i \in [\![1, n]\!]$ such that $k_i \geq 4$. If we take a look at the clauses $c_{x_i}^1, \cdots, c_{x_i}^{k_i}$, we notice that if $v'(x_i^1) = \text{FALSE}$ then for $c_{x_i}^1$ to be satisfied, $v'(\neg x_i^2) = \text{TRUE}$, i.e. $v'(x_i^2) = \text{FALSE}$, then for $c_{x_i}^2$ to be satisfied, $v'(\neg x_i^3) = \text{TRUE}$ ...etc. Repeating this argument, we obtain that if $v'(x_i^1) = \text{FALSE}$ then $v'(x_i^1) = v'(x_i^2) = \cdots = v'(x_i^{k_i}) = \text{FALSE}$. Similarly, if $v'(x_i^{k_i}) = \text{TRUE}$ then for $c_{x_i}^{k_i}$ to be satisfied, $v'(\neg x_i^{k_i-1}) = \text{FALSE}$, i.e. $v'(x_i^{k_i-1}) = \text{TRUE}$, then for $c_{x_i}^{k_i-1}$ to be satisfied, $v'(\neg x_i^{k_i-2}) = \text{FALSE}$ ...etc. Hence if $v'(x_i^{k_i}) = \text{TRUE}$ then $v'(x_i^{k_i}) = v'(x_i^{k_i-1}) = \cdots = v'(x_i^1) = \text{TRUE}$. This yields that

$$\forall i \in [\![1, n]\!] \text{ s.t. } k_i \geq 4,\ v'(x_i^1) = v'(x_i^2) = \cdots = v'(x_i^{k_i}).$$

Hence for all $i \in [\![1, n]\!]$ such that $k_i \geq 4$, we can replace $x_i^1, \cdots, x_i^{k_i}$ by a unique variable x_i and doing so the clauses $c_{x_i}^1, \cdots, c_{x_i}^{k_i}$ become trivial and can be removed and only φ remains. So the following assignment of $x_1, \cdots, x_n$:

$$v : \begin{array}{ll} x_i \mapsto v'(x_i) & \forall i \in [\![1, n]\!] \text{ s.t. } k_i \leq 3;\\ x_i \mapsto v'(x_i^1) & \forall i \in [\![1, n]\!] \text{ s.t. } k_i \geq 4;\end{array}$$

satisfies φ. We have just shown that φ and φ' are equisatisfiable. □

References

1. Asahiro, Y., Jansson, J., Miyano, E., Ono, H., Zenmyo, K.: Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 167–177. Springer, Heidelberg (2007)
2. Asahiro, Y., Miyano, E., Ono, H.: Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. In: Proceedings of the Fourteenth Computing: the Australasian Theory Symposium(CATS2008), Wollongong, NSW, Australia (2008)
3. Asahiro, Y., Jansson, J., Miyano, E., Ono, H.: Degree-constrained graph orientation: maximum satisfaction and minimum violation. In: Kaklamanis, C., Pruhs, K. (eds.) WAOA 2013. LNCS, vol. 8447, pp. 24–36. Springer, Heidelberg (2014)
4. Bang-Jensen, J., Gutin, G.: Orientations of graphs and digraphs in Digraphs: Theory, Algorithms and applications, 2nd edition, pp. 417–472. Springer (2009)

5. Biedl, T., Chan, T., Ganjali, Y., Hajiaghayi, M., Wood, D.R.: Balanced vertex-orderings of graphs. Discrete Applied Mathematics **48**(1), 27–48 (2005)
6. Chrobak, M., Eppstein, D.: Planar orientations with low out-degree and compaction of adjacency matrices. Theoretical Computer Sciences **86**, 243–266 (1991)
7. Chung, F., Garey, M., Tarjan, R.: Strongly connected orientations of mixed multigraphs. Networks **15**, 477–484 (1985)
8. Chvátal, V., Thomassen, C.: Distances in orientation of graphs. Journal of Combinatorial Theory, Series B **24**, 61–75 (1978)
9. Diestel, R.: Graph Theory, 4th edn. Springer (2010)
10. Fomin, F., Matamala, M., Rapaport, I.: Complexity of approximating the oriented diameter of chordal graphs. Journal of Graph Theory **45**(4), 255–269 (2004)
11. Ford, L.R., Fulkerson, D.R.: Flows in networks. Princeton University Press, Princeton (1962)
12. Frank, A., Gyárfás, A.: How to orient the edges of a graph? Colloquia Mathematica Societatis János Bolyai **18**, 353–364 (1976)
13. Harary, F., Krarup, J., Schwenk, A.: Graphs suppressible to an edge. Canadian Mathematical Bulletin **15**, 201–204 (1971)
14. Kára, J., Kratochvíl, J., Wood, D.R.: On the complexity of the balanced vertex ordering problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 849–858. Springer, Heidelberg (2005)
15. Landau, H.G.: On dominance relations and the structure of animal societies III. the condition for a score structure. The Bulletin of Mathematical Biophysics **15**, 143–148 (1953)
16. Mubayi, D., Will, T.G., West, D.B.: Realizing Degree Imbalances in Directed Graphs. Discrete Mathematics **239**(173), 147–153 (2001)
17. Nash-Williams, C.: On orientations, connectivity and odd vertex pairings in finite graphs. Canadian Journal of Mathematics **12**, 555–567 (1960)
18. Robbins, H.: A theorem on graphs with an application to a problem of traffic control. American Mathematical Monthly **46**, 281–283 (1939)
19. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp. 216–226 (1978)

Cheeger Inequalities for General Edge-Weighted Directed Graphs

T.-H. Hubert Chan, Zhihao Gavin Tang, and Chenzi Zhang(✉)

The University of Hong Kong, Hong Kong, China
{hubert,zhtang,czzhang}@cs.hku.hk

Abstract. We consider Cheeger Inequalities for general edge-weighted directed graphs. Previously the directed case was considered by Chung for a probability transition matrix corresponding to a strongly connected graph with weights induced by a stationary distribution. An Eulerian property of these special weights reduces these instances to the undirected case, for which recent results on multi-way spectral partitioning and higher-order Cheeger Inequalities can be applied.

We extend Chung's approach to general directed graphs. In particular, we obtain higher-order Cheeger Inequalities for the following scenarios:
(1) The underlying graph needs not be strongly connected.
(2) The weights can deviate (slightly) from a stationary distribution.

1 Introduction

There have been numerous works relating the expansion properties of an undirected graph with the eigenvalues of its Laplacian [1,3,9]. Given an undirected graph with non-negative edge weights, the weight of a vertex is the sum of the weights of its incident edges. Then, the *expansion* $\rho(S)$ of a subset S of vertices is the ratio of the sum of the weights of edges having only one end-point in S to the sum of the weights of vertices in S. The celebrated Cheeger's Inequality [1,6] relates the smallest expansion of a subset of vertices having at most half the sum of vertex weights with the second smallest eigenvalue of the corresponding normalized Laplacian. Recently, there have been extensions to the case where the expansions of k disjoint subsets are related to the k-th smallest eigenvalue [13].

The notion of expansion can be extended to directed graphs, where the weight of a vertex is the sum of the weights of its out-going edges. Then, the expansion of a subset S is defined with respect to the sum of the weights of edges going out of S. Chung [8] considered the special case for a probability transition matrix whose non-zero entries correspond to the edges of a **strongly connected graph**. The weights of the vertices are chosen according to the (unique) stationary distribution, and the weight of an edge is the probability mass going along the edge under this stationary distribution. Under this specific choice of weights, Chung has proved an analogous Cheeger's Inequality [8] for directed graphs.

This research is partially funded by a grant from Hong Kong RGC under the contract HKU17200214E.

In this paper, we explore how this relationship between expansion and spectral properties can be extended to more general cases for directed graphs. In particular, we consider the following cases.

1. The directed graph is not strongly connected.
2. The weights of vertices deviate (slightly) from the stationary distribution.

As we shall explain, each of these cases violates the technical assumptions that are used by Chung to derive the Cheeger Inequality for directed graphs. We explore what expansion notions are relevant in these scenarios, and how to define Laplacians whose eigenvalues can capture these notions.

1.1 Overview of Chung's Approach [8]

All spectral arguments rely on some symmetric matrix, which has the desirable properties of having real eigenvalues and an orthonormal basis of eigenvectors. For an undirected graph (with non-negative edge weights), its normalized Laplacian is a symmetric matrix. To apply spectral analysis on directed graphs, one should consider what the natural candidates for symmetric matrices should be and whether they have any significance. We explain the importance of the technical assumptions made by Chung in the analysis of the transition matrix $\mathbf{P}$ associated with the random walk on some directed graph $G(V, E)$.

(1) Choice of Weights. Suppose $\boldsymbol{\phi} : V \to \mathbb{R}_+$ is a stationary distribution of the transition matrix $\mathbf{P}$. Then, the weights are chosen such that each vertex u has weight $\phi(u)$, and each (directed) edge (u, v) has weight $\phi(u) \cdot P(u, v)$, which is the probability mass going from u to v in one step of the random walk starting from distribution $\boldsymbol{\phi}$.

Suppose the starting vertex u of a random walk is chosen according to distribution $\boldsymbol{\phi}$. The expansion of a subset S has the following meaning: conditioning on the event that u is in S, it is the probability that the next step of the random walk goes out of S.

This notion of expansion can be defined with respect to any distribution on the vertex set V, but the edge weights induced by a stationary distribution has the following *Eulerian* property: for any subset S of vertices, the sum of weights of edges going out of S is the same as that of edges going into S.

Hence, one can consider the underlying undirected graph such that each undirected edge has weight that is the average of those for the corresponding directed edges in each direction. Then, because of the Eulerian property, for any subset S, its expansion in the directed graph defined with respect to the out-going edges is exactly the same as its expansion defined with respect to the undirected graph (with edge weights defined above). Therefore, it suffices to consider the normalized Laplacian of the undirected graph to analyze the expansion properties of the directed graph.

(2) Irreducibility of Transition Matrix. This means that the underlying directed graph with edges corresponding to transitions with non-zero probabilities is strongly connected. Under this assumption, the stationary distribution is unique, and every vertex has a positive mass.

If the directed graph is not strongly connected, a strongly connected component is known as a *sink* if there is no edge going out of it. If there is more than one sink, the stationary distribution is not unique. Moreover, under any stationary distribution, any vertex in a non-sink has probability mass zero. Hence, Chung's method essentially deletes all non-sinks before considering the expansion properties of the remaining graph.

In this paper, we explore ways to consider expansion properties that involve the non-sinks of a directed graph that is not strongly connected.

1.2 Our Contribution

The contribution of this paper is mainly conceptual, and offers an approach to extend Chung's spectral analysis of transition matrices to the scenarios when the underlying directed graph is not strongly connected, or when the vertex weights do not follow a stationary distribution. On a high level, our technique to handle both issues is to add a new vertex to the graph and define additional transition probabilities involving the new vertex such that the new underlying graph is strongly connected, and the expansion properties for the old vertices are also preserved in the new graph. Therefore, Chung's technique can be applied after the transformation. We outline our approaches and results as follows.

(1) Transition matrix whose directed graph is not necessarily strongly connected. Given a transition matrix $\mathbf{P}$ corresponding to a random walk on a graph $G(V, E)$ and a subset $S \subseteq V$ of vertices, we denote by $\mathbf{P}|_S$ the submatrix defined by restricting $\mathbf{P}$ only to the rows and the columns corresponding to S.

It is known [7] that the eigenvalues of $\mathbf{P}$ are the union of the eigenvalues of $\mathbf{P}|_C$ over all strongly connected components C in directed graph G. An important observation is that as long as the strongly connected components and the transition probabilities within a component remain the same, the eigenvalues of $\mathbf{P}$ are independent of the transition probabilities between different strongly connected components. This suggests that it might be difficult to use spectral properties to analyze expansion properties involving edges between different strongly connected components.

Therefore, we propose that it makes sense to consider the expansion properties for each strongly connected component separately. If C is a sink (i.e., there is no edge leaving C), then $\mathbf{P}|_C$ itself is a probability transition matrix, for which Chung's approach can be applied by using the (unique) stationary distribution on C.

However, if C is a non-sink, then there is no stationary distribution for $\mathbf{P}|_C$, because there is non-zero probability mass leaking out of C in every step of the random walk. For the non-trivial case when $|C| \geq 2$, by the Perron-Frobenius Theorem [10], there exists some maximal eigenvalue $\lambda > 0$ with respect to the complex norm, and unique (left) eigenvector $\boldsymbol{\phi}$ with strictly positive coordinates such that $\boldsymbol{\phi}^{\mathrm{T}}\mathbf{P}|_C = \lambda\boldsymbol{\phi}^{\mathrm{T}}$. When $\boldsymbol{\phi}$ is normalized such that all coordinates sum to 1, we say that $\boldsymbol{\phi}$ is the *diluted stationary distribution* of $\mathbf{P}|_C$. It is stationary in the sense that if we start the random walk with distribution $\boldsymbol{\phi}$, then conditioning on the event that the next step remains in C (which has probability λ), we have the same distribution $\boldsymbol{\phi}$ on C.

Hence, we can define the expansion of a subset S in C with respect to the diluted stationary distribution $\boldsymbol{\phi}$. Given a vertex $u \in C$ and a vertex $v \in V$

(that could be outside C), the weight of the edge (u, v) is $\phi(u) \cdot P(u, v)$. Observe that the sum of weights of edges going out of u is $\phi(u)$. Hence, the expansion of a subset S in C are due to edges leaving S that can either stay in or out of the component C.

In order to analyze this notion of expansion using Chung's approach, we construct a strongly connected graph on the component C together with a new vertex v_0, which absorbs all the probabilities leaking out of C, and returns them to C according to the diluted stationary distribution $\boldsymbol{\phi}$. This defines a probability transition matrix $\widehat{\mathbf{P}}$ on the new graph that is strongly connected with various nice properties. For instance, $\widehat{\mathbf{P}}$ has 1 as the maximal eigenvalue with the corresponding left eigenvector formed from the diluted stationary distribution $\boldsymbol{\phi}$ by appending an extra coordinate corresponding to the new vertex with value $1-\lambda$.

One interesting technical result (Lemma 1) is that the new transition matrix $\widehat{\mathbf{P}}$ preserves the spectral properties of $\mathbf{P}|_C$ in the sense that the eigenvalues of $\widehat{\mathbf{P}}$ can be obtained by removing λ from the multi-set of eigenvalues of $\mathbf{P}|_C$ and including 1 and $\lambda - 1$. In other words, other than the removal of λ and the inclusion of 1 and $\lambda - 1$, all other eigenvalues are preserved, even up to their algebraic and geometric multiplicities.

Hence, we can use Chung's approach to define a symmetric Laplacian for $\widehat{\mathbf{P}}$, and use the recent results from Lee et al. [13] on higher-order Cheeger Inequalities to achieve an analogous result for a strongly connected component in a directed graph. In particular, multi-way partition expansion is considered. For a subset C of vertices, we denote:

$\rho_k(C) := \min\{\max_{i\in[k]} \rho(S_i) : S_1, S_2, \ldots, S_k$ are disjoint subsets of C$\}$.

Theorem 1. *Suppose C is a strongly connected component of size n associated with some probability transition matrix, and the expansion $\rho(S)$ of a subset S of vertices within C is defined with respect to the diluted stationary distribution $\boldsymbol{\phi}$ as described above.*

Then, one can define a Laplacian matrix with dimension $(n + 1) \times (n + 1)$ having eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n+1}$ such that for $1 \leq k \leq n$, we have $\frac{\lambda_k}{2} \leq \rho_k(C) \leq O(k^2) \cdot \sqrt{\lambda_{k+1}}$.

(2) Vertex weights deviate from stationary distribution. Given a transition matrix $\mathbf{P}$, recall that in Chung's approach, the expansion is defined with the careful choice of setting each vertex's weight according to a stationary distribution. We consider the case when the vertex weights $\boldsymbol{\phi} : V \to \mathbb{R}_+$ can deviate from a stationary distribution of $\mathbf{P}$.

Suppose each vertex is assigned a positive weight according to $\boldsymbol{\phi}$. Then, the following parameter measures how much $\boldsymbol{\phi}$ deviates from a stationary distribution:

$$\varepsilon := 1 - \min_{u\in V} \frac{\phi(u)}{\sum_{v\in V} \phi(v) \cdot P(v, u)}.$$

A smaller value of ε means that $\boldsymbol{\phi}$ is closer to a stationary distribution. In particular, if $\varepsilon = 0$, then $\boldsymbol{\phi}$ is a stationary distribution.

Our idea is to first scale down all probabilities in $\mathbf{P}$ by a factor of $(1 - \varepsilon)$. We add a new vertex v_0 to absorb the extra ε probability from each existing vertex. Then, we define the transition probabilities from v_0 to the original vertices carefully such that each original vertex u receives the same weight $\phi(u)$ after one step. In other words, we can append a new coordinate corresponding to v_0 to $\boldsymbol{\phi}$ to obtain a stationary distribution $\widehat{\boldsymbol{\phi}}$ for the transition matrix $\widehat{\mathbf{P}}$ of the augmented random walk. Moreover, for each subset $S \subset V$ of the original vertices, the new expansion $\widehat{\rho}(S)$ with respect to $\widehat{\boldsymbol{\phi}}$ and $\widehat{\mathbf{P}}$ can be related to the old expansion $\rho(S)$ as follows: $\widehat{\rho}(S) = (1 - \varepsilon) \cdot \rho(S) + \varepsilon$.

Therefore, we can apply Chung's approach to $\widehat{\mathbf{P}}$ and $\widehat{\boldsymbol{\phi}}$ to construct a symmetric Laplacian matrix, whose eigenvalues are related to the expansion properties using the results by Lee et al. [13].

Theorem 2. *Suppose n vertices have positive weights defined by $\phi : V \to \mathbb{R}_+$, and $\varepsilon \geq 0$ is the parameter defined above. Then, there exists a symmetric Laplacian matrix with dimension $(n+1)\times(n+1)$ and eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n+1}$ such that for $1 \leq k \leq n$, we have $\frac{\lambda_k}{2} \leq (1-\varepsilon)\cdot\rho_k(V)+\varepsilon \leq O(k^2)\cdot\sqrt{\lambda_{k+1}}$.*

In the full version, we show that if we allow a self-loop at the new vertex v_0 with negative weight, we can slightly improve the left hand side of the inequality in Theorem 2.

1.3 Related Work

Since the Cheeger's Inequality [6] was introduced in the context of Riemannian geometry, analogous results have been achieved by Alon et al. [1,3] to relate the expansion of an undirected graph with the smallest positive eigenvalue of the associated Laplacian matrix. The reader is referred to the standard textbook by Chung [9] on spectral graph theory for a more comprehensive introduction of the subject.

As far as we know, the only previous attempt to apply spectral analysis to directed graphs was by Chung [8], who reduced the special case of directed instances induced by stationary distributions into undirected instances. On a high level, our approach is to reduce general directed instances into instances induced by stationary distributions.

Recently, for undirected instances, Lee et al. [13] extended Cheeger's Inequality to relate higher order eigenvalues with multi-way spectral partition. This result was further improved by Kwok et al. [12]. Since Chung's approach [8] made use of the Laplacian induced by an undirected instance, the higher order Cheeger Inequalities can be directly applied to the cases considered by Chung.

The reader can refer to the survey on spectral partitioning by Shewchuck [15], who also mentioned expansion optimization problems with negative edge weights. Other applications of spectral analysis include graph coloring [2,4], web search [5,11] clustering [14], image segmentation [16,17], etc.

2 Preliminaries

We consider a graph $G(V, E)$ with non-negative edge weights $w : E \to \mathbb{R}_+$. In most cases, we consider directed graphs, but we will also use results for undirected graphs. Note that a vertex might have a self-loop (even in an undirected graph).

For a subset $S \subseteq V$, $\partial(S)$ is the set of edges leaving S in a directed graph, whereas in an undirected graph, it includes those edges having exactly one endpoint in S (excluding self-loops). Given the edge weights, vertex weights are defined as follows. For each $u \in V$, its weight $w(u)$ is the sum of the weights of its out-going edges (including its self-loop) in a directed graph, whereas in an undirected graph, the edges incident on u are considered.

The *expansion* $\rho(S)$ of a subset S (with respect to w) is defined as:

$$\frac{w(\partial(S))}{w(S)} = \frac{\sum_{e \in \partial(S)} w(e)}{\sum_{u \in S} w(u)}.$$

In this paper, we use bold capital letters (such as $\mathbf{A}$) to denote matrices and bold small letters (such as $\boldsymbol{\phi}$) to denote column vectors. The transpose of a matrix $\mathbf{A}$ is denoted as $\mathbf{A}^{\mathrm{T}}$. For a positive integer n, $\mathbf{I}_n$ is the $n \times n$ identity matrix, and $\mathbf{0}_n$ and $\mathbf{1}_n$ are the all zero's and all one's column vectors, respectively, of dimension n, where the subscript n is omitted if the dimension is clear from context.

Undirected Graphs. Suppose $\mathbf{W}$ is the symmetric matrix indicating the edge weights w of an undirected graph of size n, and $\mathbf{\Phi}$ is the diagonal matrix whose diagonal entries correspond to the vertex weights induced by w as described above. Then, the *normalized Laplacian* of $\mathbf{W}$ is $\boldsymbol{\mathcal{L}} := \mathbf{I}_n - \mathbf{\Phi}^{-\frac{1}{2}} \mathbf{W} \mathbf{\Phi}^{-\frac{1}{2}}$.

Multi-way partition expansion is considered in Lee et al. [13] by considering the following parameter. For $C \subseteq V$ and positive integer k, denote
$\rho_k(C) := \min\{\max_{i \in [k]} \rho(S_i) : S_1, S_2, \ldots, S_k$ are disjoint subsets of C$\}$.

The following fact relates the eigenvalues of $\boldsymbol{\mathcal{L}}$ with the multi-way partition expansion with respect to $\mathbf{W}$, which may contain self-loops.

Fact 1. *(Higher Order Cheeger's Inequality [13]) Given a symmetric matrix* $\mathbf{W}$ *indicating the non-negative edge weights of an undirected graph, suppose its normalized Laplacian* $\boldsymbol{\mathcal{L}}$ *as defined above has eigenvalues* $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$*. Then, for* $1 \leq k \leq n$*, we have:* $\frac{\lambda_k}{2} \leq \rho_k(V) \leq O(k^2) \cdot \sqrt{\lambda_k}$.

Chung's Approach [8] to Transition Matrices. Given a probability transition matrix $\mathbf{P}$ (which is a square matrix with non-negative entries such that every row sums to 1) corresponding to a random walk on vertex set V, and non-negative vertex weight $\boldsymbol{\phi}$, one can define edge weights $w : V \times V \rightarrow \mathbb{R}_+$ as $w(u,v) := \phi(u) \cdot P(u,v)$. (Observe that these edge weights induce vertex weights that are consistent with $\boldsymbol{\phi}$.)

One interpretation of Chung's approach is that the vertex weights $\boldsymbol{\phi}$ are chosen to be a stationary distribution of $\mathbf{P}$, i.e., $\boldsymbol{\phi}^{\mathrm{T}}\mathbf{P} = \boldsymbol{\phi}^{\mathrm{T}}$. Hence, the edge weights w satisfy the following *Eulerian property*: for any subset $S \subseteq V$, we have $w(\partial(S)) = w(\partial(\overline{S}))$, where $\overline{S} := V \setminus S$.

We can define edge weights $\widehat{w}$ for the (complete) undirected graph with vertex set V such that for $u \neq v$, $\widehat{w}(u,v) = \frac{1}{2}(w(u,v) + w(v,u))$, and each self-loop has the same weight in $\widehat{w}$ and w.

Because of the Eulerian property of w, it is immediate that for all $S \subseteq V$, $w(\partial(S)) = \widehat{w}(\partial(S))$, where $\partial(S)$ is interpreted according to the directed case on the left and to the undirected case on the right. Moreover, for all $u \in V$, $w(u) = \widehat{w}(u)$. Hence, as far as expansion is concerned, it is equivalent to consider the undirected graph with edge weights given by the matrix $\widehat{\mathbf{W}}$, for which the (higher-order) Cheeger Inequalities (as in Fact 1) can be readily applied.

Chung's approach can be applied to any stationary distribution $\boldsymbol{\phi}$ of $\mathbf{P}$, but special attention is paid to the case when $\mathbf{P}$ is irreducible, i.e., the edges corresponding to non-zero transition probabilities form a strongly connected graph on V. The advantages is that in this case, the stationary distribution is unique, and every vertex has non-zero probability.

In Section 3, we consider how to extend Chung's approach to the case where the underlying directed is not strongly connected. In Section 4, we consider the case when the edge weights $\boldsymbol{\phi}$ deviate (slightly) from a stationary distribution.

3 Directed Graphs with Multiple Strongly Connected Components

In a directed graph, we say that a strongly connected component C is a *sink*, if there is no edge leaving C. Otherwise, we say that it is a non-sink.

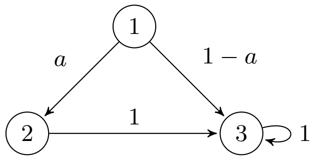
Even if the underlying directed graph of a given transition matrix is not strongly connected, Chung's approach [8] can still be applied if one chooses the vertex weights according to some stationary distribution.

However, under any stationary distribution, the weight on any vertex in a non-sink component must be zero. If we consider expansion using weights induced by a stationary distribution, essentially we are considering only the sink components. In this section, we explore if there is any meaningful way to consider expansion properties involving the non-sink components. As we shall see, it makes sense to consider the expansion properties of each strongly connected component separately.

3.1 Motivation for Considering Components Separately

Suppose $\mathbf{P}$ is a probability transition matrix corresponding to a random walk on some direted graph $G(V, E)$. Given a subset $C \subseteq V$, let $\mathbf{P}|_C$ be the square matrix restricting to the columns and the rows corresponding to C.

It is known [7, Theorem 3.22] that the eigenvalues of $\mathbf{P}$ are the union of the eigenvalues of $\mathbf{P}|_C$ over all strongly connected components C in a directed graph G. An important observation is that as long as the strongly connected components and the transition probabilities within a component remain the same, the eigenvalues of $\mathbf{P}$ are independent of the transition probabilities between different strongly connected components. For instance, the figure below depicts a directed graph, where the edges are labeled with the transition probabilities. Observe that each vertex is its own strongly connected component.



The transition matrix is:

$$\mathbf{P} = \begin{pmatrix} 0 & a & 1-a \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix},$$

whose eigenvalues are $\{0, 0, 1\}$, which are independent of the parameter a. This suggests that it might be difficult to use spectral properties to analyze expansion properties involving edges between different strongly connected components. Hence, we propose that the expansion of each connected component should be analyzed separately.

3.2 Defining Expansion via Diluted Stationary Distribution

Observe that if a strongly connected component C is a sink, then the transition matrix $\mathbf{P}|_C$ has a stationary distribution, and we can apply Chung's approach. However, if C is a non-sink, then not every row of $\mathbf{P}|_C$ sums to 1, and so $\mathbf{P}|_C$ has no stationary distribution.

However, since C is a strongly connected component, by the Perron-Frobenius Theorem [10], $\mathbf{P}|_C$ has a unique maximum eigenvalue $\lambda_{\max} \geq 0$ with algebraic and geometric multiplicity 1 such that every other eigenvalue (which might be complex) has magnitude at most $\lambda_{\max}$. Moreover,the associated eigenvector of $\lambda_{\max}$ has positive coordinates and is unique up to scaling. Suppose $\boldsymbol{\phi}$ is the (left) eigenvector which is normalized such that the coordinates sum to 1, i.e., $\sum_{u \in V} \phi(u) = 1$ and $\boldsymbol{\phi}^{\mathrm{T}} \mathbf{P}|_C = \lambda_{\max} \boldsymbol{\phi}$. We say that $\boldsymbol{\phi}$ is the *diluted stationary distribution* of $\mathbf{P}|_C$, because the distribution on vertices in C is diluted by a factor of $\lambda_{\max}$ after one step of the random walk.

We use the diluted stationary distribution $\boldsymbol{\phi}$ as vertex weights to define expansion $\rho(S)$ for $S \subseteq C$. Observe that the weight of edges leaving component C also contributes to the expansion.

3.3 Augmenting Graph to Achieve Stationary Distribution

Suppose the component C has size n, and has $\lambda_{\max} < 1$. In order to use Chung's approach, we construct an augmented graph $\widehat{G}$ consisting of the component C and an extra vertex v_0. For each $u \in C$, all the original probabilities leaking out of C from u are now directed to v_0. For the new vertex v_0, the transition probabilities from v_0 to vertices in C are given by the diluted stationary distribution $\boldsymbol{\phi}$. Hence, the augmented graph $\widehat{G}$ is strongly connected. We write $\mathbf{A} = \mathbf{P}|_C \in \mathbb{R}^{n \times n}$, and $\boldsymbol{\mu} = \mathbf{1}_n - \mathbf{A}\mathbf{1}_n \in \mathbb{R}^n$. The new transition matrix is

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} & \boldsymbol{\mu} \\ \boldsymbol{\phi}^{\mathrm{T}} & 0 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}.$$

Given a square matrix $\mathbf{M}$, its determinant is denoted as $|\mathbf{M}|$, and $eig(\mathbf{M})$ is the multi-set of its eigenvalues, which are roots of the polynomial $|\lambda\mathbf{I} - \mathbf{M}|$ in λ. We first show that the matrix $\mathbf{B}$ preserves the spectral properties of $\mathbf{A}$.

Lemma 1 (Spectral Preservation). *We have* $eig(\mathbf{B}) = eig(\mathbf{A}) - \{\lambda_{max}\} + \{1, \lambda_{max} - 1\}$. *Furthermore, if an eigenvalue* $\lambda \in eig(\mathbf{A}) - \{\lambda_{max}, \lambda_{max} - 1\}$, *it has the same geometric multiplicity in* $\mathbf{A}$ *and* $\mathbf{B}$.

Proof. To prove the first part, it suffices to show that for all $\lambda \in \mathbb{R}$,

$$\left|\lambda\mathbf{I}_n - \mathbf{A}\right| (\lambda - 1)(\lambda - (\lambda_{\max} - 1)) = \left|\lambda\mathbf{I}_{n+1} - \mathbf{B}\right| (\lambda - \lambda_{\max}),$$

because both sides are polynomials in λ of degree $n + 2$. Hence, they must be equivalent polynomials if they are equal for more than $n + 2$ values of λ.

If $\lambda = 1$, then the right hand side is zero because $\mathbf{B}$ has eigenvalue 1; similarly, if $\lambda = \lambda_{\max}$, the left hand side is zero because $\mathbf{A}$ has eigenvalue $\lambda_{\max}$.

For $\lambda \neq 1, \lambda_{\max}$, we have:

$$\left|\lambda\mathbf{I}_{n+1} - \mathbf{B}\right| = \begin{vmatrix} \lambda\mathbf{I}_n - \mathbf{A} & -\boldsymbol{\mu} \\ -\boldsymbol{\phi}^{\mathrm{T}} & \lambda \end{vmatrix} \cdot \begin{vmatrix} \mathbf{I}_n & \mathbf{1}_n \\ \mathbf{0}_n^{\mathrm{T}} & 1 \end{vmatrix} \tag{1}$$

$$= \begin{vmatrix} \lambda\mathbf{I}_n - \mathbf{A} & (\lambda - 1)\mathbf{1}_n \\ -\boldsymbol{\phi}^{\mathrm{T}} & \lambda - 1 \end{vmatrix} \tag{2}$$

$$= (\lambda - 1)\left|\lambda\mathbf{I}_n - \mathbf{A} - (\lambda - 1)\mathbf{1}_n(\lambda - 1)^{-1}(-\boldsymbol{\phi}^{\mathrm{T}})\right| \tag{3}$$

$$= (\lambda - 1)\left|\lambda\mathbf{I}_n - \mathbf{A} + \mathbf{1}_n\boldsymbol{\phi}^{\mathrm{T}}\right|, \tag{4}$$

where (1) follows because the second determinant is 1. Moreover, (2) follows from $|\mathbf{X}| \cdot |\mathbf{Y}| = |\mathbf{XY}|$. Equation (3) follows from the identity that for invertible $\mathbf{H}$,

$$\begin{vmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{vmatrix} = |\mathbf{H}| \cdot |\mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G}|.$$

Similarly, using $|\mathbf{X}| \cdot |\mathbf{Y}| = |\mathbf{XY}|$ repeatedly, we have

$$\begin{aligned}
\left|\lambda \mathbf{I}_n - \mathbf{A} + \mathbf{1}_n \boldsymbol{\phi}^{\mathrm{T}}\right| (\lambda - \lambda_{\max}) &= \begin{vmatrix} \mathbf{I}_n & -\mathbf{1}_n \\ \mathbf{0}_n^{\mathrm{T}} & 1 \end{vmatrix} \cdot \begin{vmatrix} \lambda \mathbf{I}_n - \mathbf{A} + \mathbf{1}_n \boldsymbol{\phi}^{\mathrm{T}} & \mathbf{0}_n \\ \boldsymbol{\phi}^{\mathrm{T}} & \lambda - \lambda_{\max} \end{vmatrix} \\
&= \begin{vmatrix} \mathbf{I}_n & \mathbf{0}_n \\ \frac{-\boldsymbol{\phi}^{\mathrm{T}}}{\lambda - \lambda_{\max}} & 1 \end{vmatrix} \cdot \begin{vmatrix} \lambda \mathbf{I}_n - \mathbf{A} & -(\lambda - \lambda_{\max}) \mathbf{1}_n \\ \boldsymbol{\phi}^{\mathrm{T}} & \lambda - \lambda_{\max} \end{vmatrix} \\
&= \begin{vmatrix} \lambda \mathbf{I}_n - \mathbf{A} & -(\lambda - \lambda_{\max}) \mathbf{1}_n \\ \mathbf{0}_n & \lambda - \lambda_{\max} + 1 \end{vmatrix} \\
&= \left|\lambda \mathbf{I}_n - \mathbf{A}\right| (\lambda - \lambda_{\max} + 1).
\end{aligned}$$

This completes the proof of the first part. To show that the geometry multiplicities of a common eigenvalue $\lambda \neq \lambda_{\max}, \lambda_{\max} - 1$ are equal, we show that $\mathbf{x} \longleftrightarrow \binom{\mathbf{x}}{0}$ is a bijection between $\mathbf{A}$ and $\mathbf{B}$'s corresponding right eigenvectors.

Since $\lambda \neq \lambda_{\max}$ is an eigenvalue of $\mathbf{A}$, if $\mathbf{x}$ is a corresponding eigenvector, then $\boldsymbol{\phi}^{\mathrm{T}}\mathbf{x} = 0$, because $\lambda_{\max}\boldsymbol{\phi}^{\mathrm{T}}\mathbf{x} = \boldsymbol{\phi}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \lambda\boldsymbol{\phi}^{\mathrm{T}}\mathbf{x}$. Hence, $B\binom{\mathbf{x}}{0} = \lambda\binom{\mathbf{x}}{0}$.

Conversely, suppose $\binom{\mathbf{x}}{y}$ is an eigenvector of $\mathbf{B}$ with eigenvalue λ, where $\mathbf{x} \in \mathbb{R}^n$ and $y \in \mathbb{R}$. We have

$$\begin{cases} \mathbf{A}\mathbf{x} + \boldsymbol{\mu} y = \lambda \mathbf{x} \\ \boldsymbol{\phi}^{\mathrm{T}}\mathbf{x} = \lambda y \end{cases} \quad \text{or} \quad \begin{cases} (\lambda \mathbf{I} - \mathbf{A})\mathbf{x} = \boldsymbol{\mu} y \\ \boldsymbol{\phi}^{\mathrm{T}}\mathbf{x} = \lambda y. \end{cases}$$

Then, $\boldsymbol{\phi}^{\mathrm{T}}(\lambda \mathbf{I} - \mathbf{A})\mathbf{x} = (\lambda - \lambda_{\max})\boldsymbol{\phi}^{\mathrm{T}}\mathbf{x} = (\lambda - \lambda_{\max})\lambda y$.

But we also have $\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\mu} y = \boldsymbol{\phi}^{\mathrm{T}}(\mathbf{1} - \mathbf{A}\mathbf{1}) y = (1 - \lambda_{\max}) y$. Since the two quantities are equal, this implies that $\lambda = 1, \lambda_{\max} - 1$ or $y = 0$. By assumption, $\lambda \neq 1, \lambda_{\max} - 1$, and so the only possibility is $y = 0$, then $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. This shows that λ has the same geometric multiplicity in $\mathbf{A}$ and $\mathbf{B}$. □

3.4 Higher-Order Cheeger Inequalities for Component

Given a non-sink component C, we have described how to add an extra vertex v_0 to construct an augmented graph $\widehat{G}$ with transition matrix $\mathbf{B}$. Observe that $\mathbf{B}$ has stationary distribution $\widehat{\boldsymbol{\phi}} = (\boldsymbol{\phi}, 1 - \lambda_{\max})$, where $\boldsymbol{\phi}$ is the diluted stationary distribution of $\mathbf{A}$. Hence, it follows that for all $S \subseteq C$, the old expansion $\rho(S)$ is the same as the new expansion $\widehat{\rho}(S)$ in the augmented graph.

Therefore, we can apply Chung's approach [8] and the spectral analysis by Lee et al. [13] to obtain the following lemma, which is a restatement of Theorem 1.

Lemma 2 (Cheeger Inequalities for Component C). *Suppose $\widehat{\mathbf{\Phi}}$ is the diagonal matrix whose diagonal entries are coordinates of the stationary distribution $\widehat{\boldsymbol{\phi}}$ of $\mathbf{B}$. Moreover, suppose the normalized Laplacian $\boldsymbol{\mathcal{L}}$ of the symmetric matrix $\frac{1}{2}(\widehat{\mathbf{\Phi}}\mathbf{B} + \mathbf{B}^T\widehat{\mathbf{\Phi}})$ has eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n+1}$. Then, for all $1 \leq k \leq n$, $\frac{\lambda_k}{2} \leq \rho_k(C) \leq O(k^2) \cdot \sqrt{\lambda_{k+1}}$.*

Proof. We use the following inequality from [13]:
$\frac{\lambda_k}{2} \leq \widehat{\rho}_k(\widehat{G}) \leq O(k^2) \cdot \sqrt{\lambda_k}$.

Observe that if $S \subseteq C$ does not contain the new vertex v_0, then S has the same expansion $\rho(S) = \widehat{\rho}(S)$ in both graphs.

From $k+1$ disjoint subsets in the augmented graph $\widehat{G}$, we can get at least k subsets of C by removing the one containing v_0. Hence, we have
$\rho_k(C) \leq \widehat{\rho}_{k+1}(\widehat{G}) \leq O((k+1)^2)\sqrt{\lambda_{k+1}}$.

On the other hand, k disjoint subsets in C are also disjoint in $\widehat{G}$. Therefore, we have $\rho_k(C) \geq \rho_k(\widehat{G}) \geq \frac{\lambda_k}{2}$, as required. □

4 Vertex Weights Deviate from Stationary Distribution

In this section, we consider a transition matrix $\mathbf{P}$ whose underlying directed graph $G(V, E)$ (where $n = |V|$) is not necessarily strongly connected. Moreover, each vertex has a positive weight given by a vector $\boldsymbol{\phi} \in \mathbb{R}^n$ that is not necessarily a stationary distribution of $\mathbf{P}$. We wish to analyze the expansion with respect to $\mathbf{P}$ and $\boldsymbol{\phi}$ using spectral techniques. As in Section 3, we shall add an extra vertex v_0 to form an augmented graph $\widehat{G}$.

We measure how much $\boldsymbol{\phi}$ deviates from a stationary distribution by the following parameter:

$$\varepsilon := 1 - \min_{u \in V} \frac{\phi(u)}{\sum_{v \in V} \phi(v) \cdot P(v, u)}.$$

A smaller value of ε means that $\boldsymbol{\phi}$ is closer to a stationary distribution. In particular, if $\varepsilon = 0$, then $\boldsymbol{\phi}$ is a stationary distribution.

Our idea is to first scale down all probabilities in $\mathbf{P}$ by a factor of $(1-\varepsilon)$. We add a new vertex v_0 to absorb the extra ε probability from each existing vertex. Then, we define the transition probabilities from v_0 to the original vertices carefully such that each original vertex u receives the same weight $\phi(u)$ after one step.

For each vertex u, the weight mass it obtains from vertices V through the scaled-down $\mathbf{P}$ is $(1-\varepsilon)\sum_{v \in V} \phi(v) P(v, u)$. Hence, the new vertex v_0 needs to return mass weights to vertices in V given by the vector $\mathbf{m} := \boldsymbol{\phi} - (1-\varepsilon)\mathbf{P}^{\mathrm{T}}\boldsymbol{\phi}$, whose coordinates are non-negative by the choice of ε. Normalizing by $\mathbf{m}^{\mathrm{T}}\mathbf{1}_n = \varepsilon\boldsymbol{\phi}^{\mathrm{T}}\mathbf{1}_n$, we have the vector $\boldsymbol{\mu} := \frac{\mathbf{m}}{\varepsilon\boldsymbol{\phi}^{\mathrm{T}}\mathbf{1}_n}$ of transition probabilities from v_0 to vertices in V.

The transition matrix of the augmented graph $\widehat{G}$ is

$$\widehat{\mathbf{P}} = \begin{pmatrix} (1-\varepsilon)\mathbf{P} & \varepsilon\mathbf{1}_n \\ \boldsymbol{\mu}^{\mathrm{T}} & 0 \end{pmatrix}.$$

Observe that $\widehat{G}$ is strongly connected, and its stationary distribution can be obtained by normalizing the vector $\widehat{\boldsymbol{\phi}} = (\boldsymbol{\phi}, \varepsilon\boldsymbol{\phi}^{\mathrm{T}}\mathbf{1}_n)$. In other words, we can append a new coordinate corresponding to v_0 to $\boldsymbol{\phi}$ to obtain a left eigenvector $\widehat{\boldsymbol{\phi}}$ with eigenvalue 1 for matrix $\widehat{\mathbf{P}}$.

Moreover, for each subset $S \subseteq V$ of the original vertices, the new expansion $\widehat{\rho}(S)$ with respect to $\widehat{\boldsymbol{\phi}}$ and $\widehat{\mathbf{P}}$ can be related to the old expansion $\rho(S)$ as follows: $\widehat{\rho}(S) = (1-\varepsilon) \cdot \rho(S) + \varepsilon$.

Hence, we can apply Chung's approach to $\widehat{\mathbf{P}}$ and $\widehat{\boldsymbol{\phi}}$ to construct a symmetric Laplacian matrix, whose eigenvalues are related to the expansion properties using the results by Lee et al. [13]. The following lemma is a restatement of Theorem 2, and its proof uses the same argument as in the proof of Lemma 2.

Lemma 3. *Suppose* $\widehat{\mathbf{\Phi}}$ *is the diagonal matrix whose diagonal entries are coordinates of* $\widehat{\boldsymbol{\phi}}$ *as defined above. Moreover, suppose the normalized Laplacian* $\boldsymbol{\mathcal{L}}$ *of the symmetric matrix* $\frac{1}{2}(\widehat{\mathbf{\Phi}}\widehat{\mathbf{P}} + \widehat{\mathbf{P}}^T\widehat{\mathbf{\Phi}})$ *has eigenvalues* $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n+1}$. *Then, for all* $1 \leq k \leq n$, $\frac{\lambda_k}{2} \leq (1-\varepsilon)\cdot\rho_k(V) + \varepsilon \leq O(k^2)\cdot\sqrt{\lambda_{k+1}}$.

References

1. Alon, N.: Eigenvalues and expanders. Combinatorica **6**(2), 83–96 (1986)
2. Alon, N., Kahale, N.: A spectral technique for coloring random 3-colorable graphs. SIAM J. Comput. **26**(6), 1733–1748 (1997)
3. Alon, N., Milman, V.D.: lambda, isoperimetric inequalities for graphs, and superconcentrators. J. Comb. Theory, Ser. B **38**(1), 73–88 (1985)
4. Aspvall, B., Gilbert, J.R.: Graph coloring using eigenvalue decomposition. SIAM Journal on Algebraic Discrete. Methods **5**(4), 526–538 (1984)
5. Brin, S., Page, L.: Reprint of: The anatomy of a large-scale hypertextual web search engine. Computer Networks **56**(18), 3825–3833 (2012)
6. Cheeger, J.: A lower bound for the smallest eigenvalue of the laplacian. Problems in analysis **625**, 195–199 (1970)
7. Chen, W.K., Lauwerier, H.A., Koiter, W.T.: Applied Graph Theory: Graphs and Electrical Networks. North-Holland Series in Applied Mathematics and Mechanics. Elsevier Science (2014)
8. Chung, F.: Laplacians and the cheeger inequality for directed graphs. Annals of Combinatorics **9**(1), 1–19 (2005)
9. Chung, F.R.K.: Spectral graph theory, vol. 92. American Mathematical Soc. (1997)
10. Horn, R.A., Johnson, C.R.: Matrix analysis. Cambridge University Press (1990)
11. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM **46**(5), 604–632 (1999)
12. Kwok, T.C., Lau, L.C., Lee, Y.T., Gharan, S.O., Trevisan, L.: Improved cheeger's inequality: analysis of spectral partitioning algorithms through higher order spectral gap. In: STOC, pp. 11–20 (2013)
13. Lee, J.R., Gharan, S.O., Trevisan, L.: Multiway spectral partitioning and higher-order cheeger inequalities. J. ACM **61**(6), 37 (2014)
14. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: analysis and an algorithm. In: NIPS, pp. 849–856 (2001)
15. Shewchuk, J.R.: Ladies and gentlemen, allow me to introduce spectral and isoperimetric graph partitioning (2011)
16. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000)
17. Tolliver, D., Miller, G.L.: Graph partitioning by spectral rounding: applications in image segmentation and clustering. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1053–1060 (2006)

Game Theory and Algorithms

Strategy-Proof Mechanism for Obnoxious Facility Location on a Line

Deshi Ye[1], Lili Mei[1], and Yong Zhang[2,3](✉)

[1] College of Computer Science, Zhejiang University, Hangzhou, China
{yedeshi,meilili}@zju.edu.cn

[2] Shenzhen Institutes of Advanced Technology,
Chinese Academy of Sciences, Beijing, China
zhangyong@siat.ac.cn

[3] Department of Computer Science, The University of Hong Kong, Hong Kong, China

Abstract. In the problem of obnoxious facility location, an obnoxious facility is located in an area. To maximize the social welfare, *e.g.*, the sum of distances from all the agents to the facility, we have to get the true locations of each agent. However, each agent may misreport his/her location to stay far away from the obnoxious facility. In this paper, we design strategy-proof mechanisms on locating an obnoxious facility on a real line. Two objective functions, *i.e.*, maximizing the sum of squares of distances (maxSOS) and maximizing the sum of distances (maxSum), have been considered. For maxSOS, a randomized strategy-proof mechanism with approximation ratio 5/3 is given, meanwhile the lower bound is proved to be at least 1.042. The lower bound of any randomized strategy-proof mechanisms w.r.t. maxSum is proved to be 1.077. Moreover, an extended model that each agent controls multiple locations is considered. For this model, we investigate deterministic and randomized strategy-proof mechanisms w.r.t. maxSum and maxSOS objectives, respectively. The deterministic mechanisms are shown to be tight for both objectives.

1 Introduction

We consider the problem of locating an obnoxious facility on a line, where a set of agents are located. In the view of algorithmic mechanism design, the agent's locations are private information and each agent attempts to maximize his/her utility, *i.e.*, stay far away from the obnoxious facility, by misreporting his/her location. Mechanisms receive the declaration of agents as input and determine the location. Our target is to design strategy-proof (or truthful) mechanisms to maximize social welfare, *i.e.*, the sum of squares of distances (maxSOS) or the sum of distances (maxSum).

In the classical facility location game, the utility of an agent is the distance from his/her true location to the nearest facility if there are multiple facilities,

This work is supported by NSFC (No. 61433012, U1435215, 11171086) Natural Science Foundation of Hebei A2013201218, Shenzhen basic research project (JCYJ20120615140531560), HKU small project funding 201309176064.

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 45–56, 2015.
DOI: 10.1007/978-3-319-21398-9_4

and each agent attempts to minimize his/her utility. In this work, we consider the obnoxious facility location problem, in which each agent attempts to maximize the utility, *i.e.*, stay as far as possible from the facility. These problems arise in many real applications, such as locating nuclear reactors, garbage dump sites, ammunition dumps, and polluting plants in a community.

To measure the performance of the mechanisms for classical facility location problem, the *approximation ratio* with respect to some given social objectives has been considered. The approximation ratio is defined to be the worst-case ratio between the mechanism's objective value and the optimal solution. There are mainly three objectives: *minSum* (minimizing the sum of agents' utilities), *minMax* (minimizing the maximum utility of agents), and *minSOS* (minimizing the sum of squares of agents' utilities). Feldman and Wilf [2] pointed out that the minSOS function is highly relevant in economic settings, the centroid in geometry, or the center of mass in physics. Accordingly, for obnoxious facility game, the social objectives are *maxSum* (maximizing the sum of agents' utilities), *maxMin* (maximizing the minimum utility of agents), and *maxSOS* (*i.e.*, maximizing the sum of squares of agents's utilities). Han and Du [4] mentioned that for obnoxious facility game, any deterministic mechanisms for maxMin objective are unbounded. Hence, in this paper, we only consider maxSum and maxSOS two objectives. Observe that for obnoxious facility problem, if the number of the facilities is more than one, locating all the facilities the same place is the best choice. Due to the above reasons, in this work, we only consider the single obnoxious facility game for maxSum and maxSOS objectives.

We then extend our model to a setting that each agent can control multiple locations and only one facility can be located. Procaccia and Tennenholtz [5] gave an example of real estate agents for the classical facility location game, which is also suitable for the obnoxious version. There are some other scenarios for the extended model. We can take one community or one company with many branches be an agent. For the extended model, we also consider deterministic and randomized strategy-proof mechanisms for maxSum and maxSOS objectives. And as noted by Schummer and Vohra [16], payment is not available in many real scenarios, especially in the social choice literature. Hence, we focus on the mechanism design without payment.

Related work. The classical setting of the facility location game is to minimize the utility function of each agent, where the utility function is the distance from agent's true location to the facility location. One direction of studying this setting is to investigate the characterizations of strategy-proof mechanisms. Moulin [14] and Schummer et al. [15] provided characterizations of deterministic strategy-proof mechanisms on line, tree, and cycle networks. Recently, Dokow et al. [10] gave a full characterization of strategy-proof mechanisms on line and on sufficiently large cycles on a discrete graph. Fotakis and Tzamos [11] gave a characterization of deterministic strategy-proof mechanisms with a bounded approximation ratio for 2-Facility on the line.

On the algorithmic view of the classical setting, Procaccia and Tennenholtz [5] first studied strategy-proof mechanisms with provable approximation ratios on the line for 1-Facility and 2-Facility for minSum and minMax objectives. Subsequently, Lu et al. [6,7] improved some results for 2-Facility on a line and circle under the minSum objective. Alon et al. [8] provided the approximation ratios achievable by randomized and deterministic mechanisms for 1-Facility in circle and general metrics under the minMax objective. Feldman and Wilf [2] studied the approximation ratios for randomized and deterministic mechanisms under the minSOS objective on a line. The extended model with multiple locations per agent was investigated in [5,7] for locating one facility on a line for minSum and minMax objectives. Thang [12], Fotakis and Tzamos [21] studied the k-Facility location problem.

For the setting of obnoxious facility game, the mechanism design with the social objective maxSum was first studied by Cheng et al. [1]. They presented a 3-approximation group strategy-proof deterministic mechanism and a lower bound of 2. They designed a randomized strategy-proof mechanism with 3/2-approximation. They also extended the work to tree and circle networks. Furthermore, for the general network, they proposed a 4-approximation group strategy-proof (GSP) deterministic mechanism and a 2-approximation group strategy-proof randomized mechanism, respectively.

On the direction of characterizing the mechanisms, Ibara and Nagamochi [3] first studied the characterization of deterministic strategy-proof mechanisms for obnoxious facility location game. In their paper, they showed there is no strategy-proof mechanism such that the number of candidates (locations output by the mechanism for some reported locations) is more than two. Then they characterized (group) strategy-proof mechanisms for two candidates in the general metric, in which a 4-approximation group strategy-proof mechanism in any metric was proposed. Recently, Han and Du [4] also investigated the characterization of deterministic strategy-proof mechanisms for the single-sinked policy domain. Moreover, they showed that any deterministic group strategy-proof mechanisms on the line have approximation ratio at least $(1+2^p)^{\frac{1}{p}}$ if the social objective function is L_p-norm, which implies lower bounds of 3 and 5 for maxSum and maxSOS objectives, respectively. Note that the lower bound of 3 is tight with the upper bound given in [1].

Our Contributions. In this paper, we provide approximation results for obnoxious facility game with respect to maxSum and maxSOS objectives on the line. We provide, for any randomized strategy-proof mechanisms, the lower bounds are 1.077 and 1.042 with respect to maxSum and maxSOS objective. We provide a strategy-proof randomized mechanism with approximation ratio 5/3.

We then extend the model such that each agent controls multiple locations. The utility of an agent is the sum of the distances from this agent's locations to the facility. For maxSum objective, we provide a deterministic strategy-proof mechanism with approximation ratio 3, which is tight with the lower bound [4]. In the case of randomized strategy-proof mechanism, we show a lower bound 10/9 and an upper bound 2. For maxSOS objective, the deterministic strategy-proof mechanism is 5 ,which matches the lower bound of 5 in [4]. For the randomized strategy-proof mechanisms, we show a lower bound of 1.13 and an upper bound of 4. A summary of our results are illustrated in Table 1.

Table 1. A summary of our results. UB and LB stands for upper bound and lower bound, respectively. SP and GSP represent strategy-proof and group strategy-proof, respectively. MultiLA stands for the model with multiple locations per agent.

Model	**Deterministic**		**Randomized**	
maxSum	UB: 3 GSP [1]	LB: 3 GSP [4]	UB: 3/2 [1]	LB: **1.077** SP
maxSOS	UB: 5 GSP [4]	LB: 5 GSP [4]	UB: **5/3** GSP	LB: **1.042** SP
MultiLA: maxSum	UB: **3** GSP	LB: 3 SP [4]	UB: **2** SP	LB: **10/9** SP
MultiLA: maxSOS	UB: **5** GSP	LB: 5 GSP [4]	UB: **4** SP	LB: **1.13** SP

2 Model and Preliminaries

The possible location area G is represented by a line. For obnoxious facility location problem, locating the facility at the infinity is the optimal location. Hence, we restrict the line to a closed interval. For simplicity, suppose G is within the interval $[0, 2]$, *i.e.*, $G = [0, 2]$. Let $N = \{1, 2, \ldots, n\}$ be a set of agents. Each agent i has an true location $x_i \in G$. We refer to $x = (x_1, x_2, \ldots, x_n) \in G^n$ as the location profile.

The utility of agent i is the distance from x_i to the facility. Each agent attempts to maximize his/her utility. The social welfare is maxSOS (the sum of squares of agents' utilities) or maxSum (the sum of agents' utilities). The goal of a mechanism is to provide a strategy-proof (or truthful) mechanism with the maximum social welfare.

A *deterministic mechanism* is a function $f : G^n \rightarrow G$ that maps the reported location profile to the location of a facility (which can be located anywhere in G). When the facility is located at $y \in G$, the utility of agent i is simply the distance between x_i and y, *i.e.*, $D(y, x_i) = |y - x_i|$, for all $i \in N$.

A *randomized mechanism* is a function $f : G^n \rightarrow \Delta G$ that maps the reported location profile to a probability distribution over G. If $f(x) = P$, where P is a probability distribution, agent i's utility is defined to be the expected distance between x_i and y, *i.e.*, $D(P, x_i) = E_{y \sim P}[|y - x_i|]$, for all $i \in N$.

A mechanism is *strategy-proof* if no agent can strictly increase his/her utility by misreporting the location, regardless of the declarations of other agents. Formally, given a location profile $x \in G^n$, and for all x'_i, we have $D(f(x_i, x_{-i}), x_i) \geq D(f(x'_i, x_{-i}), x_i)$, where $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ is the location profile excluding agent i.

A mechanism is said to be *group strategy-proof* (GSP) if no any set of agents can all benefit by misreporting their locations, regardless of the reports of the other agents. That is, given any location profile $x = (x_S, x_{-S})$ for any non-empty subset of agents $S \subseteq N$ and the misreported location x'_S, there exists $i \in S$ such that $D(f(x), x_i) \geq D(f(x'_S, x_{-S}), x_i)$.

Given a location profile x and the facility location y by a mechanism, the social welfare of the mechanism is defined by $SC(y, x)$. If the social objective function is maxSum, then $SC(y, x) = \sum_{i \in N} D(y, x_i) = \sum_{i \in N} |y - x_i|$. If the social objective function is maxSOS, then $SC(y, x) = \sum_{i \in N}(y - x_i)^2$. Moreover, the social welfare of a distribution P of profile x is $SC(P, x) = E_{y \sim P}[SC(y, x)]$.

We denote $SC(opt, x)$ to be the optimal social welfare with respect to the location profile x, *i.e.*, $SC(opt, x) = \max_y\{SC(y, x)\}$. A mechanism f is said to be ρ-approximation if $SC(opt, x) \leq \rho \cdot SC(f(x), x)$ for any location profile x.

Cheng et al. [1] pointed out that an optimal facility location will be at one of two extreme points 0 or 2 for the maxSum objective. In the following, we show that the property also holds for maxSOS objective.

Proposition 1. *Given a location profile x, at least one of 0 and 2 is an optimal location for maxSOS objective.*

Proof. Given a location profile x, suppose that the optimal facility location is at y, where $x_k \leq y < x_{k+1}$. If necessary, we add x_0 (or x_{n+1}) if y is located between 0 and x_1 (or x_n and 2). In the following, we show that if y is not located at 0 or 2, then we can move the facility to 0 or 2, the social welfare is not decreased, which therefore the proposition follows.

The social welfare of location y given the profile x is

$$SC(opt, x) = \sum_{i=1}^{k}(y - x_i)^2 + \sum_{i=k+1}^{n}(x_i - y)^2.$$

If $k \leq n/2$, let us consider the location at 0, otherwise the location is at 2. Due to the the symmetric, without loss of generality, we only consider the case when $k \leq n/2$.

The social welfare of location 0 is

$$\begin{aligned}
SC(0, x) &= \sum_{i=1}^{k}(x_i)^2 + \sum_{i=k+1}^{n}(x_i)^2 \\
&= \sum_{i=1}^{k}(y - x_i - y)^2 + \sum_{i=k+1}^{n}(x_i - y + y)^2 \\
&\geq SC(opt, x) - 2\sum_{i=1}^{k} y^2 + \sum_{i=1}^{k} y^2 + \sum_{i=k+1}^{n} y^2 \\
&\geq SC(opt, x).
\end{aligned}$$

□

To design strategy-proof mechanisms, we need the following notations in the remaining paper. Let $n_1 = |\{i : 0 \leq x_i \leq 1\}|$ be the number of agents whose declarations are in the interval $[0, 1]$. Let $n_2 = |\{i : 1 < x_i \leq 2\}|$ be the number of agents whose declarations are in the interval $(1, 2]$.

A deterministic strategy-proof mechanism was provided in [1] as below.

Mechanism 1: Set $f(x) = 0$ if $n_1 \leq n_2$ and otherwise $f(x) = 2$.

Mechanism 1 was proved to be strategy-proof in [1]. Han and Du [4] showed that Mechanism 1 achieves 5-approximation for maxSOS objective and the approximation ratio is the best possible.

3 Randomized Strategy-Proof Mechanisms

In this section, we study randomized strategy-proof mechanisms with respect to maxSOS objective function. We first show a family of group strategy-proof mechanisms with respect to n_1, the number of agents whose reported locations are within the interval $[0, 1]$.

Mechanism RM: For any given profile x, the mechanism places the facility at 0 with probability α, and places the facility at 2 with probability $1 - \alpha$, where $0 \le \alpha \le 1$ is a number that depends on x.

Lemma 1. *The mechanism RM is group strategy-proof if α is a non-increasing function on n_1. (The proof can be found in the full version of the paper.)*

Theorem 2. *Let $\alpha = \frac{4n_1 n_2 + n_2^2}{8n_1 n_2 + n_1^2 + n_2^2}$. Mechanism RM is group strategy-proof and its approximation ratio is at most $5/3$ for maxSOS objective.*

Proof. Let $\beta = \frac{n_1}{n_2} = \frac{n_1}{n - n_1}$. Then $\alpha = \frac{1+4\beta}{1+8\beta+\beta^2}$. It is easy to check that α is a non-increasing function on β and then a a non-increasing function on n_1. The group strategy-proof holds due to Lemma 1.

Let us consider the approximation ratio. To state conveniently, we let f denote Mechanism RM. For any given location profile $x = (x_1, x_2, \ldots, x_n)$, the social welfare of Mechanism RM is

$$SC(f(x), x) = \alpha \sum_{i=1}^{n} x_i^2 + (1 - \alpha) \sum_{i=1}^{n} (2 - x_i)^2.$$

From Proposition 1, the optimal facility location is either at 0 or 2. We estimate the following upper bounds for optimal social welfare.

$$\sum_{i=1}^{n} x_i^2 = \sum_{x_i \in [0,1)} x_i^2 + \sum_{x_i \in [1,2]} x_i^2 \le n_1 + 4n_2 \tag{1}$$

$$\sum_{i=1}^{n} (2 - x_i)^2 = \sum_{x_i \in [0,1)} (2 - x_i)^2 + \sum_{x_i \in [1,2]} (2 - x_i)^2 \le 4n_1 + n_2 \tag{2}$$

We deal with two cases with respect to the optimal location.

Case 1. The optimal location is at 0, *i.e.*, $SC(opt, x) = \sum_{i=1}^{n} x_i^2$. The social welfare of mechanism RM is

$$\begin{aligned} SC(f(x), x) &= \alpha SC(opt, x) + (1 - \alpha)\Big(\sum_{x_i \in [0,1)} (2 - x_i)^2 + \sum_{x_i \in [1,2]} (2 - x_i)^2\Big) \\ &\ge \alpha SC(opt, x) + (1 - \alpha) n_1 \\ &\ge \alpha SC(opt, x) + (1 - \alpha) \frac{n_1}{n_1 + 4n_2} SC(opt, x) \end{aligned} \tag{3}$$

Inequality (3) holds due to the inequality (1).

Case 2. The optimal location is at 2, *i.e.*, $SC(opt, x) = \sum_{i=1}^{n}(2 - x_i)^2$. The social welfare of mechanism RM is

$$\begin{aligned} SC(f(x), x) &= \alpha(\sum_{x_i \in [0,1)} x_i^2 + \sum_{x_i \in [1,2]} x_i^2) + (1-\alpha)\sum_{i=1}^{n}(2 - x_i)^2 \\ &\geq \alpha n_2 + (1-\alpha)SC(opt, x) \\ &\geq \alpha \frac{n_2}{n_1 + 4n_2} SC(opt, x) + (1-\alpha)SC(opt, x), \end{aligned} \quad (4)$$

Inequality (4) holds due to the inequality (2).

From Inequality (3) and Inequality (4), we obtain the approximation ratio ρ as below.

$$\rho \leq \max\{\frac{1}{\alpha + (1-\alpha)n_1/(n_1 + 4n_2)}, \frac{1}{\alpha n_2/(4n_1 + n_2) + (1-\alpha)}\} \quad (5)$$

The above approximation ρ reaches the maximum when $\alpha = \frac{4n_1n_2 + n_2^2}{8n_1n_2 + n_1^2 + n_2^2}$. Let $\beta = n_1/n_2$. Then $\rho = \frac{1+8\beta+\beta^2}{1+4\beta+\beta^2}$, and ρ reaches its maximum 5/3 if $\beta = 1$. □

3.1 Lower Bounds of Randomized Mechanisms

In this subsection, we first show that no randomized strategy-proof mechanism can achieve approximation ratio smaller than 1.077 for maxSum objective. Then, we extend the idea for designing a lower bound 1.04285 for any randomized mechanisms with respect to maxSOS objective.

Theorem 3. *The approximation ratio of any randomized strategy-proof mechanisms for maxSum objective is at least* $14/13 \approx 1.077$.

Proof. Consider two agents with a location profile $x = (1/3, 5/3)$. Then $SC(opt, x) = 2$. Let y be the facility location determined by a randomized mechanism.

We assume there exists a strategy-proof randomized mechanism with approximation ratio $\rho < 1.077$. It is worth to note that $SC(y, x) = E[|y - 1/3| + |y - 5/3|] \leq 2$. Without loss of generality, we assume $E[|y - 5/3|] \leq 1$.

Let us consider a new profile $x' = (1/3, 2)$, where agent 2 misreports its location to 2 other than 5/3. Let y' be the facility location of profile x'. Due to the strategy-proofness, then we obtain $E[|y' - 5/3|] \leq E[|y - 5/3|] \leq 1$.

Denote $Pr[y' < 1/3] = q$. To state simply, we let $E_{2,l}$ be $E[|y' - 5/3| : y' < 1/3]$ and $E_{2,r}$ be $E[|y' - 5/3| : y' \geq 1/3]$. Since the utility is nonnegative and the expected value is larger than the minimum value, we get that

$$\frac{4}{3}q \leq E[|y' - 5/3|] = E_{2,l} \cdot q + E_{2,r} \cdot (1 - q) \leq 1,$$

which therefore, $q \leq 3/4$.

In profile x', the optimal location is at 0. Hence, we have $SC(opt, x') = 7/3$. Note that $E[SC(y', x') : y' \geq 1/3] = 5/3$. Hence, we get an upper bound of any randomized mechanisms,

$$\begin{aligned} E[SC(y', x')] &= E[SC(y', x') : y' < 1/3]q + E[SC(y', x') : y' \geq 1/3](1-q) \\ &\leq \frac{7}{3}q + \frac{5}{3}(1-q). \end{aligned}$$

Then, the approximation ratio

$$\rho \geq \frac{7/3}{\frac{7}{3}q + \frac{5}{3}(1-q)} = \frac{7}{5+2q} \geq 14/13 \approx 1.077. \qquad \square$$

Theorem 4. *Any randomized strategy-proof mechanisms have an approximation ratio of at least* 1.04285 *for maxSOS objective.*

Proof. Consider a location profile with two agents $x = (x_1, x_2)$, where $x_1 = 1-a, x_2 = 1+a$ and $0 < a \leq 1$ is a constant that will be specified later. Then $SC(opt, x) = (1+a)^2 + (1-a)^2 = 2(1+a^2)$.

Since $E[(y-x_1)^2] + E[(y-x_2)^2] \leq SC(opt, x)$, w.l.o.g, we assume that $E[|y-x_2|^2] \leq (1+a^2)$. From Jensen's inequality, we have $(E[|y-x_2|])^2 \leq E[|y-x_2|^2] \leq (1+a^2)$, which gives $E[|y-x_2|] \leq \sqrt{1+a^2}$.

Consider a new location profile $x' = (1-a, 2)$. Let y' be the random variable of the facility location on profile x'. Due to the strategy-proofness, we have

$$E[|y'-x_2|] \leq E[|y-x_2|] \leq \sqrt{1+a^2}, \tag{6}$$

otherwise, agent 2 in profile x will lie to 2.

Let $Pr[y' < 1-a] = q$. Due to space constraints, we let $E_{2,l}$ be $E[|y'-x_2| : y' < 1-a]$ and $E_{2,r}$ be $E[|y'-x_2| : y' \geq 1-a]$. From Inequality (6), we have

$$2aq \leq E_{2,l} \cdot q \leq E[|y'-x_2|] = E_{2,l} \cdot q + E_{2,r} \cdot (1-q) \leq \sqrt{1+a^2}.$$

Thus, we get that $q \leq \frac{\sqrt{1+a^2}}{2a}$.

It is worth to note that the optimal location of profile x' is 0, which gives the optimal social welfare $SC(opt, x') = 4 + (1-a)^2$.

Now let us consider the social welfare for the profile x'. Due to space constraints, we let E_l be $E[SC(y', x') : y' < 1-a]$ and E_r be $E[SC(y', x') : y' \geq 1-a]$. We first consider the maximum value of social welfare if a facility $1-a \leq z \leq 2$ is located. We get that

$$\begin{aligned} SC(z, x') &= (z-(1-a))^2 + (2-z)^2 = 4 + (1-a)^2 + 2z^2 - 2(3-a)z \\ &\leq 4 + (1-a)^2 + 4(a-1) = SC(opt, x') + 4(a-1), \end{aligned} \tag{7}$$

where the last inequality holds since $1-a \leq z \leq 2$ and $0 < a \leq 1$.

Now we turn to the social welfare for the profile x'. The social welfare is

$$\begin{aligned} E[SC(y', x')] &= E_l \cdot q + E_r \cdot (1-q) \\ &\leq q \cdot SC(opt, x') + (1-q) \cdot (SC(opt, x') + 4(a-1)) \\ &= SC(opt, x') + 4(1-q)(a-1), \end{aligned}$$

where the inequality holds since $E_l \leq SC(opt, x')$ and Inequality (7).

Due to $0 < a \leq 1$, and $q \leq \frac{\sqrt{1+a^2}}{2a}$, it holds that

$$E[SC(y', x')] \leq 4 + (1-a)^2 + 2(1 - \frac{\sqrt{1+a^2}}{2a})(2a-2).$$

The approximation ratio of the randomized mechanism is

$$\rho \geq \frac{4 + (1-a)^2}{E[SC(y', x')]} \geq \frac{4 + (1-a)^2}{4 + (1-a)^2 + 2(1 - \frac{\sqrt{1+a^2}}{2a})(2a-2)}. \tag{8}$$

The right side of Inequality (8) reaches the maximum when $a = 0.758267$, then we finally get $\rho \geq 1.04285$ by setting $a = 0.758267$. □

4 Multiple Locations Per Agent

In this section, we consider an extended model that each agent controls multiple locations. Let w_i be the number of locations controlled by agent $i \in N$. The set of locations by agent i is then $x_i = (x_{i1}, x_{i2}, \ldots, x_{iw_i})$. The location profile is now $x = (x_1, \ldots, x_k)$, and there are total n locations, i.e $\sum_{i=1}^{k} w_i = n$.

As before, the utility of an agent i is the sum of utilities of each location, *i.e.*, $D(f(x), x_i) = \sum_{j=1}^{w_i} |f(x) - x_{ij}|$. For simplicity, let $\sum_i g(x_i)^p = \sum_i \sum_{j=1}^{w_i} g(x_{ij})^p$ for any $p \geq 1$, and any function g. In the remaining paper, the function g usually refers to $g(x_i) = x_i$ or $g(x_i) = 2 - x_i$.

4.1 Deterministic Strategy-Proof Mechanisms for Multiple Locations Per Agent

It is quite natural to consider Mechanism 1 for this model. However, Mechanism 1 is not strategy-proof.

Lemma 2. *Mechanism 1 is not strategy-proof for the multiple locations per agent. (The proof can be found in the full version of the paper.)*

Mechanism 1 is not strategy-proof since n_1 and n_2 only represent the independent locations without considering the agents. The reason motivates us to consider the locations of an agent as an entirety. An agent is said to *prefer* 0 if $D(0, x_i) \geq D(2, x_i)$.

Given profile x, let $N_1(x)$ be the set of agents who prefer 0 and $N_2(x)$ be the set of agents who prefer 0. Hence, $|N_1(x)|$ is the number of agents who prefer 0 and $|N_2(x)|$ is the number of agents who prefer 2.

It is quite natural to consider the mechanism who locates the facility at 0 if more agents prefer 0 than 2.

Mechanism MA:
For given location profile x, if $|N_1(x)| \leq |N_2(x)|$, the mechanism returns location 2, otherwise places the facility at 0.

Lemma 3. *Mechanism MA is a group strategy-proof mechanism. (The proof can be found in the full version of the paper.)*

Theorem 5. *Mechanism MA is a group strategy-proof mechanism, and its approximation ratio is $\Theta(n)$ for maxSOS objective. (The proof can be found in the full version of the paper.)*

Similar as Theorem 5, we obtain that the lower bound of Mechanism MA is $2n-1$ and the upper bound is at most $2n-1$. Thus, we have the following Corollary.

Corollary 1. *Mechanism MA is a group strategy-proof mechanism, and its approximation ratio is $\Theta(n)$ for maxSum objective.*

Though Mechanism MA is group strategy-proof, the approximation ratio is not constant. To get better approximation ratio, we provide the following mechanism.

Mechanism TMA:
For given location profile x, the mechanism places the facility at 0 if $\sum_{i\in N_1(x)} w_i \geq \sum_{i\in N_2(x)} w_i$, otherwise the facility is allocated at 2.

Theorem 6. *Mechanism TMA is a strategy-proof mechanism, and its approximation ratio is 3 for maxSum objective and 5 for maxSOS objective. (The proof can be found in the full version of the paper.)*

Remark: Han and Du [4] showed that the lower bounds of the general setting where each agent only has one location are 3 and 5 for maxSum and maxSOS objectives, respectively. The lower bounds are also valid for our models, and thus our mechanism is tight for both objectives.

4.2 Randomized Strategy-Proof Mechanisms for Multiple Locations Per Agent

In this subsection, we study randomized mechanisms for multiple locations per agent model. We first consider the mechanism that choose only location 0 and 2 with positive probabilities. The mechanism is described as below.

Mechanism RMA:
Given a location profile x, the mechanism places the facility at 0 with probability $\sum_{i\in N_1(x)} w_i/n$ and selects the facility at 2 with probability $\sum_{i\in N_2(x)} w_i/n$.

Theorem 7. *Mechanism RMA is a strategy-proof mechanism and its approximation ratio is 4 for maxSOS objective. (The proof can be found in the full version of the paper.)*

Corollary 2. *Mechanism RMA is a strategy-proof mechanism and its approximation ratio is 2 for maxSum objective. (The proof can be found in the full version of the paper.)*

4.3 Lower Bounds of Randomized Mechanisms for Multiple Locations Per Agent

The lower bounds of randomized mechanisms in Section 3 are still valid for this general model. However, we can improve the lower bounds due to multiple locations of each agent. The techniques of designing lower bounds are also different from Section 3.

Theorem 8. *Any randomized strategy-proof mechanisms with multiple locations per agent achieve an approximation ratio at least* 10/9 *for maxSum objective. (The proof can be found in the full version of the paper.)*

Theorem 9. *The approximation ratio of any randomized strategy-proof mechanisms for multiple locations per agent are at least* 1.13 *for maxSOS objective. (The proof can be found in the full version of the paper.)*

5 Concluding Remarks

In this paper, we have studied strategy-proof mechanisms for different objective functions of obnoxious facility location game on a real line. Both upper bounds and lower bounds are provided for randomized mechanisms. Besides, we studied an extended model that each agent controls multiple locations. The provided deterministic mechanisms are the best possible.

There exist plentiful researches on facility location and obnoxious facility location (*e.g.* surveys [18,19]). Tamir [20] considered k-maxMin and k-maxSum obnoxious facility location problem on graph, and showed that the problem is strongly NP-hard even the graph is a line. Berman and Drezner [17] studied the obnoxious facility on a network to maximize the minimal distance. This substantial work motivates us to investigate a lot of interesting work for future. First, one may consider to extend our model to different networks, such as a tree or a circle. Second, it is interesting to study two facilities or the general k facilities location games.

References

1. Cheng, Y., Yu, W., Zhang, G.: Strategy-proof approximation mechanisms for an obnoxious facility game on networks. Theoretical Computer Science **35**(3), 513–526 (2011)
2. Feldman, M., Wilf, Y.: Strategyproof facility location and the least squares objective. In: Proceedings of the 14th ACM Conference on Electronic Commerce (EC 2013), pp. 873–890 (2013)
3. Ibara, K., Nagamochi, H.: Characterizing mechanisms in obnoxious facility game. In: Lin, G. (ed.) COCOA 2012. LNCS, vol. 7402, pp. 301–311. Springer, Heidelberg (2012)
4. Han, Q., Du, D.: Moneyless strategy-proof mechanism on single-sinked policy domain: characterization and applications, Issue 2012, Part 8. Working paper series (University of New Brunswick, Faculty of Business Administration)

5. Procaccia, A.D., Tennenholtz, M.: Approximate mechanism design without money. In: Proceedings of the 10th ACM conference on Electronic Commerce (EC 2009), pp. 177–186 (2009)
6. Lu, P., Sun, X., Wang, Y., Zhu, Z.A.: Asymptotically optimal strategy-proof mechanisms for two-facility games. In: Proceedings of the 11th ACM conference on Electronic Eommerce (EC 2010), pp. 315–324 (2010)
7. Lu, P., Wang, Y., Zhou, Y.: Tighter bounds for facility games. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 137–148. Springer, Heidelberg (2009)
8. Alon, N., Feldman, M., Procaccia, A.D., Tennenholtz, M.: Strategyproof approximation of the minimax on networks. Mathematics of Operations Research **35**(5), 513–526 (2010)
9. Nissim, K., Smorodinsky, R., Tennenholtz, M.: Approximately optimal mechanism design via differential privacy. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS 2012), pp. 203–213 (2012)
10. Dokow, E., Feldman, M., Meir, R., Nehama, I.: Mechanism design on discrete lines and cycles. In: Proceedings of the 13th ACM Conference on Electronic Commerce (EC 2012), pp. 423–440 (2012)
11. Fotakis, D., Tzamos, C.: On the power of deterministic mechanisms for facility location games. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 449–460. Springer, Heidelberg (2013)
12. Thang, N.K.: On (group) strategy-proof mechanisms without payment for facility location games. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 531–538. Springer, Heidelberg (2010)
13. Gibbard, A.: Manipulation of voting schemes: a general result. Econometrica: Journal of the Econometric Society, 587–601 (1973)
14. Moulin, H.: On strategy-proofness and single peakedness. Public Choice **35**(4), 437–455 (1980)
15. Schummer, J., Vohra, R.V.: Strategy-proof location on a network. Journal of Economic Theory **104**(2), 405–428 (2002)
16. Schummer, J., Vohra, R.V.: Mechanism design without money. In: Algorithmic Game Theory, chap. 10, pp. 243–299, Cambridge (2007)
17. Berman, O., Drezner, Z.: A note on the location of an obnoxious facility on a network. European Journal of Operational Research **120**(1), 215–217 (2000)
18. Erkut, E., Neuman, S.: Analytical models for locating undesirable facilities. European Journal of Operational Research **40**(3), 275–291 (1989)
19. ReVelle, C., Eiselt, H.: Location analysis: A synthesis and survey. European Journal of Operational Research **165**(1), 1–19 (2005)
20. Tamir, A.: Obnoxious facility location on graphs. SIAM Journal on Discrete Mathematics **4**(4), 550–567 (1991)
21. Fotakis, D., Tzamos, C.: Strategyproof facility location for concave cost functions. In: Proceedings of the 14th ACM Conference on Electronic Commerce (EC 2013), pp. 435–452 (2013)

Bin Packing Game with an Interest Matrix

Zhenbo Wang[1], Xin Han[2(✉)], György Dósa[3], and Zsolt Tuza[4,5]

[1] Department of Mathematical Sciences, Tsinghua University, Beijing 100084, China
zwang@math.tsinghua.edu.cn

[2] Software School, Dalian University of Technology, Dalian 116620, China
hanxin.mail@gmail.com

[3] Department of Mathematics, University of Pannonia, Veszprém 8200, Hungary
dosagy@almos.vein.hu

[4] Department of Computer Science and Systems Technology, University of Pannonia, Veszprém, Hungary

[5] Alfréd Rényi Institute of Mathematics, Budapest, Hungary
tuza@dcs.uni-pannon.hu

Abstract. In this paper we study a game problem, called bin packing game with an interest matrix, which is a generalization of all the currently known bin packing games. In this game, there are some items with positive sizes and identical bins with unit capacity as in the classical bin packing problem; additionally we are given an interest matrix with rational entries, whose element a_{ij} stands for how much item i likes item j. The payoff of item i is the sum of a_{ij} over all items j in the same bin with item i, and each item wants to stay in a bin where it can fit and its payoff is maximized. We find that if the matrix is symmetric, a pure Nash Equilibrium always exists. However the *PoA* (Price of Anarchy) may be very large, therefore we consider several special cases and give bounds for *PoA* in them. We present some results for the asymmetric case, too.

1 Introduction

In the bin packing problem, there are n items with positive rational sizes $\{s_1, s_2, ..., s_n\}$, where each item has size at most 1, and infinitely many bins with unit capacity are available. The goal is to pack the items into a minimum number of bins, so that in any bin B_k, the sum of the sizes of the items being packed there (called level of the bin) does not exceed the capacity of the bin; i.e., the quantity $s(B_k) = \sum_{i \in B_k} s_i$ is at most 1. There are many papers on this topic; we refer to [2,3,6,7,9] for details.

The first bin packing game was introduced by Bilò [1]. Later another version was proposed by Ma et al. [10], and recently a general version was developed by

Z. Wang—Partially supported by NSFC No. 11371216.

X. Han—Partially supported by NSFC(11101065), LJQ2012003, RGC(HKU716-412E) and "the Fundamental Research Funds for the Central Universities".

G. Dósa, Z. Tuza—Partially supported by TÁMOP-4.2.2.A-11/1/KONV-2012-0072.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 57–69, 2015.
DOI: 10.1007/978-3-319-21398-9_5

Dósa and Epstein [4]. We call these bin packing games (or models) as BPG1, BPG2, and BPG3, respectively.

In case of the BPG1 model, the items in the same bin pay unit cost in total for being in this bin. The items share the cost proportionally to their sizes: a bigger item pays more, a smaller item pays less, i.e. an item with size s_i pays $s_i/s(B_k)$ for being in bin B_k.

In case of the BPG2 model, the cost of any bin is again unit, but the items of any bin pay the same price for being in this bin, i.e. any item pays $1/k$ if there are k items in the bin.

In case of model BPG3, each item i has two parameters s_i and u_i, where s_i is the size of the item (as usually), and a nonnegative weight u_i is also specified for item i. Then, for being in any one bin B_k, the items in B_k pay proportionally to their weights rather than to their sizes or their cardinality; i.e., item i pays u_i/U_k cost for being in B_k, where $U_k = \sum_{i \in B_k} u_i$. This is a common generalization of the two previous models BPG1 and BPG2, since if $u_i = s_i$ for any item, we get model BPG1, or if $u_i = 1$ for any item, we get model BPG2.

Generalized Bin Packing Game. We introduce a new type of bin packing games. This new game is a common generalization of all the above three models, thus we call it Generalized Bin Packing Game and abbreviate it as GBPG for short.

The motivation of this new model is to express that people make their decisions not only considering money or cost, but they often also take into account how much they like a certain situation. Let us consider the next simple example: There is a party where the people sit down at tables (tables = bins). Then a person is interested not only in the cost of sitting at some table (and paying for the food and drinks he will have), but would also like to enjoy the party, and therefore chooses a table where he/she finds the people appealing.

Formally, an instance $\mathcal{I} = (A, S)$ of GBPG is given as follows. There are n items with sizes $S = \{s_1, s_2, ..., s_n\}$, where $0 < s_i \leq 1$, and an $n \times n$ rational matrix $A = [a_{ij}]$, called the interest matrix, is also given. The payoff of item i is $p_i = \sum_{j \in B_k} a_{ij}$ if i is packed into bin B_k. Each item wants to stay in a bin where it can fit and its payoff is maximized. All bins are assumed to be identical with unit capacity. In the discussion below we assume that all $a_{ij} \geq 0$, although some facts remain valid for negative values, too. (Some remarks of this kind will be given.) We note that also a_{ii} is taken into account when defining the payoff p_i of item i.

A packing of the items is called a Nash Equilibrium [11], or NE for short, if no item can improve its payoff by moving to another bin in which it can fit. Moreover, if all the items are packed into the minimum number of bins, we call this packing an optimum packing, and denote it by OPT. Without the danger of confusion, we also use OPT and NE to denote the bins used by an OPT packing and an NE packing, respectively.

Price of Anarchy (PoA). An often used metric in case of bin packing games is the Price of Anarchy (PoA, for short); it measures how large a NE can be compared to OPT, when the value of OPT gets large. More exactly,

$$PoA = \lim_{k\to\infty} \sup\left\{ \frac{NE}{OPT} \mid OPT = k \right\}.$$

There are further metrics, too, such as price of stability (PoS), strong price of anarchy, and so on; in this paper we deal only with PoA.

Previous Results. The bin packing game BPG1 was studied first by Bilò in 2006 [1]. He proved that this game admits a NE, and that the PoA is in the interval $[1.6, 1.667]$. Epstein and Kleiman [8] obtained stronger estimates, proving that the PoA is in $[1.6416, 1.6428]$.

For the BPG2 model it was proven that its PoA is at most 1.7. This result got further improved in [5].

In case of model BPG3 [4], it was shown that many kinds of Nash equlibria (NE, Strong NE, Strongly Pareto Optimal NE and Weakly Pareto Optimal NE) exist. For the case of unit weights (which is equivalent to model BPG2), the PoA is in [1.6966, 1.6994]; and for the weighted case, both of the lower and upper bounds are 1.7. For other results, we refer to [5,13].

Our Contribution. When the interest matrix is symmetric, we prove that there exists a NE for any instance. Generally, the value of PoA can be very large, therefore we consider several specific types of the interest matrix $[a_{ij}]$. The results are listed in the following table, where s_i is the size of item i and u_i is the weight of item i. The bounds for $\max\{s_i, s_j\}$ are quoted from [8], and the lower bound in the last two columns is from [4].

$a_{ij} =$	general	$\max\{s_i, s_j\}$	$\min\{s_i, s_j\}$	$\max\{u_i, u_j\}$	$u_i + u_j$
PoA	∞	[1.6416, 1.6428]	∞	[1.6966, 1.723]	[1.6966, 2]

Lastly we investigate the case of asymmetric matrices and find that a NE packing does not exist for some instances. We give a sufficient condition to recognize this situation of non-existing NE. On the other hand we give a sufficient condition which guarantees that an asymmetric interest matrix can be converted to a symmetric one.

2 Preliminaries

In this section, we first recall the definition of a classical algorithm for the bin packing problem, and then prove that the game we study is a generalization of all the known bin packing games.

First Fit for Bin Packing. For an input I of bin packing, let $ALG(I)$ be the number of bins used by algorithm ALG to pack this input, and let $OPT(I)$ denote an optimal solution. Algorithm First Fit (FF) is a classical algorithm, which packs each item into the first bin where it fits. (If the item does not fit into any opened bin, it is packed into a new bin.) First Ullmann [12] proved that

$FF(I) \leq 1.7 \cdot OPT(I) + 3$. Then, after several attempts to decrease the additive constant, finally Dósa and Sgall [6] proved that $FF(I) \leq 1.7 \cdot OPT(I)$, what means that the absolute approximation ratio of FF is at most 1.7. In another work, Dósa and Sgall [7] give a matching lower bound, thus these two papers together prove that the bound 1.7 is tight.

Relation to the Earlier Models. The players in the games introduced earlier wish to minimize their costs, while in our game the players wish to maximize their payoff. In spite of this, the former bin packing game models can be considered as special cases of Model GBPG, in the following way:

– BPG1 model: let $a_{ij} = s_i \cdot s_j$; or $a_{ij} = s_j$.
– BPG2 model: let $a_{ij} = 1$; or $a_{ij} = s_i$.
– BPG3 model: let $a_{ij} = u_i \cdot u_j$; or $a_{ij} = u_j$.

We only prove this claim for the BPG1 model; the assertion for the other two models can be shown in a similar way.

Lemma 1. *If $a_{ij} = s_i \cdot s_j$, or if $a_{ij} = s_j$, for all $1 \leq i, j \leq n$, then our game is equivalent to the BPG1 model in the sense that a NE for GBPG is also a NE for BPG1, and vice versa.*

Proof. Suppose $a_{ij} = s_i \cdot s_j$, and an item i is packed into bin B_k. Then the payoff of this item is $p_i = \sum_{j \in B_k} (s_i \cdot s_j) = s_i \cdot s(B_k)$. This item is intended to go into another bin B_l, if the payoff of item i will be bigger there (and item i fits there). This new payoff is $p'_i = \sum_{j \in B_l} (s_i \cdot s_j) + s_i^2 = s_i \cdot (s(B_l) + s_i)$. Thus the movement is possible if and only if $p'_i > p_i$, i.e. $s(B_l) + s_i > s(B_k)$. This is exactly the same case (when the movement is possible) like the one in model BPG1. $\square$

3 The Symmetric Case

We prove that for any symmetric matrix A, there always exists a pure NE for our model GBPG. We give the proof by using a potential function.

Theorem 1. *If matrix A is symmetric, then GBPG always has an NE.*

Proof. The high-level idea is to associate each feasible packing with a potential in such a way that the potential function is upper-bounded by a value computable from the input, and to prove that once an item moves from one bin to another, the total potential strictly increases. If this property holds, then no previous state can occur again, thus a NE surely exists because there are only a finite number of different configurations (packings).

Recall that if item i is packed into bin B_k, then its payoff is $p_i = \sum_{j \in B_k} a_{ij}$. Given an input I and a packing for I, we define a potential function as

$$P = \sum_{i \in I} p_i \leq n^2 \max_{i,j} \{a_{ij}\}.$$

Next we prove that if item i moves from bin B_k to bin B_h, then the value of P increases. Before moving from bin B_k to bin B_h, let p_j be the payoff of item j and $P = \sum_{i \in I} p_i$. After the movement, let $p_j^{'}$ be the payoff of item j and $P^{'} = \sum_{i \in I} p_i^{'}$, and name bins B_k and B_h as $B_k^{'}$ and $B_h^{'}$, respectively.

Observe that for item j which is not packed in bin B_k or B_h, its payoff does not change. Then we have

$$\begin{aligned}
P^{'} - P &= \sum_{j \in B_k^{'}} p_j^{'} + \sum_{j \in B_h^{'}} p_j^{'} - \sum_{j \in B_k} p_j - \sum_{j \in B_h} p_j \\
&= (\sum_{j \in B_h^{'}} p_j^{'} - \sum_{j \in B_h} p_j) + (\sum_{j \in B_k^{'}} p_j^{'} - \sum_{j \in B_k} p_j) \\
&= (p_i^{'} + \sum_{j \in B_h} a_{ji}) - (p_i + \sum_{j \in B_k^{'}} a_{ji}) \\
&= (p_i^{'} - p_i) + (\sum_{j \in B_h} a_{ij} - \sum_{j \in B_k^{'}} a_{ij}) \quad \text{by } a_{ij} = a_{ji} \\
&= (p_i^{'} - p_i) + (a_{ii} + \sum_{j \in B_h} a_{ij} - a_{ii} - \sum_{j \in B_k^{'}} a_{ij}) \\
&= (p_i^{'} - p_i) + (p_i^{'} - p_i) = 2(p_i^{'} - p_i) > 0.
\end{aligned}$$

For the convergence steps, if we set A a rational matrix, let $\Delta > 0$ be the minimal integer such that Δa_{ij} is integer for all components of A, and therefore $\Delta(p_i^{'} - p_i) \geq 1$, and the potential function will increase at least $2/\Delta$ after a selfish movement. After at most $\Delta n^2 \max_{i,j}\{a_{ij}/2\}$ steps, we will have a NE. □

Remark 1. One can observe that the above proof works even when matrix A has zero or negative entries. A natural interpretation of this extension is that a_{ij} is positive if person i likes person j, and is negative if i dislikes j.

Observation 1. *For arbitrarily large real k there exists an interest matrix A, for which the PoA is bigger than k, even if $a_{ij} \in \{0, 1\}$ is required for all $1 \leq i, j \leq n$.*

Proof. Given any real k, let n be an integer such that $n > k$. There are n items $\{1, 2, ..., n\}$, each with size $\epsilon \leq 1/n$. We pack the items into mutually distinct bins. Let $a_{ii} = 1$ for every i, and $a_{ij} = 0$ for all $i \neq j$. The actual packing is a NE, whereas the optimal solution uses only one bin. □

Recall that if all entries in the interest matrix A have the same value $a_{ij} = 1$, the PoA is upper-bounded by 1.6994 [4]. However, we find that the PoA can be very large even if almost all entries are $a_{ij} = 1$ and all the other elements satisfy $a_{ij} = 1 - \epsilon$ where $\epsilon > 0$ could be arbitrarily small.

Proposition 1. *Given $0 < \delta < 1/2$, let k be an integer for which $k\delta > 1$. Then there is a matrix A with size $n \times n$, where $n = k^4$, in which $a_{ij} = 1$ for at least $(1 - 1/k) \cdot n^2$ different pairs (i, j), moreover $a_{ij} = 1 - \delta$ for all the other entries, and the PoA is at least k^2.*

The proof is left in the Appendix due to the page limitation.

3.1 Special Models

Now we give lower and upper estimates on PoA for several special cases of GBPG: $a_{ij} = \max\{s_i, s_j\}$, $a_{ij} = \min\{s_i, s_j\}$, $a_{ij} = \max\{u_i, u_j\}$, and $a_{ij} = u_i + u_j$, where u_i is the weight of item i, which may be different from s_i.

In the model $a_{ij} = \max\{u_i, u_j\}$, where u_i means something that how much person or item i is important. So a special (and natural) case is, where the items correspond to some persons, some of them are famous while the other are not famous. If we assume that $u_i = 1$ if any person i is famous, and $u_i = p < 1$ otherwise, we model the situation that people like to be in the presence of famous, or important persons. Or, more generally, any person gets an "importance" index, this is the u_i value. Then, the happiness between two persons, i and j is defined as $a_{ij} = \max\{u_i, u_j\}$. Further more in the special model $a_{ij} = \max\{s_i, s_j\}$, the bigger item can enforce his want to the smaller item. This is a typical situation in many cases.

In order to explore the relationship between a matrix A and the corresponding value of PoA, we begin with the settings $a_{ij} = max\{s_i, s_j\}$ and $a_{ij} = \min\{s_i, s_j\}$. For the former we prove that the PoA is at most 1.7, and find that some earlier results also remain valid for this model. But the latter is substantially different as the PoA can be arbitrarily large.

Proposition 2. *If $a_{ij} = \max\{s_i, s_j\}$, then PoA is at most 1.7.*

Proof. Our key observation is as follows: For any bin in a given packing, the payoff of the smallest item is the total size of items in the bin, and the payoff of any other item in the bin is at least this value. Consider a NE with bins $B_1, B_2, ..., B_m$. Assume that the bins are sorted such that $s(B_1) \geq s(B_2) \geq ... \geq s(B_m)$. We claim that no item in B_k fits into B_h for any $h < k$, i.e. the packing can be viewed as a result of FF packing. Let i be the smallest item in B_k, and suppose for a contradiction that it fits into B_h. In B_k, the payoff of item i is $p_i = s(B_k)$, whereas the payoff of this item is at least $s(B_h) + s_i > s(B_k)$ if it moves to B_h; this contradicts the assumption that we are in a NE state. Thus the claim follows. As we know that the asymptotic approximation ratio of FF (and even the absolute approximation ratio of FF) is 1.7, we obtain that the PoA in the current model is at most 1.7. □

Remark 2. We find that using the methods in [1], one can get $1.6 \leq PoA \leq 1.667$; and using the methods in [8], one can further get $1.6416 \leq PoA \leq 1.6428$.

Proposition 3. *If $a_{ij} = \min\{s_i, s_j\}$, then PoA can be arbitrarily large.*

The proof is left in the appendix.

Theorem 2. *Assume that each item i is associated with two parameters, the size s_i and the weight $u_i > 0$. If $a_{ij} = \max\{u_i, u_j\}$, then PoA is at most $\frac{31}{18} < 1.723$.*

Proof. Let us consider an NE with bins $B_1, B_2, ..., B_m$. Given a bin B_j, let the total weight and total size of the items in B_j be $u(B_j)$ and $s(B_j)$, respectively; and let the number of items in B_j be $|B_j|$. Suppose that there are two bins, say B_1 and B_2, such that $s(B_1) + s(B_2) \leq 1$. Assume without loss of generality that $u(B_1) \geq u(B_2)$. Let i be the item with the smallest weight in B_2. The payoff of i is exactly $u(B_2)$, and it becomes at least $u(B_1) + u_i > u(B_2)$ if the item moves to B_1, contradicting the assumption that the packing is an NE. Consequently, $s(B_1) + s(B_2) > 1$ holds for any two bins. Moreover, we have the following properties.

1. If item i is packed into B_j, its payoff satisfies $p_i \geq u(B_j)$ and equality holds only if i has the minimum weight in B_j.
2. If item k has the smallest or second smallest weight in B_j, then $p_k < u(B_j) + u_k$.

Now we divide the bins into four groups:

$$G_1 = \{B_j \mid 0 < s(B_j) \leq 1/2\}, \quad G_2 = \{B_j \mid 1/2 < s(B_j) \leq 2/3\},$$
$$G_3 = \{B_j \mid 2/3 < s(B_j) \leq 3/4\}, \quad G_4 = \{B_j \mid s(B_j) > 3/4\}.$$

Claim 1. $|G_1| \leq 1$.

Proof: This follows from the fact that, as we have shown the beginning of the argument, any two bins satisfy $s(B_1) + s(B_2) > 1$ in any NE. □

$$\text{Define} \quad G_2^1 = \{B_j \mid B_j \in G_2, |B_j| = 1\}, \quad G_3^1 = \{B_j \mid B_j \in G_3, |B_j| = 1\},$$
$$G_2^{2+} = \{B_j \mid B_j \in G_2, |B_j| \geq 2\}, \quad G_3^{2+} = \{B_j \mid B_j \in G_3, |B_j| \geq 2\}.$$

Claim 2. The sole item in each bin of $G_2^1 \cup G_3^1$ has a size larger than 1/2 (by definition).

Claim 3. $|G_2^{2+}| \leq 1$.

Proof: Suppose for a contradiction that at least two bins, say B_1 and B_2 belong to G_2^{2+}; assume without loss of generality that $u(B_1) \geq u(B_2)$. From the definition of G_2^{2+}, we see that the item with the smallest weight or the second smallest weight has a size at most $\frac{1}{3}$. Let item k be the item. If item k moves to bin B_1, then its payoff is at least $u(B_1) + u_k \geq u(B_2) + u_k > p_k$, where p_k is the payoff of item k in bin B_2. So it is not difficult to see that the item in B_2 has an incentive to move to bin B_1. Hence the assumption is false and $|G_2^{2+}| \leq 1$. □

Claim 4. With the exception of at most one bin, in each bin of G_3^{2+}, both the item with the minimum weight and the second minimum weight have size larger than 1/4.

Proof: Consider any two bins of G_3^{2+}, say B_1 and B_2. Assume without loss of generality that $u(B_1) \geq u(B_2)$. In B_2, if any item i with the smallest or the second smallest weight has a size at most $\frac{1}{4}$, then its payoff will get improved from at most $u(B_2)+u_i$ to at least $u(B_1)+u_i$, by Properties 1 and 2. Therefore each of them has a size larger than 1/4. □

Now we can proceed further with the proof of the theorem. We have an upper bound

$$\begin{aligned} NE &= |G_1| + |G_2^1| + |G_3^1| + |G_2^{2+}| + |G_3^{2+}| + |G_4| \\ &\leq 2 + |G_2^1| + |G_3^1| + |G_3^{2+}| + |G_4|, \end{aligned}$$

and a lower bound

$$OPT \geq \frac{|G_2^1| + |G_3^1|}{2} + \frac{2|G_3^{2+}|}{3} + \frac{3|G_4|}{4}. \tag{1}$$

The size of each item in the bins of $G_2^1 \cup G_3^1$ is larger than 1/2; let us call these items big. Hence, there are $|G_2^1| + |G_3^1|$ big items. In each bin of G_3^{2+}, except at most one bin, there are at least two items with a size larger than 1/4 each; let us call these items medium-sized items. Hence, there are at least $2(|G_3^{2+}|-1)$ medium-sized items. Note that no two big items can be packed into the same bin, therefore

$$OPT \geq |G_2^1| + |G_3^1|. \tag{2}$$

Case 1. $|G_2^1| + |G_3^1| \geq 2(|G_3^{2+}|-1)$. Then we also have

$$OPT \geq 2|G_3^{2+}| - 2. \tag{3}$$

We multiply the inequalities (1), (2), (3) by $\frac{24}{18}$, $\frac{6}{18}$, and $\frac{1}{18}$, respectively. Adding them we obtain

$$\frac{31}{18} OPT \geq |G_2^1| + |G_3^1| + |G_3^{2+}| + |G_4| - 1/9,$$

thus

$$NE \leq |G_2^1| + |G_3^1| + |G_3^{2+}| + |G_4| + 2 \leq \frac{31}{18} OPT + \frac{19}{9} < 1.723 \cdot OPT + 2.12.$$

Case 2. $|G_2^1| + |G_3^1| < 2(|G_3^{2+}|-1)$. Now, in any feasible packing, at most one medium-sized item can be packed with a big item into the same bin, and the remaining medium-sized items need at least $\frac{2(|G_3^{2+}|-1)-|G_2^1|-|G_3^1|}{3}$ bins. Therefore,

$$\begin{aligned} OPT &\geq |G_2^1| + |G_3^1| + \frac{2(|G_3^{2+}|-1) - |G_2^1| - |G_3^1|}{3} \\ &= \frac{2(|G_2^1| + |G_3^1|)}{3}| + \frac{2|G_3^{2+}| - 2}{3}. \end{aligned} \tag{4}$$

Now we multiply the inequalities (1), (2), and (4) by $\frac{24}{18}$, $\frac{4}{18}$, and $\frac{3}{18}$, respectively. Adding them we obtain

$$\frac{31}{18}OPT \geq |G_2^1| + |G_3^1| + |G_3^{2+}| + |G_4| - 1/9,$$

therefore we also have $NE \leq \frac{31}{18}OPT + \frac{19}{9} < 1.723OPT + 2.12$. □

Proposition 4. *Assume that $a_{ij} = u_i + u_j$, where $u_i > 0$ is the weight of item i. Then PoA is at most 2.*

Proof. It suffices to show that the average level of bins in any NE is larger than 1/2. In fact, the average is this large already for any two bins. Indeed, consider an NE, and assume for a contradiction that there are two bins, say B_1 and B_2, such that their total level is at most 1. Let $u(B_1)$ and $u(B_2)$ be defined as $u(B_1) = \sum_{j \in B_1} u_j$ and $u(B_2) = \sum_{j \in B_2} u_j$, respectively; we assume without loss of generality that $u(B_1) \geq u(B_2)$. Let l and k be the numbers of items in B_1 and B_2, respectively. Suppose first that $l \geq k$, and let i be an arbitrary item in B_2. We claim that i would like to move to B_1. The actual payoff for item i is

$$p_i = \sum_{j \in B_2} a_{ij} = \sum_{j \in B_2} (u_i + u_j) = k \cdot u_i + u(B_2).$$

If i moves to bin B_1, its payoff will be there

$$p_i' = \sum_{j \in B_1} a_{ij} + a_{ii} = \sum_{j \in B_1} (u_i + u_j) + 2u_i = (l+2) \cdot u_i + u(B_1),$$

which is bigger than p_i, thus the claim is verified. This contradicts the assumption that the packing is a NE, therefore we must have $l < k$. Let i be the item in B_1 for which u_i is the biggest, and j be the item in B_2, for which u_j is the smallest. Then

$$u_i \geq u(B_1)/l \geq u(B_2)/l > u(B_2)/k \geq u_j. \tag{5}$$

If we move i from B_1 to B_2, its payoff changes by

$$\begin{aligned} c_1 = {p_i}' - p_i &= (k \cdot u_i + u(B_2) + 2u_i) - (l \cdot u_i + u(B_1)) \\ &= (k + 2 - l)u_i + u(B_2) - u(B_1); \end{aligned}$$

and if we move j from B_2 to B_1, its payoff changes by

$$\begin{aligned} c_2 = {p_j}' - p_j &= (l \cdot u_j + u(B_1) + 2u_j) - (k \cdot u_j + u(B_2)) \\ &= (l + 2 - k)u_j + u(B_1) - u(B_2). \end{aligned}$$

Moreover, from (5) we have

$$c_1 + c_2 = 2(u_i + u_j) + (k - l)(u_i - u_j) > 0.$$

This means that at least one of items i and j will improve its payoff by moving to the other bin, a contradiction. □

Remark 3. In the two latter models, if we set $u_i = 1$ for all i, then $PoA \geq 1.6966$ can be proved by the approach given in [4].

4 Asymmetric Case

In this section we deal with the case where the matrix A is not symmetric. First we observe by giving an example that NE may not always exist; more precisely, from a suitably chosen initial packing, NE is not reached after any infinite sequence of selfish steps. Then we describe a sufficient condition for problem inputs, which ensures that there exists an initial packing and an infinite sequence of feasible steps which never lead to an NE. However we find by another example that the condition is not necessary. Finally we also give a sufficient condition, where NE always exist.

Example 1. The following instance admits an initial packing which never terminates with an NE, independently of the value of the parameter $p < 1$. Take three items 1, 2, and 3, each with size 0.5. Let $a_{i,i} = 1$ for $i = 1, 2, 3$, $a_{12} = a_{23} = a_{31} = 1$, and $a_{21} = a_{32} = a_{13} = p$.

Proof. Assume that the three items are packed in three distinct bins. Then first item 1 moves to share a bin with item 2. Then item 2 leaves item 1 alone and moves to share a bin with item 3. Then item 3 leaves item 2 alone and moves to share a bin with item 1. Then item 1 moves and again shares a bin with item 2, and the movement can be continued to infinity. Then there is no NE for the above instance. □

In the following we give a generalization of the instance in Example 1, where NE does not exits after any sequence of selfish steps. This part is related to line graphs.

Line Graph. According to standard terminology, the vertices in the line graph $L(G)$ of G represent the edges of G, and two vertices of $L(G)$ are adjacent if and only if the corresponding edges of G share a vertex. So, each edge in $L(G)$ identifies three vertices, say v_i, v_j, v_k in G, and two edges v_iv_j, v_iv_k on them, sharing one vertex v_i. Originally in G the edges are undirected; but now their common vertex v_i specifies the *ordered* pairs $(i, j), (i, k)$. In case if $a_{ij} = a_{ik}$, we remove the corresponding edge from $L(G)$; and if equality does not hold, then we orient the edge of $L(G)$ from the smaller to the larger a-value; see Fig. 1 for an illustration. We denote by $H = (X, F)$ the oriented graph obtained in this way. (This H, obviously, does not contain cycles of length 2.)

Compatibility Graph. Let $\mathcal{I} = (A, S)$ be an instance of GBPG. We define the *compatibility graph* to represent the pairs of items which can occur together in a bin. This undirected graph, which we denote by $G = (V, E)$, is described by the following rules:

– the vertices are $v_1, v_2, ..., v_n$, indexed according to the items;
– an unordered vertex pair v_iv_j is an edge if and only if $s_i + s_j \leq 1$.

$$A = \left| \begin{array}{cccc} 1 & 1 & 1 & \epsilon \\ 1{-}\epsilon & 1 & p & \epsilon \\ p & 1 & 1 & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon \end{array} \right|$$

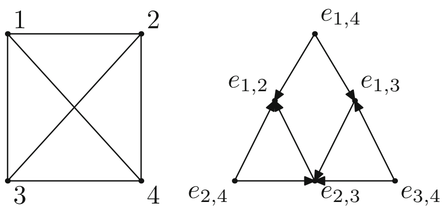


Fig. 1. Matrix, Compatibility Graph and Line Graph

Theorem 3. *Let $\mathcal{I} = (A, S)$ be an instance of GBPG, and let $H = (X, F)$ be the oriented partial line graph as described above. If H contains a directed (cyclically oriented) cycle, then there exists an initial packing of the items and an infinite sequence of feasible steps along which NE is never reached.*

Proof. Let $C = x_1 x_2 \dots x_\ell$ be a directed cycle in H. We assume, without loss of generality, that C is a *shortest* cycle in H. Each x_k $(1 \le k \le \ell)$ corresponds to some edge $e_k := v_{i_k} v_{j_k}$ of G, and we have $e_k \cap e_{k+1} \ne \emptyset$ for all $1 \le k \le \ell$ (subscript addition is taken modulo ℓ throughout the proof). We further observe:

Claim. For all k we have $e_k \cap e_{k+1} \ne e_{k+1} \cap e_{k+2}$.

Proof. Suppose for a contradiction that $e_k \cap e_{k+1} = e_{k+1} \cap e_{k+2} = v_k$, and assume that $e_i = v_k v_{j_i}$ for $i = k, k+1, k+2$. Then, by the construction of H, we have

$$a_{k,j_k} < a_{k,j_{k+1}} < a_{k,j_{k+2}}.$$

As a consequence, also $x_k x_{k+2}$ is an arc in H, because e_k and e_{k+2} share v_k and the corresponding s-values satisfy the required inequality. This contradicts the assumption that C is a shortest cycle in H, and hence the claim follows. □

To prove the theorem we start with the initial packing where the two items corresponding to the vertices of e_1 are in the same bin, and all the other items are in mutually distinct bins. The claim implies that moving the item of $e_1 \cap e_2$ from its bin to the bin of $e_2 \setminus e_1$ is feasible. More generally, from a bin whose contents are the two items belonging to e_k, it is feasible to move $e_k \cap e_{k+1}$ to the bin of $e_{k+1} \setminus e_k$, for any $1 \le k \le \ell$. Consequently, in the first $\ell - 1$ bins the first ℓ items can circulate forever, without reaching NE at any time. □

Remark 4. The line graph of the input can be constructed in linear time in terms of the input size, and it can also be tested in polynomial time whether the line graph contains a directed cycle.

Lemma 2. *Even in a line graph of an input instance there is no directed cycle, NE may not occurs after finite steps of selfish improvement.*

The proof is left in the appendix.

Next we give a sufficient condition, which ensures that the game in an asymmetric case can be converted to a symmetric game. The proof is left in the Appendix.

Proposition 5. *Assume that $a_{ij} > 0$ for any pair (i, j). If there is a univariate function $g : N \to R^+$ such that $\frac{a_{ij}}{a_{ji}} = \frac{g(i)}{g(j)}$, then the asymmetric case can be transformed to the symmetric case, and hence NE exists.*

5 Conclusions and Further Research

In this paper we introduced a new type of bin packing game, which is a common generalization of all the previously considered bin packing games. Then we studied several special types of it. There are several open questions left, let us mention two of them.

1. What are the tight bounds on PoA for the models studied?
2. Determine the algorithmic complexity of the following decision problems. *Given $\mathcal{I} = (A, S)$, an instance of GBPG, together with an initial packing,*
   (a) is it true that NE is reached after a suitable sequence of steps?
   (b) is it true that NE is reached after a finite number of steps, no matter which feasible step is chosen at any time?

   Is any of these problems polynomial-time solvable?

References

1. Bilò, V.: On the packing of selfish items. In: Proc. of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), 9 p. IEEE (2006)
2. Coffman Jr., E.G., Csirik, J., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: survey and classification. In: Pardalos, P.M., Du, D.-Z., Graham, R.L. (eds.) Handbook of Combinatorial Optimization, pp. 455–531. Springer, New York (2013)
3. Coffman, E.G., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: combinatorial analysis. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, pp. 151–208. Kluwer, Dordrecht (1999)
4. Dosa, G., Epstein, L.: Generalized selfish bin packing, arXiv:1202.4080, 1–43 (2012)
5. Dósa, G., Epstein, L.: The convergence time for selfish bin packing. In: Lavi, R. (ed.) SAGT 2014. LNCS, vol. 8768, pp. 37–48. Springer, Heidelberg (2014)
6. Dósa, G., Sgall, J.: First fit bin packing: a tight analysis. In: Portier, N., Wilke, T. (eds) Proceedings of the 30th Symposium on the Theoretical Aspects of Computer Science (STACS 2013), pp. 538–549, Kiel, Germany (2013)
7. Dósa, G., Sgall, J.: Optimal analysis of best fit bin packing. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 429–441. Springer, Heidelberg (2014)
8. Epstein, L., Kleiman, E.: Selfish Bin Packing. Algorithmica **60**(2), 368–394 (2011)
9. Garey, M.R., Johnson, D.S.: Computer and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

10. Ma, R., Dósa, G., Han, X., Ting, H.-F., Ye, D., Zhang, Y.: A note on a selfish bin packing problem. Journal of Global Optimization **56**(4), 1457–1462 (2013)
11. Nash, J.: Non-cooperative games. Annals of Mathematics **54**(2), 286–295 (1951)
12. Ullman, J.D.: The performance of a memory allocation algorithm. Technical Report 100, Princeton Univ., Princeton, NJ (1971)
13. Yu, G., Zhang, G.: Bin packing of selfish items. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 446–453. Springer, Heidelberg (2008)

The Least-Core and Nucleolus of Path Cooperative Games

Qizhi Fang[1], Bo Li[1](✉), Xiaohan Shan[2], and Xiaoming Sun[2]

[1] School of Mathematical Sciences, Ocean University of China, Qingdao, China
qfang@ouc.edu.cn, boli198907@gmail.com

[2] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{shanxiaohan,sunxiaoming}@ict.ac.cn

Abstract. Cooperative games provide an appropriate framework for fair and stable profit distribution in multiagent systems. In this paper, we study the algorithmic issues on path cooperative games that arise from the situations where some commodity flows through a network. In these games, a coalition of edges or vertices is successful if they establish a path from the source to the sink in the network, and lose otherwise. Based on dual theory of linear programming and the relationship with flow games, we provide the characterizations on the CS-core, least-core and nucleolus of path cooperative games. Furthermore, we show that the least-core and nucleolus are polynomially solvable for path cooperative games defined on both directed and undirected network.

1 Introduction

A central question in cooperative game theory is how to distribute a certain amount of profit generated by a group of agents N, denoted by a function $\gamma(N)$, to each individual. It is often assumed that the grand coalition N is formed, since in many games the total profit or costs are optimized if all agents work together. To achieve this goal, the collective profit should be distributed properly so as to minimize the incentive of subgroups of agents to deviate and form coalitions of their own. A number of solution concepts have been proposed to capture this intuition, such as the core, the least-core, and the nucleolus, which will be the focus of this paper.

In this paper, we consider a kind of cooperative game models, *path cooperative games* (PC-games), arising from the situations where some commodity (traffic, liquid or information) flows through a network. In these games, each player

Q. Fang—The first author is supported by the National Natural Science Foundation of China (NSFC) (NO. 11271341). The fourth author is supported in part by the National Natural Science Foundation of China Grant 61170062, 61222202, 61433014 and the China National Program for support of Top-notch Young Professionals. This work is also partially supported by the National Natural Science Foundation of China (NSFC) (NO. 61173009.) Finally, we would like to acknowledge our editors and a superb set of anonymous referees for their excellent suggestions.

controls an edge or a vertex of the network (called edge path cooperative games or vertex edge path cooperative games, respectively), a coalition of players wins if they establish a path from the source to the sink, and lose otherwise. We will focus on the algorithmic problems on game solutions of path cooperative games, especially core related solutions.

Path cooperative games have a natural correspondence with flow games. Flow games were first introduced by Kalai and Zemel [13] and studied extensively by many researchers. When there are public arcs in the network, the core of the flow game is nonempty if and only if there is a minimum (s, t)-cut containing no public arcs. And in this case, the core can be characterized by the minimum (s, t)-cuts[13,18], and the nucleolus can also be computed efficiently[5,17]. Recently, Aziz *et al.* [1] introduced the threshold versions of monotone games, including PC-games as a special case. Yoram [3] showed that computing ε-core for threshold network flow games is polynomial time solvable for unit capacity networks, and NP-hard for networks with general capacities. For PC-games defined on series-parallel graphs, Aziz *et al.*[1] showed that the nucleolus can be computed in polynomial time. However, the complexity of computing the nucleolus for general PC-games remains open, from the algorithmic point of view, the solution concepts of general PC-games have not been systematically discussed.

The algorithmic problems in cooperative games are especially interesting, since except for the fairness and rationality requirements in the solution definitions, computational complexity is suggested be taken into consideration as another measure of rationality for evaluating and comparing different solution concepts (Deng and Papadimitriou [6]). The computational complexity of classical solution concepts has therefore been studied with growing interest during the last decades. On the positive side, efficient algorithms have been proposed for computing the core, the least-core and the nucleolus for, such as, assignment games [20], cardinality matching games [14], unit flow games [5] and weighted voting games [8]. On the negative side, the problems of computing the nucleolus and testing whether a given distribution belongs to the core or the nucleolus are proved to be NP-hard for minimum spanning tree games [9,10], flow games and linear production games [5,11].

The main contribution of this work is the efficient characterizations of the CS-core, least-core and the nucleolus of PC-games, based on linear programming technique and the relationship with flow games. These characterizations yield directly to efficient algorithms for the related solutions. The organization of the paper is as follows.

In Section 2, the relevant definitions in cooperative game are introduced. In Section 3, we first define PC-games (edge path cooperative game and vertex path cooperative game), and then give the the characterizations of the core and CS-core. Section 4 is dedicated to the efficient description of the least-core for PC-games. In Section 5, we prove that the nucleolus is polynomially solvable for both edge and vertex path cooperative games.

2 Preliminaries

A cooperative game $\Gamma = (N, \gamma)$ consists of a player set $N = \{1, 2, \cdots, n\}$ and a characteristic function $\gamma : 2^N \to R$ with $\gamma(\emptyset) = 0$. For each coalition $S \subseteq N$, $\gamma(S)$ represents the profit obtained by S without help of other players. The set N is called the grand coalition. In what follows, we assume that $\gamma(S) \geq 0$ for all $S \subseteq N$, and $\gamma(\emptyset) = 0$.

An imputation of Γ is a payoff vector $x = (x_1, ...x_n)$ such that $\sum_{i \in N} x_i = \gamma(N)$ and $x_i \geq \gamma(\{i\})$, $\forall i \in N$. The set of imputations is denoted by $\mathcal{I}(\Gamma)$. Throughout this paper, we use the shorthand notation $x(S) = \sum_{i \in S} x_i$. Given a payoff vector $x \in \mathcal{I}(\Gamma)$, the excess of coalition $S \subseteq N$ with respect to x is defined as: $e(x, S) = x(S) - \gamma(S)$. This value measures the degree of S's satisfaction with the payoff x.

Core. The *core* of a game Γ, denoted by $\mathcal{C}(\Gamma)$, is the set of payoff vectors satisfying that, $x \in \mathcal{C}(\Gamma)$ if and only if $e(x, S) \geq 0$ for all $S \subseteq N$. These constraints, called group rationality, ensure that no coalition would have an incentive to split from the grand coalition N, and do better on its own.

Least-core. When $\mathcal{C}(\Gamma)$ is empty, it is meaningful to relax the group rationality constraints by $e(x, S) \geq \varepsilon$ for all $S \subseteq N$ ($\varepsilon < 0$). We shall find the maximum value ε^* such that the set $\{x \in \mathcal{I}(\Gamma) : e(x, S) \geq \varepsilon^*, \forall S \subseteq N\}$ is nonempty. This set of imputations is called the *least-core*, denoted by $\mathcal{LC}(\Gamma)$, and ε^* is called the value of $\mathcal{LC}(\Gamma)$ or $\mathcal{LC}$-value.

Nucleolus. Now we turn to the concept of the *nucleolus*. A payoff vector x generates a 2^n-dimensional excess vector $\theta(x) = (e(x, S_1), \cdots, e(x, S_{2^n}))$, whose components are arranged in a non-decreasing order. That is, $e(x, S_i) \leq e(x, S_j)$ for $1 \leq i < j \leq 2^n$. The nucleolus, denoted by $\eta(\Gamma)$, is defined to be a payoff vector that lexicographically maximizes the excess vector $\theta(x)$ over the set of imputations $\mathcal{I}(\Gamma)$. It was proved by Schmeidler [19] that the nucleolus of a game with the nonempty imputation set contains exactly one element.

Monotone Games and Simple Games. A game $\Gamma = (N, \gamma)$ is *monotone* if $\gamma(S') \leq \gamma(S)$ whenever $S' \subseteq S$. A game is called a simple game if it is a monotonic game with $\gamma : 2^N \to \{0, 1\}$ such that $\gamma(\emptyset) = 0$ and $\gamma(N) = 1$. Simple games can be usually used to model situations where there is a task to be completed, a coalition is labeled as winning if and only if it can complete the task. Formally, coalition $S \subseteq N$ is *winning* if $\gamma(S) = 1$, and *losing* if $\gamma(S) = 0$. A player i is called a *veto* player if he or she belongs to all winning coalitions. It is easy to see that, in a simple game, i is a veto player if and only if $\gamma(N) = 1$ but $\gamma(N \setminus \{i\}) = 0$.

For simple games, Osborne[16] and Elkind *et al.* [7] gave the following result on the core and the nucleolus.

Lemma 1. *A simple game $\Gamma = (N, \gamma)$ has a nonempty core if and only if there exists a veto player. Moreover,*

1. *$x \in \mathcal{C}(\Gamma)$ if and only if $x_i = 0$ for each $i \in N$ who is not a veto player;*
2. *when $\mathcal{C}(\Gamma) \neq \emptyset$, the nucleolus of Γ is given by $x_i = \frac{1}{k}$ if i is a veto player and $x_i = 0$ otherwise, where k is the number of veto players.*

CS-core. Taking coalition structure into consideration, we can arrive at another solution concept, *CS-core.* Given a cooperative game $\Gamma = (N, \gamma)$, a coalition structure over N is a partition of N, *i.e.*, a collection of subsets $CS = \{C^1, \cdots, C^k\}$ with $\cup_{j=1}^{k} C^j = N$ and $C^i \cap C^j = \emptyset$ for $i \neq j$ and $i, j \in \{1, \cdots, k\}$. A vector $x = (x_1, \cdots, x_n)$ is a payoff vector for a coalition structure $CS = \{C^1, \cdots, C^k\}$ if $x_i \geq 0$ for all $i \in N$, and $x(C^j) = \gamma(C^j)$ for each $j \in \{1, \cdots, k\}$.

In general, an outcome of the game Γ is a pair (CS, x), where CS is a coalition structure and x is a corresponding payoff vector. The CS-core of the game $\Gamma = (N, \gamma)$, denoted by $\mathcal{C}_{cs}(\Gamma)$, is the set of outcomes (CS, x) satisfying the constraints of "group rationality". That is,

$$\mathcal{C}_{cs}(\Gamma) = \{(CS, x) : \forall C \in CS, x(C) = \gamma(C) \text{ and } \forall S \subseteq N, x(S) \geq \gamma(S)\}.$$

A stronger property that is also enjoyed by many practically useful games is superadditivity. The game $\Gamma = (N, \gamma)$ is *superadditive* if it satisfies $\gamma(S_1 \cup S_2) \geq \gamma(S_1) + \gamma(S_2)$ for every pair of disjoint coalitions $S_1, S_2 \subseteq N$. This implies that the agents can earn at least as much profit by working together within the grand coalition. Therefore, for superadditive games, it is always assumed that the agents form the grand coalition. For a (non-superadditive) game $\Gamma = (N, \gamma)$, we can define a new game $\Gamma^* = (N, \gamma^*)$ by setting

$$\gamma^*(S) = \max_{CS \in \mathcal{CS}_S} \gamma(CS), \ \forall S \subseteq N$$

where $\mathcal{CS}_S$ denotes the space of all coalition structures over S and $\gamma(CS) = \sum_{C \in CS} \gamma(C)$. It is easy to verify that the game Γ^* is superadditive, and it is called the *superadditive cover* of Γ. The relationship between the CS-core of Γ and the core of its superadditive cover Γ^* is presented in the following lemma [4,12].

Lemma 2. *A cooperative game $\Gamma = (N, \gamma)$ has nonempty CS-core if and only if its superadditive cover $\Gamma^* = (N, \gamma^*)$ has a non-empty core. Moreover, if $\mathcal{C}(\Gamma^*) \neq \emptyset$, then $\mathcal{C}_{cs}(\Gamma) = \mathcal{C}(\Gamma^*)$.*

3 Path Cooperative Game and Its Core

Let $D = (V, E; s, t)$ be a connected flow network with unit arc capacity (called unit flow network), where V is the vertex set, E is the arc set, $s, t \in V$ are the source and the sink of the network respectively. In this paper, an (s, t)-*path* is referred to as a *directed* path from s to t that visits each vertex in V at most once.

Let $U, W \subseteq V$ be a partition of the vertex set V such that $s \in U$ and $t \in W$, then the set of arcs with tails in U and heads in W is called an (s,t)-*edge-cut*, denoted by $\bar{E} \subseteq E$. An (s,t)-*vertex-cut* is a vertex subset $\bar{V} \subseteq V \setminus \{s,t\}$ such that $D \setminus \bar{V}$ is disconnected. An (s,t)-edge(vertex)-cut is minimum if its cardinality is minimum. In the remainder of the paper, (s,t)-edge(vertex)-cuts will be abbreviated as edge(vertex)-cut S for short. Given an edge-cut $\bar{E}$, we denote its indicator vector by $\mathcal{H}_{\bar{E}} \in \{0,1\}^{|E|}$, where $\mathcal{H}_{\bar{E}}(e) = 1$ if $e \in \bar{E}$, and 0 otherwise. The indicator vector of a vertex-cut is defined analogously.

Now we introduce two kinds of path cooperative games (PC-games), *edge path cooperative games* and *vertex path cooperative games.*

Definition 1 (Path cooperative game, PC-game). *Let $D = (V, E; s, t)$ be a unit flow network.*

1. *The associated edge path cooperative game (EPC-game) $\Gamma_E = (E, \gamma_E)$ is:*
   – *The player set is E;*
   – $\forall S \subseteq E$, $\begin{cases} \gamma_E(S) = 1 & \text{if } D[S] \text{ admits an } (s,t)\text{-path;} \\ \gamma_E(S) = 0 & \text{otherwise.} \end{cases}$
   Here, $D[S]$ denotes the induced subgraph with vertex set V and edge set S.
2. *The associated vertex path cooperative game(VPC-game) $\Gamma_V = (V, \gamma_V)$ is:*
   – *The player set is $V \setminus \{s,t\}$;*
   – $\forall T \subseteq V$, $\begin{cases} \gamma_V(T) = 1 & \text{if induced subgraph } D[T] \text{ admits an } (s,t)\text{-path;} \\ \gamma_V(T) = 0 & \text{otherwise.} \end{cases}$

Clearly, PC-games fall into the class of simple games. Therefore, we can get the necessary and sufficient condition of the non-emptiness of the core directly from Lemma 1.

Proposition 1. *Given an EPC-game Γ_E and a VPC-game Γ_V associated with network $D = (V, E; s, t)$, then*

1. *$\mathcal{C}(\Gamma_E) \neq \emptyset$ if and only if the size of the minimum edge-cut of D is 1;*
2. *$\mathcal{C}(\Gamma_V) \neq \emptyset$ if and only if the size of the minimum vertex-cut of D is 1.*

Moreover, when the core of a PC-game is nonempty, the only edge (vertex) in the edge(vertex)-cut is a veto player, both the core and the nucleolus can be given directly. In the following two sections, we only consider PC-games with empty core.

We note that PC-games also have a natural correspondence with *flow games* and in what follows, we will reveal the close relationship between flow games and PC-games. Let $D = (V, E; s, t)$ be a unit flow network. Given $N \subseteq E$, each edge $e \in N$ is controlled by one player, *i.e.*, we can identify the set of edges N with the set of players. Edges not under control of any players, in $E \setminus N$, are called public arcs; they can be used freely by any coalition. Thus, a unit flow network with player set N is denoted as $D\langle N \rangle = (V, E; s, t)$

Definition 2 (Simple flow game). *The simple flow game $\Gamma_f\langle N \rangle = (N, \gamma)$ associated with the unit network $D\langle N \rangle$ is defined as:*

1. The player set is N;
2. $\forall S \subseteq N$, $\gamma(S)$ is the value of the max-flow from s to t in $D[S \cup (E \setminus N)]$ (using only the edges in S and public edges).

Flow game is a classical combinatorial optimization game, which has been extensively studied. The core of the flow game $\Gamma_f\langle N\rangle$ is nonempty if and only if there is a minimum edge-cut without public edges [18]. In this case, the core is exactly the convex hull of the indicator vectors of minimum edge-cuts without public edges in D [13,18], and the nucleolus can also be computed in polynomial time [5,17].

Now we turn to discuss the *CS*-core of PC-games. It is easy to see that for the network D without public edges, the associated flow game is the superadditive cover of the corresponding EPC-game. Thus, the nonemptiness of *CS*-core of EPC-game follows directly from Lemma 2.

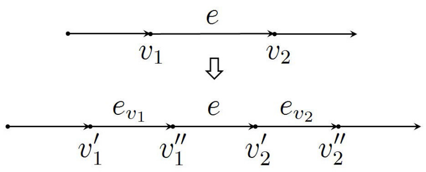
Proposition 2. *Given an EPC-game Γ_E associated with network $D = (V, E; s, t)$, then the CS-core of Γ_E is nonempty and it is exactly the convex hull of the indicator vectors of minimum edge-cuts of D.*

For a VPC-game, we can also establish some relationship with a flow game. Given a network $D = (V, E; s, t)$, we transform it into a new network D_V in the following way.

(1) For each $v \in V \setminus \{s, t\}$, split it into two distinct vertices v' and v'';

(2) Connect v' and v'' by a new directed edge $e_v = (v', v'')$. The set of all such edges is denoted by E_V;

(3) For original edge $e = (u, v) \in E$, transform it into a new edge $e = (u'', v')$ in D_V $(s = s' = s''$ and $t = t' = t'')$.



In the new constructed network D_V, the player set is just the set E_V and all the other edges are viewed as public edges. It is easy to show that in the new network D_V, there must be a minimum edge-cut containing only edges in E_V. Hence, we can verify that the flow game associated with the network $D_V\langle E_V\rangle$ is the superadditive cover of the corresponding VPC-game defined on D. Similarly, the nonemptiness of *CS*-core of VPC-game follows from Lemma 2 and the results of core nonemptiness of flow games.

Proposition 3. *Given an VPC-game Γ_V associated with network $D = (V, E; s, t)$, then the CS-core of Γ_V is nonempty and it is exactly the convex hull of the indicator vectors of minimum vertex-cuts of D.*

4 Least-Core of PC-Games

In this section, we first discuss the least-core of EPC-games. Throughout this section, $\mathit{\Gamma}_E$ is an EPC-game associated with the network $D = (V, E; s, t)$ with $|E| = n$. Denote by $\mathcal{P}$ the set of all (s,t)-path in D, and $|\mathcal{P}| = m$. According to the definitions of EPC-game and the least-core, it is shown that $\mathcal{LC}(\mathit{\Gamma}_E)$ can be formulated as the following linear program:

$$\begin{array}{ll} \max & \varepsilon \\ \text{s.t.} & \begin{cases} x(E) = 1 & \\ x(P) \geq 1 + \varepsilon & \forall P \in \mathcal{P} \\ x_i \geq 0 & \forall\ i \in E \end{cases} \end{array} \tag{1}$$

In spite that the number of the constraints in (1) may be exponential in $|E|$, the $\mathcal{LC}$-value and a least-core imputation can be found efficiently by ellipsoid algorithm with a polynomial-time separation oracle: Let (x, ε) be a candidate solution for LP($\mathcal{LC}_E$). We first check whether constraints $x(E) = 1$ and $x(e) \geq 0$ ($\forall e \in E$) are satisfied. Then, checking whether $x(P) \geq 1 + \varepsilon$ ($\forall P \in \mathcal{P}$) are satisfied is transformed to solving the shortest (s, t)-path in D with respect to the edge length $x(e)$ ($\forall e \in E$), and this can aslo be done in polynomial time.

In what follows, we aim at giving a succinct characterization of the least-core for EPC-games. We first give the linear program model of the max-flow problem on D and its dual:

$$\text{LP(flow):} \quad \begin{array}{ll} \max & \sum_{j=1}^{m} y_j \\ \text{s.t.} & \begin{cases} \sum\limits_{P_j : e_i \in P_j} y_j \leq 1 & i = 1, 2, ..., n \\ y_j \geq 0 & j = 1, 2, ..., m \end{cases} \end{array} \tag{2}$$

$$\text{DLP(flow) :} \quad \begin{array}{ll} \min & \sum_{i=1}^{n} x_i \\ \text{s.t.} & \begin{cases} \sum_{e_i : e_i \in P_j} x_i \geq 1 & j = 1, 2, ..., m \\ x_i \geq 0 & i = 1, ..., n \end{cases} \end{array} \tag{3}$$

Due to *max-flow and min-cut* theorem, the optimum value of (2) and (3) are equal, and the set of optimal solutions of (3) is exactly the convex hull of the indicator vectors of the minimum edge-cut of D, which is denoted by $\mathbb{C}_E$. On the other hand, it is known that the core of the flow game $\mathit{\Gamma}_f$ defined on $D\langle E \rangle$ is also the convex hull of the indicator vectors of the minimum edge-cut of D. Hence, we have

Theorem 1. *Let $\mathit{\Gamma}_E$ and $\mathit{\Gamma}_f$ be an EPC-game and a flow game defined on $D = (V, E; s, t)$, respectively, f^* be the value of the max-flow of D. Then,*

$$x \in \mathcal{LC}(\mathit{\Gamma}_E) \textit{ if and only if } x = z/f^* \textit{ for some } z \in \mathbb{C}_E.$$

Proof. Let $x = (1 + \varepsilon)z$ be a transformation, then (1) can be rewritten as

$$\begin{array}{ll} \max & \varepsilon \\ \text{s.t.} & \begin{cases} z(E) = 1/(1 + \varepsilon) & \\ z(P) \geq 1 & \forall P \in \mathcal{P} \\ z_i \geq 0 & \forall e_i \in E \end{cases} \end{array} \tag{4}$$

Combining the first constraint $z(E) = 1/(1+\varepsilon)$ and the objective function, it is easy to see that linear program (4) is the same as DLP(flow) (3). Since the optimal value of (3) is also f^*, Theorem 1 thus follows. □

Based on the relationship between a VPC-game and the corresponding flow game discussed in Section 3, we can obtain a similar result on the least-core for VPC-games (The proof is omitted).

Theorem 2. *Let $\Gamma_V = (E, \gamma_V)$ be a VPC-game defined on $D = (V, E; s, t)$, f^* be the value of the max-flow of D, then*

$$x \in \mathcal{LC}(\Gamma_V) \text{ if and only if } x = z/f^* \text{ for some } z \in \mathbb{C}_V.$$

Here $\mathbb{C}_V$ is the convex hull of the indicator vectors of minimum vertex-cuts in D.

Theorem 1 and 2 show that for the unit flow network, the least-core of the PC-game is equivalent to the core of the corresponding flow game in the sense of scaling down by $1/f^*$. Hence, all the following problems for PC-games can be solved efficiently:

– Computing the $\mathcal{LC}$-value;
– Finding an imputation in $\mathcal{LC}(\Gamma_E)$ and $\mathcal{LC}(\Gamma_V)$;
– Checking whether a given imputation is in $\mathcal{LC}(\Gamma_E)$ or $\mathcal{LC}(\Gamma_V)$.

Remark. Path cooperative games have close relationship with a noncooperative two-person zero-sum game, called *path intercept game* [21]. In this model, an "evader" attempts to select a path P from the source to the sink through a given network. At the same time, an "interdictor" attempts to select an edge e in this network to detect the evader. If the evader traverses through arc e, he is detected; otherwise, he goes undetected. The interdictor aims to find a probabilistic "edge-inspection" strategy to maximize the average probability of detecting the evader. While for the evader, he wants to find a "path-selection strategy" to minimize the interdiction probability. Aziz *et al.*[2] observed that the mixed Nash Equilibrium of path intercept games is the same as the least-core of EPC-games. With max-min theorem in matrix game theory, the same result can be obtained based on the similar analysis as in the proof of Theorem 1.

5 Nucleolus of PC-games

In this section, we aim at showing that the nucleolus of PC-games can be computed in polynomial time. Given a game $\Gamma = (N, \gamma)$, Kopelowitz [15] showed that the nucleolus $\eta(\Gamma)$ can be obtained by recursively solving the following standard sequence of linear programs $SLP(\eta(\Gamma))$:

$$\begin{array}{c} LP_k \\ (k=1,2,\cdots) \end{array} : \quad \begin{array}{ll} & \max \varepsilon \\ \text{s.t.} & \begin{cases} x(S) = \gamma(S) + \varepsilon_r, \;\; \forall S \in \mathcal{J}_r \quad r = 0, 1, \cdots, k-1 \\ x(S) \geq \gamma(S) + \varepsilon, \quad \forall \emptyset \neq S \subset N \setminus \cup_{r=0}^{k-1} \mathcal{J}_r \\ x \in \mathcal{I}(\Gamma). \end{cases} \end{array}$$

Initially, set $\mathcal{J}_0 = \{\emptyset, N\}$ and $\varepsilon_0 = 0$. The number ε_r is the optimal value of the r-th program LP_r, and $J_r = \{S \subseteq N : x(S) = \gamma(S) + \varepsilon_r, \forall x \in X_r\}$, where $X_r = \{x \in R^n : (x, \varepsilon_r)$ is an optimal solution of $LP_r\}$.

As in the last section, we first discuss the nucleolus of EPC-games. Let Γ_E be the EPC-game associated with network $D = (V, E; s, t)$ with $|E| = n$, $\mathcal{P}$ be the set of all (s,t)-paths and f^* be the value of the max-flow of D. Denote $\mathcal{E}_\Gamma$ be the set of coalitions consisting of one-edge coalitions and path coalitions, *i.e.*,

$$\mathcal{E}_\Gamma = \{\{e\} : e \in E\} \cup \{P \subseteq E : P \in \mathcal{P} \text{ is an } (s,t)\text{-path}\}.$$

We show that the sequential linear programs $SLP(\eta(\Gamma_E))$ of EPC-game Γ_E can be simplified as follows.

$$LP'_k : \quad \begin{array}{ll} \max & \varepsilon \\ \text{s.t.} & \begin{cases} x(e) = \varepsilon_r, & \forall e \in E_r, r = 0, 1, ..., k-1 \\ x(e) \geq \varepsilon, & \forall e \in E \setminus \bigcup_{r=0}^{k-1} E_r \\ x(P) = 1/f^* + \varepsilon_r, & \forall P \in \mathcal{P}_r, r = 0, 1, ..., k-1 \\ x(P) \geq 1/f^* + \varepsilon, & \forall P \in \mathcal{P} \setminus \bigcup_{r=0}^{k-1} \mathcal{P}_r \\ x(e) \geq 0, & \forall e \in E \\ x(E) = 1. & \end{cases} \end{array} \tag{5}$$

where ε_r is the optimum value of LP_r, $X_r = \{x \in R^n : (x, \varepsilon_r)$ is an optimal solution of $LP_r\}$, $\mathcal{P}_r = \{P \in \mathcal{P} : x(P) = 1 + \varepsilon_r, \forall x \in X_r\}$ and $E_r = \{e \in E : x(e) = \varepsilon_r, \forall x \in X_r\}$. Initially, $\varepsilon_0 = 0$, $\mathcal{P}_0 = \emptyset$ and $E_0 = \emptyset$.

Proposition 4. *The nucleolus $\eta(\Gamma_E)$ of EPC-game Γ_E defined on the network $D = (V, E; s, t)$ can be obtained by computing the linear programs LP'_k in (5).*

Proof. Firstly, we show that in sequential linear programs $SLP(\eta(\Gamma))$, only the constraints corresponding to the the coalitions in $\mathcal{E}_\Gamma$ (*i.e.*, the one-edge coalitions and path coalitions) are necessary in determining the nucleolus $\eta(\Gamma_E)$.

In fact, for any winning coalition $S \subseteq N$ (not a path), S can be decomposed into a path P and some edges $E' = S \setminus E(P)$. Then,

$$x(S) - \gamma(S) = x(P) - 1 + \sum_{e \in E'} x(e) \geq x(P) - 1.$$

Since $x(e) \geq 0$ for all $e \in E'$, S cannot be fixed before P or any $e \in E'$. After P and all $e \in E'$ are fixed, S is also fixed, *i.e.*, S is redundant. If S is a losing coalition, then S is a set of edges with $\gamma(S) = 0$ and $x(S) - \gamma(S) = \sum_{e \in S} x(e) \geq x(e), \forall e \in S$. That is to say, S cannot be fixed before any $e \in S$. When all edges in S are fixed, S is fixed accordingly, *i.e.* S is also redundant in this case. Therefore, deleting all the constraints corresponding to the coalitions not in $\mathcal{E}_\Gamma$ will not change the result of $SLP(\eta(\Gamma))$.

The key point in remainder of the proof is the correctness of the third and the forth constraints in (5), where we replace the original constraints $x(P) = 1 + \varepsilon_r$ and $x(P) \geq 1 + \varepsilon$ in $SLP(\eta(\Gamma))$ with new constraints $x(P) = 1/f^* + \varepsilon_r$ and $x(P) \geq 1/f^* + \varepsilon$, respectively.

In the process of solving the sequential linear programs, the optimal values increase with k. Since $\mathcal{C}(\Gamma_E) = \emptyset$, we know $\varepsilon_1 < 0$. Note that we can always find an optimal solution such that $\varepsilon_1 > -1$ (for example $x(e) = \frac{1}{n}, \forall e \in E$ is a feasible solution of the linear programming of $\mathcal{LC}(\Gamma_E)$).

We can divide the process into two stages. The first stage is the programs with $-1 < \varepsilon_r < 0$. In this case, the constraints $x(e) \geq \varepsilon, \forall e \in E$ cannot affect the optimal solutions of the current programs, because $x(e) \geq 0$. Ignoring the invalid constraints we can get (5) directly.

The second stage is the programs with $\varepsilon_r \geq 0$. When the programs arrive at this stage, we can claim that all paths have been fixed. Otherwise, if there is a path satisfying $x(p) = 1 + \varepsilon_r \geq 1$, then we have $x(p) = 1$ (note $x(E) = 1$), contradicting with the precondition that the value of maximum flow $f^* \geq 2$. We can omit the path constraints in this stage and then this implies (5).

This completes the proof of Proposition 4. $\square$

In the following, we shall show that the nucleolus of PC-games can be solved in polynomial time. Let $\Gamma_f = (E, \gamma)$ be the flow game defined on the unit flow network $D = (V, E; s, t)$. It is easy to show that the sequential linear programs $LP(\eta(\Gamma_f))$ can be simplified as $\widetilde{LP}_k, (k = 1, 2, ...)$:

$$\widetilde{LP}_k : \quad \begin{array}{ll} \max & \varepsilon \\ \text{s.t.} & \begin{cases} x(e) = \varepsilon_r & \forall e \in E_r, r = 0, 1, ..., k-1 \\ x(P) = 1 + \varepsilon_r & \forall P \in \mathcal{P}_r, r = 0, 1, ..., k-1 \\ x(e) \geq \varepsilon & \forall e \in E \setminus \bigcup_{r=0}^{k-1} E_r \\ x(P) \geq 1 + \varepsilon & \forall P \in \mathcal{P} \setminus \bigcup_{r=0}^{k-1} \mathcal{P}_r \\ x(E) = f^*, \end{cases} \end{array} \tag{6}$$

where ε_r is the optimum value of $\widetilde{LP}_r$, $X_r = \{x \in R^n : (x, \varepsilon_r)$ is an optimal solution of $\widetilde{LP}_r\}$, $\mathcal{P}_r = \{P \in \mathcal{P} : x(P) = 1 + \varepsilon_r, \forall x \in X_r\}$ and $E_r = \{e \in E : x(e) = \varepsilon_r, \forall x \in X_r\}$. Initially, $\varepsilon_0 = 0$, $\mathcal{P}_0 = \emptyset$ and $E_0 = \emptyset$.

Deng *et al.* [5] proved that the sequential linear programs $\widetilde{LP}_k, (k = 1, 2...)$ can be transformed to another sequential linear programs with only polynomial number of constraints, and it follows that the nucleolus of flow game $\eta(\Gamma_f)$ can be found efficiently. Futhermore, Potters *et al.* [17], show that the nucleolus of flow games with public edges can also be found in polynomial time when the core is nonempty. Based on these known results, we discuss the algorithmic problem on the nucleolus of PC-games in the following theorems.

Theorem 3. *Let Γ_E and Γ_f be the EPC-game and flow game defined on a unit flow network $D = (V, E; s, t)$, respectively. The nucleolus of Γ_E can be computed in polynomial time. Furthermore,*

$$x \in \eta(\Gamma_E) \text{ if and only if } z = x \cdot f^* \in \eta(\Gamma_f),$$

where f^* is the value of the max-flow of D.

Proof. Notice that the dimension of the feasible regions of $LP'_k(k = 1, 2...)$ decreases in each step, so we can complete the process within at most $|N|$ steps.

The key point here is to show that there is a one-to-one correspondence between the optimal solutions of $\widetilde{LP}_k$ (6) and that of LP'_k (5) ($\forall k = 1, 2, \cdots$).

We first prove that if $(z^*, \tilde{\varepsilon}^*)$ is an optimal solution of $\widetilde{LP}_k$ (6), then $(x^*, \varepsilon^*) = (z^*/f^*, \tilde{\varepsilon}^*/f^*)$ is an optimal solution of LP'_k (5).

When $k = 1$, we have $E_0 = \emptyset$, $\mathcal{P}_0 = \emptyset$ in LP'_1. And it is easy to check the feasibility and the optimality of (z^*, ε^*) in LP'_1. To continue the proof recursively, we need to explain $E_1 = \tilde{E}_1$ and $\mathcal{P}_1 = \tilde{\mathcal{P}}_1$, *i.e.*, the constraints which become tight in every iteration are exactly the same in the two linear programs. For each $e \in E$, if $z^*(e) = \tilde{\varepsilon}^*$, then $x^*(e) = z^*(e)/f^* = \tilde{\varepsilon}^*/f^* = \varepsilon^*$. And if $z^*(e) > \tilde{\varepsilon}^*$, then we have $x^*(e) > \varepsilon^*$. Thus, $E_1 = \tilde{E}_1$. $\mathcal{P}_1 = \tilde{\mathcal{P}}_1$ can be shown analogously. The other direction of the result can be shown similarly. That is, the conclusion holds for $k = 1$.

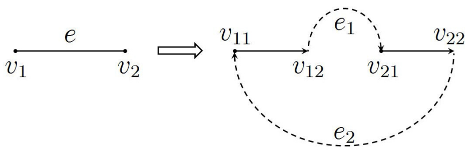
For the rest iterations $k = 2, 3, \cdots$, the proof can be carried out in a same way. Here we omit the detail of the proof. Since the nucleolus of flow game can be found in polynomial time, it follows that the nucleolus of EPC-game is also efficiently solvable. □

As for the nucleolus of VPC-games, we also show that it is polynomially solvable based on the relationship between a VPC-game and the corresponding flow game demonstrated in Section 3. Due to the space limitation, the proof of the following theroem is omitted.

Theorem 4. *The nucleolus of VPC-games can be solved in polynomial time.*

PC-games on Undirected Networks. Given an undirected network $D = (V, E; s, t)$, we construct a directed network $\overrightarrow{D} = (V, \overrightarrow{E}; s, t)$ derived from D as follows (see the following figure):

1. For edge $e \in E$ with end vertices v_1 and v_2, transform it into two directed edges $\overrightarrow{e}_{v_1} = (v_{11}, v_{12})$ and $\overrightarrow{e}_{v_2} = (v_{21}, v_{22})$;
2. Connect the two directed edges into a directed cycle via two supplemental directed edges $\overrightarrow{e}_1$ and $\overrightarrow{e}_2$.



Thus, the EPC-game defined on undirected network $D = (V, E; s, t)$ is transformed to an EPC-game defined on the constructed directed network $\overrightarrow{D} = (V, \overrightarrow{E}; s, t)$. Furthermore, it is easy to check that there exists one-to-one correspondence for the game solution (such as, the core, the least-core and the nucleolus) between the two games. As for a VPC-game defined on an undirected network, we first transform it into EPC-game on an undirected network as demonstrated in Section 3, and then transform it to EPC-game on a directed network in the same way as above. Henceforth, the algorithmic results for PC-games can be generalized from directed networks to undirected networks.

Theorem 5. *Computing the least-core and the nucleolus can be done in polynomial time for both EPC-games and VPC-games defined on undirected networks.*

References

1. Aziz, H., Brandt, F., Harrenstein, P.: Monotone cooperative games and their threshold versions. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 1107–1114 (2010)
2. Aziz, H., Sørensen, T.B.: Path coalitional games (2011). arXiv preprint arXiv:1103.3310
3. Bachrach, Y.: The least-core of threshold network flow games. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 36–47. Springer, Heidelberg (2011)
4. Chalkiadakis, G., Elkind, E., Wooldridge, M.: Computational aspects of cooperative game theory. Synthesis Lectures on Artificial Intelligence and Machine Learning **5**(6), 1–168 (2011)
5. Deng, X., Fang, Q., Sun, X.: Finding nucleolus of flow game. Journal of combinatorial optimization **18**(1), 64–86 (2009)
6. Deng, X., Papadimitriou, C.H.: On the complexity of cooperative solution concepts. Mathematics of Operations Research **19**(2), 257–266 (1994)
7. Elkind, E., Goldberg, L.A., Goldberg, P.W., Wooldridge, M.: Computational complexity of weighted threshold games. In: Proceedings of the National Conference on Artificial Intelligence, vol. 22, p. 718 (2007)
8. Elkind, E., Pasechnik, D.: Computing the nucleolus of weighted voting games. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 327–335 (2009)
9. Faigle, U., Kern, W., Fekete, S.P., Hochstättler, W.: On the complexity of testing membership in the core of min-cost spanning tree games. International Journal of Game Theory **26**(3), 361–366 (1997)
10. Faigle, U., Kern, W., Kuipers, J.: Note computing the nucleolus of min-cost spanning tree games is np-hard. International Journal of Game Theory **27**(3), 443–450 (1998)
11. Fang, Q., Zhu, S., Cai, M., Deng, X.: On computational complexity of membership test in flow games and linear production games. International Journal of Game Theory **31**(1), 39–45 (2002)
12. Greco, G., Malizia, E., Palopoli, L., Scarcello, F.: On the complexity of the core over coalition structures. In: IJCAI, vol. 11, pp. 216–221. Citeseer (2011)
13. Kalai, E., Zemel, E.: Generalized network problems yielding totally balanced games. Operations Research **30**(5), 998–1008 (1982)
14. Kern, W., Paulusma, D.: Matching games: the least-core and the nucleolus. Mathematics of Operations Research **28**(2), 294–308 (2003)
15. Kopelowitz, A.: Computation of the kernels of simple games and the nucleolus of n-person games. Technical report, DTIC Document (1967)

16. Osborne, M.J., Rubinstein, A.: A course in game theory. Cambridge, Massachusetts (1994)
17. Potters, J., Reijnierse, H., Biswas, A.: The nucleolus of balanced simple flow networks. Games and Economic Behavior **54**(1), 205–225 (2006)
18. Reijnierse, H., Maschler, M., Potters, J., Tijs, S.: Simple flow games. Games and Economic Behavior **16**(2), 238–260 (1996)
19. Schmeidler, D.: The nucleolus of a characteristic function game. SIAM Journal on applied mathematics **17**(6), 1163–1170 (1969)
20. Solymosi, T., Raghavan, T.E.S.: An algorithm for finding the nucleolus of assignment games. International Journal of Game Theory **23**(2), 119–143 (1994)
21. Washburn, A., Wood, K.: Two-person zero-sum games for network interdiction. Operations Research **43**(2), 243–251 (1995)

Reversible Pebble Game on Trees

Balagopal Komarath(✉), Jayalal Sarma, and Saurabh Sawlani

Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai, India
baluks@gmail.com

Abstract. A surprising equivalence between different forms of pebble games on graphs - Dymond-Tompa pebble game (studied in [4]), Raz-McKenzie pebble game (studied in [10]) and reversible pebbling (studied in [1]) - was established recently by Chan[2]. Motivated by this equivalence, we study the reversible pebble game and establish the following results.

– We give a polynomial time algorithm for computing reversible pebbling number of trees. As our main technical contribution, we show that the reversible pebbling number of any tree is exactly one more than the edge rank colouring of the underlying undirected tree.
– By exploiting the connection with the Dymond-Tompa pebble game, we show that complete binary trees have optimal pebblings that take at most $n^{O(\log\log(n))}$ steps. This substantially improves the previous bound of $n^{O(\log(n))}$ steps.
– Furthermore, we show that *almost optimal* (within $(1+\epsilon)$ factor for any constant $\epsilon > 0$) pebblings of complete binary trees can be done in polynomial number of steps.
– We also show a time-space tradeoff for reversible pebbling for families of bounded degree trees: for any constant $\epsilon > 0$, such families can be pebbled using $O(n^{\epsilon})$ pebbles in $O(n)$ steps. This generalizes a result of Královic[7] who showed the same for chains.

1 Introduction

Pebbling games on graphs of various forms abstracts out resources in different combinatorial models of computation (see [3]). The most obvious connection is to the space used by the computation process. A pebble placed on a vertex in a graph corresponds to storing the value at that vertex and an edge (a, b) in the graph would represent a data-dependency - namely, value at b can be computed only if the value at a is known (or stored). Devising the rules of the pebbling game to capture the moves in the computation, and establishing bounds for the total number of pebbles used at any point in time, give rise to a combinatorial approach to proving bounds on the space used by the computation. The Dymond-Tompa pebble game and the Raz-Mckenzie pebble games depict some of the combinatorial barriers in proving bounds for depth (or parallel time) of Boolean circuits (or parallel algorithms).

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 83–94, 2015.
DOI: 10.1007/978-3-319-21398-9_7

Motivated by applications in the context of reversible computation (for example, quantum computation), Bennett[1] introduced the reversible pebbling game. Given any DAG G with a unique sink vertex r, the reversible pebbling game starts with no pebbles on G and ends with a pebble (only) on r. Pebbles can be placed or removed from any vertex according to the following two rules.

1. To pebble v, all in-neighbours of v must be pebbled.
2. To unpebble v, all in-neighbours of v must be pebbled.

The goal of the game is to pebble the DAG G using the minimum number of pebbles (also using the minimum number of steps).

Recently, Chan[2] showed that for any DAG G the number of pebbles required for the reversible pebbling game is exactly the same as the number of pebbles required for the Dymond-Tompa pebble game and the Raz-Mckenzie pebble game. Chan[2] also studied the complexity of the following problem – Given a DAG $G = (V, E)$ with sink r and an integer $1 \leq k \leq |V|$, check if G can be pebbled using at most k pebbles. He showed that this problem is PSPACE-complete.

The irreversible black and black-white pebble games are known to be PSPACE-complete on DAGs (see [5], [6]). When we restrict the irreversible black pebbling game to be read-once (each vertex is pebbled only once), then the problem becomes NP-complete (see [11]). However, if we restrict the DAG to a tree, the irreversible black pebble game[9] and black-white pebble game[13] are solvable in polynomial time. The key insight is that optimal *irreversible* (black or black-white) pebbling number of trees can be achieved by read-once pebblings of trees. This fact simplifies many arguments for irreversible pebblings of trees. For example, deciding whether the pebbling number is at most k is in NP since the optimal pebbling can be used as the certificate. We cannot show that reversible pebbling is in NP using the same argument as we do not know whether the optimal value can always be achieved using pebblings taking only polynomially many steps.

Our Results: In this paper, we study reversible pebblings on trees. We show that the reversible pebbling number of trees along with strategies achieving the optimal value can be computed in polynomial time. Our main technical result is that the reversible pebbling number of any tree is exactly one more than the edge rank colouring of the underlying undirected tree. We then use the linear-time algorithm given by Lam and Yue [8] for finding an optimal edge rank coloring of the underlying undirected tree and show how to convert an optimal edge rank coloring into an optimal reversible pebbling.

Chan[2] also raised the question whether we can find connections between other parameters of different pebbling games. Although, we do not answer this question, we show that the connection with Dymond-Tompa pebble game can be exploited to show that complete binary trees have optimal pebblings that take at most $n^{O(\log\log(n))}$ steps. This is a significant improvement over the trivial $n^{O(\log(n))}$ steps.

Furthermore, we show that "almost" (within $(1+\epsilon)$ factor for any constant $\epsilon > 0$) optimal pebblings of complete binary trees can be done in polynomial number of steps. We also generalize a time-space tradeoff result given for chains by Královic [7] to families of bounded degree trees showing that for any constant $\epsilon > 0$, such families can be pebbled using $O(n^\epsilon)$ pebbles in $O(n)$ steps.

2 Preliminaries

We assume familiarity with basic definitions in graph theory, such as those found in [12]. A directed tree $T = (V, E)$ is called a *rooted directed tree* if there is an $r \in V$ such that r is reachable from every vertex in T. The vertex r is called the root of the tree.

An *edge rank coloring* of an undirected tree T with k colours $\{1, \ldots, k\}$ labels each edge of T with a colour such that if two edges have the same colour i, then the path between these two edges consists of an edge with some colour $j > i$. The minimum number of colours required for an edge rank colouring of T is denoted by $\chi'_e(T)$.

Definition 1. *(Reversible Pebbling[1]) Let G be a rooted DAG with root r. A* reversible pebbling configuration *of G is a subset of V which denotes the set of pebbled vertices). A* reversible pebbling *of G is a sequence of reversible pebbling configurations $\mathcal{P} = (P_1, \ldots, P_m)$ such that $P_1 = \phi$ and $P_m = \{r\}$ and for every $i, 2 \leq i \leq m$, we have*

1. *$P_i = P_{i-1} \cup \{v\}$ or $P_{i-1} = P_i \cup \{v\}$ and $P_i \neq P_{i-1}$ (Exactly one vertex is pebbled/unpebbled at each step).*
2. *All in-neighbours of v are in P_{i-1}.*

The number m is called the time taken by the pebbling $\mathcal{P}$. The number of pebbles or space used in a reversible pebbling of G is the maximum number of pebbles on G at any time during the pebbling. The persistent reversible pebbling number *of G, denoted by $R^\bullet(G)$, is the minimum number of pebbles required to pebble G.*

A closely related notion is that of visiting *reversible pebbling, where the pebbling $\mathcal{P}$ satisfies (1) $P_1 = P_m = \phi$ and (2) there exists a j such that $r \in P_j$. The minimum number of pebbles required for a visiting pebbling of G is denoted by $R^\phi(T)$.*

It is easy to see that $R^\phi(G) \leq R^\bullet(G) \leq R^\phi(G) + 1$ for any DAG G.

Definition 2. *(Dymond-Tompa Pebble Game [4]) Let G be a DAG with root r. A Dymond-Tompa pebble game is a two-player game on G where the two players, the pebbler and the challenger take turns. In the first round, the pebbler pebbles the root vertex and the challenger challenges the root vertex. In each subsequent round, the pebbler pebbles a (unpebbled) vertex in G and the challenger either challenges the vertex just pebbled or re-challenges the vertex challenged in the* *previous round. The pebbler wins when the challenger challenges a vertex v and all in-neighbours of v are pebbled.*

The Dymond-Tompa pebble number of G, denoted $DT(G)$, is the minimum number of pebbles required by the pebbler to win against an optimal challenger play.

The Raz-Mckenzie pebble game is also a two-player pebble game played on DAGs. The optimal value is denoted by $RM(G)$. A definition for the Raz-Mckenzie pebble game can be found in [10]. Although the Dymond-Tompa game and the reversible pebbling game look quite different. The following theorem reveals a surprising connection between them.

Theorem 1. *(Theorems 6 and 7, [2]) For any rooted DAG G, we have $DT(G) = R^{\bullet}(G) = RM(G)$.*

Definition 3. *(Effective Predecessor [2]) Given a pebbling configuration P of a DAG G with root r, a vertex v in G is called an* effective predecessor *of r if there exists a path from v to r with no pebbles on the vertices in the path (except at r).*

Lemma 1. *(Claim 3.11, [2]) Let G be any rooted DAG. There exists an optimal pebbler strategy for the Dymond-Tompa pebble game on G such that the pebbler always pebbles an effective predecessor of the currently challenged vertex.*

We call the above pebbling strategy (resp. pebbler) as an upstream pebbling strategy(resp. upstream pebbler). The height or depth of a tree is defined as the maximum number of vertices in any root to leaf path. We denote by Ch_n the rooted directed path on n vertices with a leaf as the root. We denote by Bt_h the the complete binary tree of height h. We use $root(Bt_h)$ to refer to the root of Bt_h. If v is any vertex in Bt_h, we use $left(v)$ ($right(v)$) to refer to the left (right) child of v. We use $right^i$ and $left^i$ to refer to iterated application of these functions. We use the notation $Ch_i + Bt_h$ to refer to a tree that is a chain of i vertices where the source vertex is the root of a Bt_h.

Definition 4. *We define the language* TREE-PEBBLE *(*TREE-VISITING-PEBBLE*) as the set of all tuples (T, k), where T is a rooted directed tree and k is an integer satisfying $1 \leq k \leq n$, such that $R^{\bullet}(T) \leq k$ ($R^{\phi}(T) \leq k$).*

In the rest of the paper, we use the term pebbling to refer to *persistent reversible pebbling* unless explicitly stated otherwise.

3 Main Theorem

Definition 5. *(Strategy Tree) Let T be a rooted directed tree. If T only has a single vertex v, then any strategy tree for T only has a single vertex labelled v. Otherwise, we define a strategy tree for T as any tree satisfying*

1. *The root vertex is labelled with some edge $e = (u, v)$ in T.*

2. The left subtree of root is a strategy tree for T_u and the right subtree is a strategy tree for $T \setminus T_u$.

The following properties are satisfied by any strategy tree S of $T = (V, E)$.

1. Each vertex has 0 or 2 children.
2. There are bijections from E to internal vertices of S & from V to leaves of S.
3. Let v be any vertex in S. Then the subtree S_v corresponds to the subtree of T spanned by the vertices labelling the leaves of S_v. If u and v are two vertices in S such that one is not an ancestor of the other, then the subtrees in T corresponding to u and v are vertex-disjoint.

Lemma 2. *Let T be a rooted directed tree. Then $R^\bullet(T) \leq k$ if and only if there exists a strategy tree for T of depth at most k.*

Proof. We prove both directions by induction on $|T|$. If T is a single vertex tree, then the statement is trivial.

(if) Assume that the root of a strategy tree for T of depth k is labelled by an edge (u, v) in T. The pebbler then pebbles the vertex u. If the challenger challenges u, the pebbler follows the strategy for T_u given by the left subtree of root. If the challenger rechallenges, the pebbler follows the strategy for $T \setminus T_u$ given by the right subtree of the root. The remaining game takes at most $k - 1$ pebbles by the inductive hypothesis. Therefore, the total number of pebbles used is at most k.

(only if) Consider an upstream pebbler that uses at most k pebbles. We are going to construct a strategy tree of depth at most k. Assume that the pebbler pebbles u in the first move where $e = (u, v)$ is an edge in T. Then the root vertex of S is labelled e. Now we have $R^\bullet(T_u), R^\bullet(T \setminus T_u) \leq k - 1$. Let the left (right) subtree be the strategy tree obtained inductively for T_u $(T \setminus T_u)$. Since the pebbler is upstream, the pebbler never places a pebble outside T_u $(T \setminus T_u)$ once the challenger has challenged u (the root). □

Definition 6. *(Matching Game) Let U be an undirected tree. Let $T_1 = U$. At each step of the matching game, we pick a matching M_i from T_i and contract all the edges in M_i to obtain the tree T_{i+1}. The game ends when T_i is a single vertex tree. We define the* contraction number *of U, denoted $c(U)$, as the minimum number of matchings in the matching sequence required to contract U to the single vertex tree.*

Lemma 3. *Let T be a rooted directed tree and let U be the underlying undirected tree for T. Then $R^\bullet(T) = k + 1$ if and only if $c(U) = k$.*

Proof. First, we describe how to construct a matching sequence of length k from a strategy tree S of depth $k + 1$. Let the leaves of S be the level 0 vertices. For $i \geq 1$, we define the level i vertices to be the set of all vertices v in S such that one child of v has level $i - 1$ and the other child of v has level at most $i - 1$.

Define M_i to be the set of all edges in U corresponding to level i vertices in S. We claim that $M_1, \dots, M_k$ is a matching sequence for U. Define S_i as the set of all vertices v in S such that the parent of v has level at least $i+1$ (S_k contains only the root vertex). Let $Q(i)$ be the statement "T_{i+1} is obtained from T_1 by contracting all subtrees corresponding to vertices (see Property 3) in S_i". Let $P(i)$ be the statement "M_{i+1} is a matching in T_{i+1}". We will prove $Q(0)$ and $Q(i) \implies P(i)$ and $(Q(i) \wedge P(i)) \implies Q(i+1)$. Indeed for $i = 0$, we have $Q(0)$ because $T_1 = U$ and S_0 is the set of all leaves in S or vertices in T (Property 2). To prove $Q(i) \implies P(i)$, observe that the edges of M_{i+1} correspond to vertices in S where both children are in S_i. So these edges correspond to edges in T_{i+1} (by $Q(i)$) and these edges are pairwise disjoint since no two vertices in S have a common child.

To prove that $(Q(i) \wedge P(i)) \implies Q(i+1)$, consider the tree T_{i+2} obtained by contracting M_{i+1} from T_{i+1}. Since $Q(i)$ is true, this is equivalent to contracting all subtrees corresponding to S_i and then contracting the edges in M_{i+1} from T_1. The set S_{i+1} can be obtained from S_i by adding all vertices in S corresponding to edges in M_{i+1} and then removing both children (of these newly added vertices) from S_i. This is equivalent to combining the subtrees removed from S_i using the edge joining them. This is because M_{i+1} is a matching by $P(i)$ and hence one subtree in S_i will never be combined with two other subtrees in S_i. But then contracting subtrees in S_{i+1} from T_1 is equivalent to contracting S_i followed by contracting M_{i+1}.

We now show that a matching sequence of length at most k can be converted to a strategy tree of depth at most $k+1$. We use proof by induction. If the tree T is a single vertex tree, then the statement is trivial. Otherwise, let e be the edge in the last matching M_k in the sequence and let (u, v) be the corresponding edge in T. Label the root of S by e and let the left (right) subtree of root of S be obtained from the matching sequence $M_1, \dots, M_{k-1}$ restricted to T_u ($T \setminus T_u$). By the inductive hypothesis, these subtrees have height at most $k-1$. □

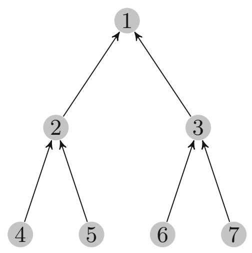
Lemma 4. *For any undirected tree U, we have $c(U) = \chi'_e(U)$.*

Proof. Consider an optimal matching sequence for U. If the edge e is contracted in M_i, then label e with the color i. This is an edge rank coloring. Suppose for contradiction that there exists two edges e_1 and e_2 with label i such that there is no edge labelled some $j \geq i$ between them. We can assume without loss of generality that there is no edge labelled i between e_1 and e_2 since if there is one such edge, we can let e_2 to be that edge. Then e_1 and e_2 are adjacent in T_i and hence cannot belong to the same matching.

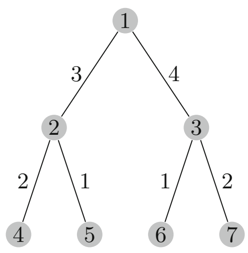
Consider an optimal edge rank coloring for U. Then in the i^{th} step all edges labelled i are contracted. This forms a matching since in between any two edges labelled i, there is an edge labelled $j > i$ and hence they are not adjacent in T_i. □

The theorems in this section are summarized in Fig. 1

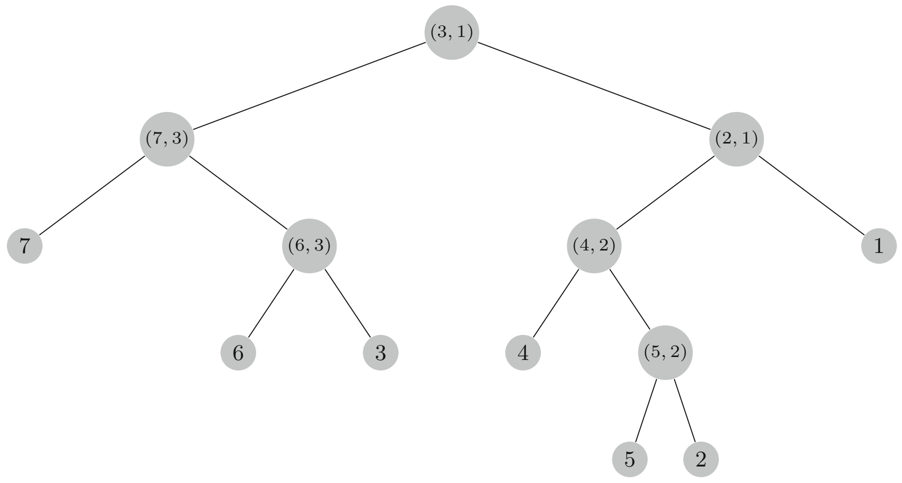
Theorem 2. *Let T be a rooted directed tree and let U be the underlying undirected tree for T. Then we have $R^{\bullet}(T) = \chi'_e(U) + 1$.*



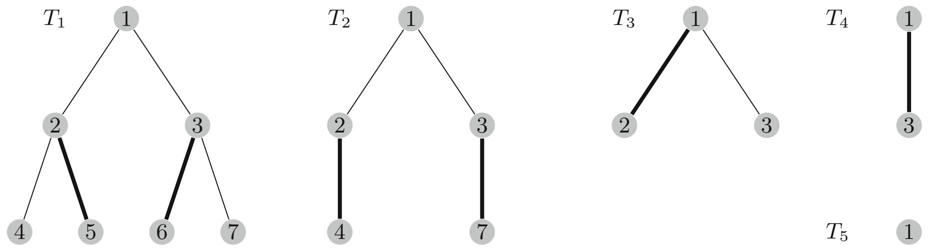
(a) The complete binary tree of height 3



(b) Optimal edge rank colouring



(c) Optimal strategy tree



(d) Optimal matching sequence

Fig. 1. This figure illustrates the equivalence between persistent reversible pebbling, matching game and edge rank coloring on trees by showing an optimal strategy tree and the corresponding matching sequence and edge rank colouring for height 3 complete binary tree

Corollary 1. *$R^{\phi}(T)$ and $R^{\bullet}(T)$ along with strategy trees achieving the optimal pebbling value can be computed in polynomial time for trees.*

Proof. We show that TREE-PEBBLE and TREE-VISITING-PEBBLE are polynomial time equivalent. Let T be an instance of TREE-PEBBLE. Pick an arbitrary leaf v of T and root the tree at v. By Theorem 2, the reversible pebbling number of this tree is the same as that of T. Let T' be the subtree rooted at the child of v. Then we have $R^{\bullet}(T) \leq k \iff R^{\phi}(T') \leq k-1$.

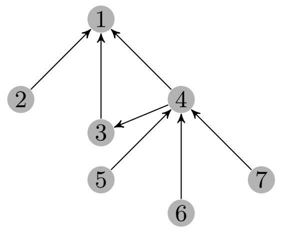
Let T be an instance of TREE-VISITING-PEBBLE. Let T' be the tree obtained by adding the edge (r, r') to T where r is the root of T. Then we have $R^{\phi}(T) \leq k \iff R^{\bullet}(T') \leq k+1$.

The statement of the theorem follows from Theorem 2 and the linear-time algorithm for finding an optimal edge rank coloring of trees[8]. □

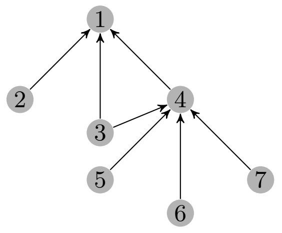
The following corollary is immediate from Theorem 1.

Corollary 2. *For any rooted directed tree T, we can compute $DT(T)$ and $RM(T)$ in polynomial time.*

An interesting consequence of Theorem 2 is that the persistent reversible pebbling number of a tree depends only on its underlying undirected graph. We remark that this does not generalize to DAGs. Below we show two DAGs with the same underlying undirected graph and different pebbling numbers.



(a) $R^{\bullet}(G_1) = 5$



(b) $R^{\bullet}(G_2) = 6$

Fig. 2. DAGs G_1 and G_2 have the same underlying undirected graph and different persistent pebbling numbers

4 Time Upper-Bound for an Optimal Pebbling of Complete Binary Trees

Proposition 1. *The following statements hold.*

1. $R^{\bullet}(Bt_h) \geq R^{\bullet}(Bt_{h-1}) + 1$
2. $R^{\bullet}(Bt_h) \geq h + 2$ *for* $h \geq 3$
3. *([1])* $R^{\bullet}(Ch_n) \leq \lceil \log_2(n) \rceil + 1$ *for all* n

Proof. (1) In any persistent pebbling of Bt_h, consider the earliest time after pebbling the root at which one of the subtrees of the root vertex has $R^{\phi}(Bt_{h-1})$ pebbles. At this time, there is a pebble on the root and there is at least one pebble on the other subtree of the root vertex. So, in total, there are at least $R^{\phi}(Bt_{h-1}) + 2 \geq R^{\bullet}(Bt_{h-1}) + 1$ pebbles on the tree.

(2) Item (1) and the fact that $R^{\bullet}(Bt_3) = 5$. □

Theorem 3. *There exists an optimal pebbling of Bt_h that takes at most $n^{O(\log\log(n))}$ steps.*

Proof. We will describe an optimal upstream pebbler in a pebbler-challenger game who pebbles $root(Bt_h)$, $left(root(Bt_h))$, $left(right(root(Bt_h)))$ and so on. In general, the pebbler pebbles $left(right^{i-1}(root(Bt_h)))$ in the i^{th} step for $1 \leq i < h - \log(h)$. An upper bound on the number of steps taken (denoted by $t(h)$) by the reversible pebbling obtained from this game (which is, recursively pebble $left(right^{i-1}(root(Bt_h)))$ for $0 \leq i < h - \log(h)$ and optimally pebble the remaining tree $Ch_{h-\log(h)} + Bt_{\log(h)}$ using any algorithm) is given below. Here the term $(2h - \log(h) + 1)^{3\log(h)}$ is an upper bound on the number of different pebbling configurations with $3\log(h)$ pebbles, and therefore an upper bound for time taken for optimally pebbling the tree $Ch_{h-\log(h)} + Bt_{\log(h)}$.

$$\begin{aligned}
t(h) &\leq 2\left[t(h-1) + t(h-2) + \ldots + t(\log(h)+1)\right] + (2h - \log(h) + 1)^{3\log(h)} \\
&\leq 2ht(h-1) + (2h - \log(h) + 1)^{3\log(h)} \\
&= O\left((2h)^h (2h)^{3\log(h)}\right) \\
&= (\log(n))^{O(\log(n))} = n^{O(\log\log(n))}
\end{aligned}$$

In the first step, the pebbler will place a pebble on $left(root(Bt_h))$ and the challenger will re-challenge the root vertex. These moves are optimal. Before the i^{th} step, the tree has pebbles on the root and $left(right^{j}(root(Bt_h)))$ for $0 \leq j < i - 1$. We argue that if $i < h - \log(h)$, placing a pebble on $left(right^{i-1}(root(Bt_h)))$ is an optimal move. If the pebbler makes this move, then the cost of the game is $\max(R^{\bullet}(Bt_{h_1-1}), R^{\bullet}(Ch_i + Bt_{h_1-1})) = R^{\bullet}(Ch_i + Bt_{h_1-1}) \leq R^{\bullet}(Bt_{h_1-1}) + 1 = p$, where $h_1 = h - i + 1$. Note that the inequality here is true when $i < h - \log(h)$ by Prop 1. We consider all other possible pebble placements on i^{th} step and prove that all of them are inferior.

- *A pebble is placed on the path from the root to $right^{i-1}(root(Bt_h))$ (inclusive)*: The challenger will challenge the vertex on which this pebble is placed. The cost of this game is then at least $R^{\bullet}(Bt_{h_1}) \geq p$.
- *A pebble is placed on a vertex with height less than $h_1 - 1$*: The challenger will re-challenge the root vertex and the cost of the game is at least $R^{\bullet}(Ch_i + Bt_{h_1-1})$.

The theorem follows. □

5 Almost Optimal Pebblings of Complete Binary Trees

In this section, we show that we can get arbitrarily close to optimal pebblings for complete binary trees using a polynomial number of steps.

Theorem 4. *For any constant $\epsilon > 0$, we can pebble Bt_h using at most $(1+\epsilon)h$ pebbles and $n^{O(\log(1/\epsilon))}$ steps for sufficiently large h.*

Proof. Let $k \geq 1$ be an integer. Then consider the following pebbling strategy parameterized by k.

1. Recursively pebble the subtrees rooted at $left(right^i(root(Bt_h)))$ for $0 \leq i \leq k-1$ and $right^k(root(Bt_h))$.
2. Leaving the $(k+1)$ pebbles on the tree (from the previous step), pebble the root vertex using an additional k pebbles in $2k-1$ steps.
3. Retaining the pebble on the root, reverse step (1) to remove every other pebble from the tree.

The number of pebbles and the number of steps used by the above strategy on Bt_h for sufficiently large h is given by the following recurrences.

$$S(h) \leq S(h-k) + (k+1) \leq \frac{(k+1)}{k}h$$

$$T(h) \leq 2\left[\sum_{i=1}^{k} T(h-i)\right] + (2k+2) \leq (2k)^h(2k+2) \leq n^{\log(k)+1}(2k+2)$$

where n is the number of vertices in Bt_h.

If we choose $k > 1/\epsilon$, then the theorem follows. □

6 Time-Space Trade-Offs for Bounded-Degree Trees

In this section, we study time-space trade-offs for bounded-degree trees.

Theorem 5. *For any constant positive integer k, a bounded-degree tree T consisting of n vertices can be pebbled using at most $O\left(n^{1/k}\right)$ pebbles and $O\left(2^k n\right)$ pebbling moves.*

Proof. Let us prove this by induction on the value of k. In the base case $(k=1)$, we are allowed to use $O(n)$ pebbles. So, the best strategy would to place a pebble on every vertex of T in bottom-up fashion, starting from the leaf vertices. After the root is pebbled, we unpebble each vertex in exactly the reverse order, while leaving the root pebbled.

In this strategy, clearly, each vertex is pebbled and unpebbled at most once. Hence the number of pebbling moves must be bounded by $2n$. Hence, a tree can be pebbled using $O(n)$ pebbles in $O(2n)$ moves.

Now consider that for $k \leq k_0 - 1$, where k_0 is an integer ≥ 2, any bounded-degree tree T with n vertices can be pebbled using $O\left(n^{1/k}\right)$ pebbles in $O\left(2^k n\right)$ moves. Assume that we are allowed $O\left(n^{1/k_0}\right)$ pebbles. To apply induction, we will be decomposing the tree into smaller components. We prove the following.

Claim. Let T' be any bounded-degree tree with $n' > n^{(k_0-1)/k_0}$ vertices and maximum degree Δ. There exists a subtree T'' of T' such that the number of vertices in T'' is at least $\lfloor n^{(k_0-1)/k_0}/2 \rfloor$ and at most $\lceil n^{(k_0-1)/k_0} \rceil$.

Proof. From the classical tree-separator theorem, we know that T' can be divided into two subtrees, where the larger subtree has between $\lfloor n'/2 \rfloor$ and $\left\lceil n' \cdot \dfrac{\Delta}{\Delta+1} \right\rceil$ vertices. The key is to recursively subdivide the tree in this way and continually choose the larger subtree. However, we need to show that in doing this we will definitely strike upon a subtree with the number of vertices within the required range. Let $T'_1, T'_2, \ldots$ be the sequence of subtrees we obtain in these iterations. Also let n_i be the number of vertices in T'_i for every i. Note that $\forall i, \lfloor n_i/2 \rfloor \leq n_{i+1} \leq \left\lceil v_i \cdot \dfrac{\Delta}{\Delta+1} \right\rceil$. Assume that j is the last iteration where $n_j > \lceil n^{(k_0-1)/k_0} \rceil$. Clearly $n_{j+1} \geq \lfloor n^{(k_0-1)/k_0}/2 \rfloor$. Also, by the definition of j, $n_{j+1} \leq \lceil n^{(k_0-1)/k_0} \rceil$. Hence the proof. □

The final strategy will be as follows:

1. Separate the tree into $\theta(n^{1/k_0})$ connected subtrees, each containing $\theta(n^{(k_0-1)/k_0})$ vertices. Claim 6 shows that this can always be done.
2. Let us number these subtrees in the following inductive fashion: denote by T_1, the 'lowermost' subtree, i.e. every path to the root of T_1 must originate from a leaf of T. Denote by T_i, the subtree for which every path to the root originates from either a leaf of T or the root of some T_j for $j < i$. Also, let n_i denote the number of vertices in T_i.
3. Pebble T_1 using $O\left(n_1^{1/(k_0-1)}\right) = O\left(n^{1/k_0}\right)$ pebbles. From the induction hypothesis, we know that this can be done using $O\left(2^{k_0-1} n_1\right)$ pebbling moves.
4. Retaining the pebble on the root vertex of T_1, proceed to pebble T_2 in the same way as above. Continue this procedure till the root vertex of T is pebbled. Then proceed to unpebble every vertex other than the root of T by executing every pebble move upto this instant in reverse order.

Now we argue the bounds on the number of pebbles and pebbling moves of the algorithm. Recall that the number of these subtrees is $O\left(n^{1/k_0}\right)$. Therefore, the number of intermediate pebbles at the root vertices of these subtrees is $O\left(n^{1/k_0}\right)$. Additionally, while pebbling the last subtree, $O\left(n^{1/k_0}\right)$ pebbles are used. Therefore, the total number of pebbles at any time remains $O\left(n^{1/k_0}\right)$. Each of the subtrees are pebbled and unpebbled once (effectively pebbled twice). Thus, the total number of pebbling moves is at most $\sum_i 2O\left(2^{k_0-1} n_i\right) = O\left(2^{k_0} n\right)$. □

7 Discussion and Open Problems

We studied reversible pebbling on trees. Although there are polynomial time algorithms for computing black and black-white pebbling numbers for trees, it was unclear, prior to our work, whether the reversible pebbling number for trees could be computed in polynomial time. We also established that almost optimal pebbling can be done in polynomial time.

We conclude with the following open problems.

– Prove or disprove that there is an optimal pebbling for complete binary trees that takes at most $O\left(n^k\right)$ steps for a fixed k.
– Prove or disprove that the there is a constant k such that optimal pebbling for any tree takes at most $O\left(n^k\right)$ (for black and black-white pebble games, this statement is true with $k = 1$).
– Give a polynomial time algorithm for computing optimal pebblings of trees that take the smallest number of steps.

References

1. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM Journal of Computing **18**(4), 766–776 (1989)
2. Chan, S.M.: Just a pebble game. In: Proceedings of the 28th Conference on Computational Complexity (CCC), pp. 133–143 (2013)
3. Chan, S.M.: Pebble Games and Complexity. PhD thesis, EECS Department, University of California, Berkeley, August 2013
4. Dymond, P.W., Tompa, M.: Speedups of deterministic machines by synchronous parallel machines. Journal of Computer and System Sciences **30**(2), 149–161 (1985)
5. Gilbert, J.R., Lengauer, T., Tarjan, R.E.: The pebbling problem is complete in polynomial space. SIAM Journal on Computing **9**(3), 513–524 (1980)
6. Hertel, P., Pitassi, T.: The pspace-completeness of black-white pebbling. SIAM J. Comput. **39**(6), 2622–2682 (2010)
7. Král'ovic, R.: Time and space complexity of reversible pebbling. In: Pacholski, L., Ružička, P. (eds.) SOFSEM 2001. LNCS, vol. 2234, p. 292. Springer, Heidelberg (2001)
8. Lam, T.W., Yue, F.L.: Optimal edge ranking of trees in linear time. In: Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 436–445 (1998)
9. Loui, M.C.: The space complexity of two pebbles games on trees. Technical Report MIT/LCS/TM-133, Massachusetts Institute of Technology (1979)
10. Raz, R., McKenzie, P.: Separation of the monotone NC hierarchy. Combinatorica **19**(3), 403–435 (1999). Conference version appeared in proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS 1997, Pages 234–243)
11. Sethi, R.: Complete register allocation problems. SIAM Journal on Computing, pp. 226–248(1975)
12. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, September 2000
13. Yannakakis, M.: A polynomial algorithm for the min-cut linear arrangement of trees. Journal of the ACM **32**(4), 950–988 (1985)

Computational Complexity

Combinations of Some Shop Scheduling Problems and the Shortest Path Problem: Complexity and Approximation Algorithms

Kameng Nip, Zhenbo Wang$^{(\boxtimes)}$, and Wenxun Xing

Department of Mathematical Sciences, Tsinghua University, Beijing, China
zwang@math.tsinghua.edu.cn

Abstract. We study several combinatorial optimization problems which combine the classic shop scheduling problems (open shop scheduling or job shop scheduling) and the shortest path problem. The objective of the considered problem is to select a subset of jobs that forms a feasible solution of the shortest path problem, and to execute the selected jobs on the open shop or job shop machines such that the makespan is minimized. We show that these problems are NP-hard even if the number of machines is two, and they cannot be approximated within a factor of less than 2 if the number of machines is an input unless P = NP. We present several approximation algorithms for these problems.

Keywords: Approximation algorithm · Combination of optimization problems · Job shop · Open shop · Scheduling · Shortest path

1 Introduction

Combinatorial optimization involves many active subfields, e.g. network flows, scheduling, bin packing. Usually these subfields are motivated by various applications or theoretical interests, and separately developed. The development of science and technology makes it possible to integrate manufacturing, service and management. At the same time, the decision-makers always need to deal with problems involving more than one combinatorial optimization problems. For instance, the network monitoring scenario described in [17] and the railway manufacturing scenario [12].

Wang and Cui [17] introduced a problem combining two classic combinatorial optimization problems, namely parallel machine scheduling and the vertex cover problem. The combination problem is to select a subset of jobs that forms a vertex cover, and to schedule it on some identical parallel machines such that the makespan is minimized. This work also inspired the study of the combination of different combinatorial optimization problems.

Flow shop, open shop and job shop are three basic models of multi-stage scheduling problems. Nip and Wang [12] studied a combination problem that combines two-machine flow shop scheduling and the shortest path problem.

DOI: 10.1007/978-3-319-21398-9_8

They argued that this problem is NP-hard, and proposed two approximation algorithms with worst-case ratio 2 and $\frac{3}{2}$ respectively. Recently, Nip et al. [13] extended the results to the case that the number of flow shop machines is arbitrary. One motivation of this problem is manufacturing rail racks. We plan to build a railway between two cities. How should we choose a feasible path in a map, such that the corresponding rail tracks (jobs) can be manufactured on some shop machines as early as possible? Similar scenarios can be found in telecommunications and other transportation industries. It connects two classic combinatorial optimization problems, say shop scheduling and the shortest path problem. An intuitive question is what will happen if the shop environment is one of the other two well-known shop environments, i.e. open shop and job shop. This is the core motivation for this current work. In this paper, we mainly study two problems: the combination of open shop scheduling and the shortest path problem, and the combination of job shop scheduling and the shortest path problem.

The contributions of this paper are described as follows: (1) we argue that these combination problems are NP-hard even if the number of machines is two, and if the number of machines is an input, these problems cannot be approximated within a factor less than 2 unless P = NP; (2) we present several approximation algorithms with performance ratio summarized as follows in which $\epsilon > 0$ is any constant and μ is the maximum operations per job in job shop scheduling.

Table 1. Performance of our algorithms

Number of Machines	Open Shop	Job Shop
2	FPTAS	$\frac{3}{2} + \epsilon$*
m (fixed)	PTAS**	$O\left(\frac{\log^2(m\mu)}{\log\log(m\mu)}\right)$
m (input)	m	m

* Assume that each job has at most 2 operations.
** A $(2 + \epsilon)$-approximation algorithm is also proposed.

The rest of the paper is organized as follows. In Section 2, we give a formal definition of the combination problems stated above, and briefly review some related problems and algorithms that will be used subsequently. In Section 3, we study the computational complexity of these combination problems and give an inapproximability result when the number of machines is an input. Section 4 provides several approximation algorithms for these problems. Some concluding remarks are provided in Section 5.

2 Preliminaries

2.1 Problem Description

We first recall the definitions of open shop and the job shop scheduling problems in the literatures.

Given a set of n jobs $J = \{J_1, \cdots, J_n\}$ and m machines $M = \{M_1, \cdots, M_m\}$, each job has several operations. At the same time, each machine can process at most one job and each job can be processed on one machine. In the open shop scheduling problem ($Om||C_{\max}$), each job must be processed on each machine exactly once, but the processing order can be arbitrary (in other words, the sequence of machines through which job passes can differ between jobs). In the job shop scheduling ($Jm||C_{\max}$), the processing order of each job is given in advance, and may differ between jobs. Furthermore, each job is allowed to be processed on the same machine more than once but consecutive operations of the same job must be processed on different machines, and is not necessary to go through all machines in the job shop. The goal of $Om||C_{\max}$ or $Jm||C_{\max}$ is to find a feasible schedule such that the makespan, that is, the completion time of the last stage among all the jobs is minimum.

Now we define the combination problems considered in this paper.

Definition 1 **($Om|\text{shortest path}|C_{\max}$).** *Given a directed graph $G = (V, A)$ with two distinguished vertices $s, t \in V$, and m machines. Each arc $a_j \in A$ corresponds to a job $J_j \in J$. The $Om|\text{shortest path}|C_{\max}$ problem is to find an $s-t$ directed path P of G, and to schedule the jobs of J_P on the open (job) shop machines such that the minimum makespan over all P, where J_P denotes the set of jobs corresponding to the arcs in P.*

Definition 2 **($Jm|\text{shortest path}|C_{\max}$).** *Given a directed graph $G = (V, A)$ with two distinguished vertices $s, t \in V$, and m machines. Each arc $a_j \in A$ corresponds to a job $J_j \in J$. The $Jm|\text{shortest path}|C_{\max}$ problem is to find an $s-t$ directed path P of G, and to schedule the jobs of J_P on the job shop machines such that the minimum makespan over all P, where J_P denotes the set of jobs corresponding to the arcs in P.*

Let the number of jobs (arcs) be n, i.e. $|A| = |J| = n$. Let p_{ij} be the processing times for J_j on machine M_i, and μ_{ij} be the frequency of J_j processed on M_i. Notice $\mu_{ij} = 1$ in the open shop.

It is not difficult to see that the shop scheduling problem (open shop or job shop) and the classic shortest path problem are special cases of our problems. For example, consider the following instances with $m = 2$ ([13]). If there is a unique path from s to t in G, as shown in the left of Fig. 1, our problem is the two-machine shop scheduling problem (open shop or job shop). If all the processing times on the second machine are zero, as shown in the right of Fig. 1, then our problem is equivalent to the classic shortest path with respect to the processing times on the first machine. Therefore we say the considered problems are the combinations of the shop scheduling problems and the shortest path problem.

In this paper, we will use the results of some optimization problems that have a similar structure to the classic shortest path problem. We introduce the generalized shortest path problem defined in [13].

Definition 3. *Given a weighted directed graph $G = (V, A, w^1, \cdots, w^K)$ and two distinguished vertices $s, t \in V$ with $|A| = n$, each arc $a_j \in A, j = 1, \cdots, n$ is*

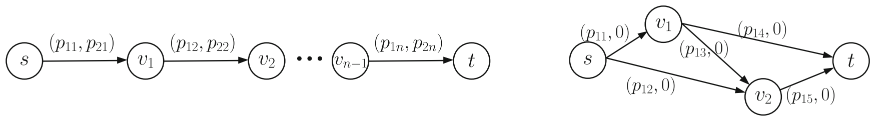


Fig. 1. Special cases of our problems

associated with K weights $w_j^1, \cdots, w_j^K$, and we define vector $w^k = (w_1^k, w_2^k, \cdots, w_n^k)$ for $k = 1, 2, \cdots, K$. The goal of our shortest path problem $SP(G, s, t, f)$ is to find an $s-t$ directed path P that minimizes $f(w^1, w^2, \cdots, w^K; x)$, in which f is a given objective function and $x \in \{0, 1\}^n$ contains the decision variables such that $x_j = 1$ if and only if $a_j \in P$.

For simplicity of notation, we denote SP instead of $SP(G, s, t, f)$ in the rest of the paper. Notice SP is a generalization of various shortest path problems. For example, if we set $K = 1$ and $f(w^1, x) = w^1 \cdot x$, where $\cdot$ is the dot product, it is the classic shortest path problem. If $f(w^1, w^2, \cdots, w^K; x) = \max\{w^1 \cdot x, w^2 \cdot x, \cdots, w^K \cdot x\}$, it is the min-max shortest path problem [1].

2.2 Review of Open Shop and Job Shop Scheduling

Gonzalez and Sahni [5] first gave a linear time optimal algorithm for $O2||C_{\max}$. They also proved that $Om||C_{\max}$ is NP-hard for $m \geq 3$, however whether it is strongly NP-hard is still an outstanding open problem. A feasible shop schedule is called dense when any machine is idle if and only if there is no job that could be processed on it. Rácsmány (see Bárány and Fiala [2]) observed that for any dense schedule, the makespan is at most twice that of the optimal solution, which leads to a greedy algorithm. Sevastianov and Woeginger [15] presented a PTAS for fixed m, which is obtained by dividing jobs into large jobs and small jobs. Their algorithm first optimally schedules the large jobs, then fills the operations of the small jobs into the 'gaps'. In this paper, we will use these algorithms, and refer to them as the GS algorithm, Rácsmány algorithm and the SW algorithm respectively. We present the main results of these algorithms as follows.

Theorem 1 ([5]). *The GS algorithm returns an optimal schedule for $O2||C_{\max}$ in linear time such that $C_{\max} = \max\left\{\max_{J_j \in J}(p_{1j} + p_{2j}), \sum_{J_j \in J} p_{1j}, \sum_{J_j \in J} p_{2j}\right\}$.*

Theorem 2 ([2,16]). *Rácsmány algorithm returns a 2-approximation algorithm for $Om||C_{\max}$ such that $C_{\max} \leq \sum_{J_j \in J} p_{lj} + \sum_{i=1}^{m} p_{ik} \leq 2C^*_{\max}$, where J_k is the last completed job and it is processed on M_l, and $C^*_{\max}$ denotes the optimal makespan.*

Theorem 3 ([15]). *The SW algorithm is a PTAS for $Om||C_{\max}$.*

For job shop scheduling problems, few polynomially solvable cases are known. One is $J2|op \leq 2|C_{\max}$, which can be solved by Jackson's rule [6] that is an extension of Johnson's rule for $F2||C_{\max}$ (flow shop scheduling problem with two machines [8]), where $op \leq 2$ means there are at most 2 operations per job.

In fact, a slight change may lead to NP-hard problems. For instance, $J2|op \leq 3|C_{\max}$ and $J3|op \leq 2|C_{\max}$ are NP-hard [9], $J2|p_{ij} \in \{1, 2\}|C_{\max}$ and $J3|p_{ij} = 1|C_{\max}$ are strongly NP-hard [10]. For the general case $J||C_{\max}$, Shmoys, Stein and Wein [16] constructed a randomized approximation algorithm with worst-case ratio $O\left(\frac{\log^2(m\mu)}{\log\log(m\mu)}\right)$, where μ is the maximum number of operations per job. Schmidt, Siegel and Srinivasan [14] obtained a deterministic algorithm with the same bound by derandomizing. We refer to it as the SSW-SSS algorithm. Moreover, for fixed m, the best known approximation algorithm is also proposed in [16] with an approximation factor $2 + \epsilon$, where $\epsilon > 0$ is an arbitrary constant. If μ is a constant, the problem is denoted as $Jm|op \leq \mu|C_{\max}$ and admits a PTAS [7]. We list the main results mentioned above as follows.

Theorem 4 ([6]). *Jackson's rule solves $J2|op \leq 2|C_{\max}$ in $O(n \log n)$ time.*

Theorem 5 ([14,16]). *The SSW-SSS algorithm solves $Jm||C_{\max}$ in polynomial time, and returns a schedule with makespan*

$$O\left(\frac{\log^2(m\mu)}{\log\log(m\mu)}\left(\max_{i\in\{1,\cdots,m\}}\sum_{J_j\in J}\mu_{ij}p_{ij} + \max_{J_j\in J}\sum_{i=1}^{m}\mu_{ij}p_{ij}\right)\right).$$

Furthermore, a well-known inapproximability result is that $O||C_{\max}$, $F||C_{\max}$ and $J||C_{\max}$ cannot be approximated within $\frac{5}{4}$ unless P = NP [18]. Recently, Mastrolilli and Svensson [11] showed that $J||C_{\max}$ cannot be approximated within $O(\log(m\mu)^{1-\epsilon})$ for $\epsilon > 0$ based on a stronger assumption than P $\neq$ NP.

To conclude this subsection, we list some trivial bounds for a dense shop schedule. Denote by $C_{\max}$ the makespan of an arbitrary dense shop schedule with job set J, and we have

$$C_{\max} \geq \max_{i\in\{1,\cdots,m\}}\left\{\sum_{J_j\in J}\mu_{ij}p_{ij}\right\}, \tag{1}$$

and

$$C_{\max} \leq \sum_{J_j\in J}\sum_{i=1}^{m}\mu_{ij}p_{ij}. \tag{2}$$

For each job, we have

$$C_{\max} \geq \sum_{i=1}^{m}\mu_{ij}p_{ij}, \qquad \forall J_j \in J. \tag{3}$$

2.3 Review of Shortest Path Problems

It is well-known that Dijkstra algorithm solves the classic shortest path problem with nonnegative edge weights in $O(|V|^2)$ time [3]. We have mentioned the min-max shortest path problem, that is NP-hard even for $K = 2$, and Aissi, Bazgan and Vanderpooten proposed an FPTAS if K is a fixed number [1]. We refer to their algorithm as the ABV algorithm, which has the following result.

Theorem 6 ([1]). *Given $\epsilon > 0$, in a directed graph with K nonnegative weights on each arc, where K is a fixed number, the ABV algorithm finds a path P between two specific vertices satisfying $\max_{i\in\{1,2,\cdots,K\}}\left\{\sum_{a_j\in P} w_j^i\right\} \leq (1+\epsilon)\max\limits_{i\in\{1,2,\cdots,K\}}\left\{\sum_{a_j\in P'} w_j^i\right\}$ for any path P' between the two specified vertices, and the running time is $O(|A||V|^{K+1}/\epsilon^K)$.*

In this paper, sometimes we need to find the min-max shortest path among all the paths visiting some specified arcs if such a path exists. We propose a modified ABV algorithm for this problem, which will be involved in the complete version.

3 Computational Complexity

First, notice that $Om||C_{\max}$ and $Jm||C_{\max}$ are special cases of the corresponding combination problems, thus the combination problem is at least as hard as its component optimization problems. On the other hand, we know that $O2||C_{\max}$ and $J2|op \leq 2|C_{\max}$ are polynomially solvable. However, we can simply verify that the corresponding combination problems, say $O2|\text{shortest path}|C_{\max}$ and $J2|op \leq 2, \text{shortest path}|C_{\max}$, are NP-hard by adopting the same reduction proposed in Theorem 2 of [12] for the NP-hardness of $F2|\text{shortest path}|C_{\max}$. We summarize the results as Theorem 7.

Theorem 7. *$J2|\text{shortest path}|C_{\max}$ is strongly NP-hard; $O2|\text{shortest path}|C_{\max}$ and $J2|op \leq 2, \text{shortest path}|C_{\max}$ are NP-hard.*

Now we consider the case where the number of machines m is part of the input. Williamson et al.[18] showed that it is NP-hard to approximate $O||C_{\max}$, $F||C_{\max}$ or $J||C_{\max}$ within a factor less than $\frac{5}{4}$ by a reduction from the restricted versions of 3-SAT. They also showed that deciding if there is a scheduling of length at most 3 is in P. We show that for these problems combining with shortest path problem, deciding if there is a scheduling of length at most 1 is still NP-hard. Our proof is established by constructing a reduction from 3-Dimensional Matching (3DM) that is NP-complete [4].

Theorem 8. *For $O|\text{shortest path}|C_{\max}$, deciding if there is a scheduling of length at most 1 is NP-hard.*

Notice that the reduction in Theorem 8 is also valid for $F|\text{shortest path}|C_{\max}$ and $J|\text{shortest path}|C_{\max}$, since each job in the reduction has only one nonzero processing time. Therefore we have the following result.

Corollary 1. *The problems* $O|\text{shortest path}|C_{\max}$*,* $F|\text{shortest path}|C_{\max}$ *and* $J|\text{shortest path}|C_{\max}$ *do not admit an approximation algorithm with worst-case ratio less than* 2*, unless* $\text{P} = \text{NP}$*.*

To our knowledge, the best known inapproximability results based on $\text{P} \neq \text{NP}$ for $F||C_{\max}$, $O||C_{\max}$ and $J||C_{\max}$ are still $\frac{5}{4}$. The corollary implies that the combination problems of the three shop scheduling problems and the shortest path problem may have stronger inapproximability results than the original problems.

4 Approximation Algorithms

4.1 An Intuitive Algorithm for Arbitrary *m*

An intuitive algorithm was proposed for $F2|\text{shortest path}|C_{\max}$ in [12]. The idea is to find the classic shortest path by setting the weight of an arc to be the sum of processing times of its corresponding job, and then schedule the returned jobs by Johnson's rule. This simple idea can be extended to the combination problems we considered, even if the number of machines is an input.

Algorithm 1. The SD algorithm for $O|\text{shortest path}|C_{\max}$ ($J|\text{shortest path}|C_{\max}$)

```
1: Find the shortest path in $G$ with weights $w_j^1 := \sum_{i=1}^{m} \mu_{ij} p_{ij}$ by Dijkstra algorithm.
   For the returned path $P$, construct the job set $J_P$.
2: Obtain a dense schedule for the jobs of $J_P$ by an arbitrary open (job) shop scheduling algorithm. Let $\sigma$ be the returned job schedule and $C_{\max}$ the returned makespan,
   and denote the job set $J_P$ by $S$.
3: return $S$, $\sigma$ and $C_{\max}$.
```

Theorem 9. *For* $O|\text{shortest path}|C_{\max}$ *and* $J|\text{shortest path}|C_{\max}$*, the SD algorithm is m-approximated, and this bound is tight.*

4.2 A Unified Algorithms for Fixed *m*

In [12], a $\frac{3}{2}$-approximation algorithm was proposed for $F2|\text{shortest path}|C_{\max}$. The idea is to iteratively find a feasible path by the ABV algorithm [1] with two weights for each arc, and schedule the corresponding jobs by Johnson's rule, then adaptively modify the weights of arcs and repeat the procedures until we obtain a feasible schedule with good guarantee. We generalize this idea to solve the combination problems considered in this paper. We first propose a unified framework which is denoted as UAR(Alg, ρ, m), where Alg is a polynomial time algorithm used for shop scheduling, ρ is a control parameter to decide the termination rule of the iterations and the jobs to be modified, and m is the number of machines. The pseudocode of the UAR(Alg, ρ, m)algorithm is described by Algorithm 2.

By setting the appropriate scheduling algorithms and control parameters, we can derive algorithms for different combination problems. Notice that at most n jobs are modified in the UAR(Alg, ρ, m) algorithm, therefore the iterations

Algorithm 2. Algorithm UAR(Alg, ρ, m)

```
1: Initially,$(w_j^1, w_j^2, \cdots, w_j^m) := (\mu_{1j}p_{1j}, \mu_{2j}p_{2j}, \cdots, \mu_{mj}p_{mj})$, for $a_j \in A$ corresponding to $J_j$.
2: Given $\epsilon > 0$, use the ABV algorithm [1] to obtain a feasible path $P$ to $SP$, and construct the corresponding job set as $J_P$.
3: Schedule the jobs of $J_P$ by the algorithm $Alg$, denote the returned makespan as $C'_{\max}$, and the job schedule as $\sigma'$.
4: $S := J_P$, $\sigma := \sigma'$, $C_{\max} := C'_{\max}$, $D := \emptyset$, $M := (1+\epsilon) \sum_{J_j \in J} \sum_{i=1}^{m} \mu_{ij}p_{ij} + 1$.
5: while $J_P \cap D = \emptyset$ and there exists $J_j$ in $J_P$ satisfying $\sum_{i=1}^{m} \mu_{ij}p_{ij} \geq \rho C'_{\max}$ do
6:     for all jobs satisfy $\sum_{i=1}^{m} \mu_{ij}p_{ij} \geq \rho C'_{\max}$ in $J \backslash D$ do
7:         $(w_j^1, w_j^2, \cdots, w_j^m) := (M, M, \cdots, M)$, $D := D \cup \{J_j\}$.
8:     end for
9:     Use the ABV algorithm [1] to obtain a feasible path $P$ to $SP$, and construct the corresponding job set as $J_P$.
10:    Schedule the jobs of $J_P$ by the algorithm $Alg$, denote the returned makespan as $C'_{\max}$, and the job schedule as $\sigma'$.
11:    if $C'_{\max} < C_{\max}$ then
12:        $S := J_P$, $\sigma := \sigma'$, $C_{\max} := C'_{\max}$.
13:    end if
14: end while
15: return $S$, $\sigma$ and $C_{\max}$.
```

execute at most n times. Since the scheduling algorithms for shop scheduling and the ABV algorithm [1] are all polynomial time algorithms (for fixed m and ϵ), we claim that the following algorithms based on UAR(Alg, ρ, m) are polynomial-time algorithms. We present the algorithms and prove their performance as follows.

We first apply the UAR(Alg, ρ, m) algorithm to $O2|\text{shortest path}|C_{\max}$ by setting Alg be the GS algorithm [5] and $\rho = 1$. We refer to this algorithm as the GAR algorithm.

Algorithm 3. The GAR algorithm for $O2|\text{shortest path}|C_{\max}$

```
1: Let $m = 2$, $Alg$ be the GS algorithm [5] for $O2||C_{\max}$ and $\rho = 1$.
2: Solve the problem by using UAR($Alg$, $\rho$, $m$).
```

Theorem 10. *The GAR algorithm is an FPTAS for* $O2|\text{shortest path}|C_{\max}$.

We point out that the proofs of the worst-case performance of algorithms based on UAR(Alg, ρ, m) are quite similar. In the following proofs of this subsection, we will only describe the key ideas and main steps since the results can be obtained by analogous arguments. We will adopt the same notations as in the proof of Theorem 10, and also analyze the same two cases.

For $Om|\text{shortest path}|C_{\max}$ where m is fixed, we obtain the following RAR algorithm based on UAR(Alg, ρ, m) and Rácsmány algorithm [2,16].

Algorithm 4. The RAR algorithm for $Om|\text{shortest path}|C_{\max}$

```
1: Let $Alg$ be Rácsmány algorithm [2,16] for $Om||C_{\max}$ and $\rho = \frac{1}{2}$.
2: Solve the problem by using UAR($Alg$, $\rho$, $m$).
```

Theorem 11. *Given $\epsilon > 0$, the RAR algorithm is a $(2+\epsilon)$-approximation algorithm for $Om|\text{shortest path}|C_{\max}$.*

The framework can also be applied to the combination problem of job shop scheduling and the shortest path problem. For the combination of $J2|op \leq 2|C_{\max}$ and the shortest path problem, we obtain a $(\frac{3}{2}+\epsilon)$-approximation algorithm by using Jackson's rule and setting $\rho = \frac{2}{3}$ in the UAR(Alg, ρ, m) algorithm. We refer to this algorithm as the JJAR algorithm, and describe it in Algorithm 5. Recall that all $\mu_{ij} = 1$ in $J2|op \leq 2|C_{\max}$.

Algorithm 5. The JJAR algorithm for $J2|op \leq 2, \text{shortest path}|C_{\max}$

```
1: Let $m = 2$, $Alg$ be Jackson's rule for $J2|op \leq 2|C_{\max}$ and $\rho = \frac{2}{3}$.
2: Solve the problem by using UAR($Alg$, $\rho$, $m$).
```

Before studying the worst-case performance of the JJAR algorithm, we establish the following lemma. Let $(1 \rightarrow 2)$ ($(2 \rightarrow 1)$) indicate the order that a job needs to be processed on M_1 (M_2) first and then on M_2 (M_1).

Lemma 1. *For $J2|op \leq 2|C_{\max}$, let $C^J_{\max}$ be the makespan returned by Jackson's rule. Suppose we change the processing order of all jobs to be $(1 \rightarrow 2)$ ($(2 \rightarrow 1)$), and the processing times keep unchanged. Then schedule the jobs by Johnson's rule for $F2||C_{\max}$, and denote the makespan as $C^1_{\max}$ ($C^2_{\max}$). We have $C^J_{\max} \leq \max\{C^1_{\max}, C^2_{\max}\}$.*

Now we can study the performance of the JJAR algorithm for $J2|op \leq 2, \text{shortest path}|C_{\max}$.

Theorem 12. *Given $\epsilon > 0$, the JJAR algorithm is a $(\frac{3}{2}+\epsilon)$-approximation algorithm for $J2|op \leq 2, \text{shortest path}|C_{\max}$.*

Finally, we study the general case $Jm|\text{shortest path}|C_{\max}$, where m is fixed. By Theorem 5, we know that there exists $\alpha > 0$, such that the SSW-SSS algorithm [14,16] returns a schedule satisfying

$$C'_{\max} \leq \alpha \frac{\log^2(m\mu)}{\log\log(m\mu)} \left(\max_{i\in\{1,\cdots,m\}} \sum_{J_j \in J'} \mu_{ij} p_{ij} + \max_{j \in J'} \sum_{i=1}^{m} \mu_{ij} p_{ij} \right). \tag{4}$$

The factor α is decided by choosing the probability of the randomized steps and the subsequent operations in the SSW-SSS algorithm [14,16] [14,16], and its value can be obtained by complicated calculation. Assume we determine such value of α. We can design an approximation algorithm with worst-case ratio $O\left(\frac{\log^2(m\mu)}{\log\log(m\mu)}\right)$ for $Jm|\text{shortest path}|C_{\max}$. We refer to this algorithm as the SAR algorithm, and describe it in Algorithm 6.

Algorithm 6. The SAR algorithm for $Jm|\text{shortest path}|C_{\max}$

```
1: Let $Alg$ be the SSW-SSS algorithm [14,16] for $Jm||C_{\max}$ and $\rho = \frac{\log\log(m\mu)}{2\alpha\log^2(m\mu)}$.
2: Solve the problem by using UAR($Alg$, $\rho$, $m$).
```

Theorem 13. *The SAR algorithm is an* $O\left(\frac{\log^2(m\mu)}{\log\log(m\mu)}\right)$*-approximation algorithm for* $Jm|\text{shortest path}|C_{\max}$.

Remind that the SAR algorithm relies on the assumption, that we can determine the constant α for the SSW-SSS algorithm [14,16]. We can calculate it by following the details of the SSW-SSS algorithm [14,16], and in fact we can choose α large enough to guarantee the performance ratio of our algorithm.

4.3 A PTAS for $Om|\text{shortest path}|C_{\max}$

In the previous subsection, we introduced a $(2+\epsilon)$-approximation algorithm for $Om|\text{shortest path}|C_{\max}$ based on the UAR(Alg, ρ, m) algorithm. By a different approach, we propose a $(1+\epsilon)$-approximation algorithm for any $\epsilon > 0$, i.e. a PTAS. We also iteratively find feasible solutions, but guarantee that one of the returned solutions has the same first N-th largest jobs with an optimal solution where N is a given constant. Precisely speaking, we say job J_j is larger than job J_k if $\max_{i\in\{1,\cdots,m\}} p_{ij} > \max_{i\in\{1,\cdots,m\}} p_{ik}$. To do this, we enumerate all size N subsets J^N of J, and then iteratively modify the weights of the graph such that the jobs larger than any job in J^N will not be chosen. Then find a feasible solution which contains all the jobs in J^N corresponding to the modified graph, i.e., the corresponding path is constrained to visit all the arcs corresponding to J^N if such a path exists.

To find a feasible solution in each iteration, we adopt the modified ABV algorithm to obtain a near optimal min-max shortest path among all the paths visiting the arcs corresponding to J^N if such a path exists. Then we schedule the selected jobs by [15] which is denoted as the SW algorithm [15] for $Om||C_{\max}$. We refer to our algorithm as the SAE algorithm, and describe it in Algorithm 7.

There are $\binom{n}{N}$ distinct subsets J^N, thus the iterations between line 4 - line 12 run at most $O(n^N)$ times, that is a polynomial of n since N is a constant when m and ϵ are fixed. Since the modified ABV algorithm is an FPTAS and the SW algorithm [15] is a PTAS, the running time of each iteration is also bounded by the polynomial of n if m and ϵ are fixed. It suffices to show that the SAE algorithm terminates in polynomial time. The following theorem indicates the SAE algorithm is a PTAS.

Algorithm 7. The SAE algorithm for $Om|\text{shortest path}|C_{\max}$

```
1: Given $0 < \epsilon < 1$, set $N = m\left(\frac{m(3+\epsilon)}{\epsilon}\right)^{2^{\frac{m(3+\epsilon)}{\epsilon}}}$.
2: Let $D := \emptyset$, $M := (1 + \frac{\epsilon}{3}) \sum_{J_j \in J} \sum_{i=1}^{m} p_{ij} + 1$, $C_{\max} := \sum_{J_j \in J} \sum_{i=1}^{m} p_{ij}$.
3: Initially, $(w_j^1, w_j^2, \cdots, w_j^m) := (p_{1j}, p_{2j}, \cdots, p_{mj})$, for $a_j \in A$ corresponding to $J_j$.
4: for all $J^N \subset J$, with $|J^N| = N$ do
5:     $(w_j^1, w_j^2, \cdots, w_j^m) := (p_{1j}, p_{2j}, \cdots, p_{mj})$, $D := \emptyset$.
6:     For jobs $J_k \in J \setminus J^N$ with $\max_{i \in \{1,\cdots,m\}} p_{ik} > \min_{J_j \in J^N} \max_{i \in \{1,\cdots,m\}} p_{ij}$, set
       $(w_k^1, w_k^2, \cdots, w_k^m) := (M, M, \cdots, M)$, $D := D \cup \{J_k\}$.
7:     Use the modified ABV algorithm to obtain a feasible path $P$ of $SP$ such that
       the returned path visits all the arcs corresponding to $J^N$ if such a path exists.
       Construct the corresponding job set as $J_P$.
8:     Schedule the jobs of $J_P$ by the SW algorithm [15], denote the returned makespan
       as $C'_{\max}$, and the job schedule as $\sigma'$.
9:     if $C'_{\max} < C_{\max}$ then
10:        $S := J_P$, $\sigma := \sigma'$, $C_{\max} := C'_{\max}$.
11:    end if
12: end for
13: return $S$, $\sigma$, $C_{\max}$.
```

Theorem 14. *The SAE algorithm is a PTAS for* $Jm|\text{shortest path}|C_{\max}$.

5 Conclusions

This paper studies several problems combining two well-known combinatorial optimization problems. We show the hardness of the problems, and present some approximation algorithms. It is interesting to find approximation algorithms with better worst-case ratios for $J2|op \leq 2, \text{shortest path}|C_{\max}$ and $Jm|\text{shortest path}|C_{\max}$. Moreover, it needs further study to close the gap between the 2-inapproximability results and the m-approximation algorithms for $O|\text{shortest path}|C_{\max}$ and $J|\text{shortest path}|C_{\max}$. We can also consider other interesting combinations of combinatorial optimization problems.

Acknowledgments. Wang's research has been supported by NSFC No. 11371216 and Bilateral Scientific Cooperation Project between Tsinghua University and KU Leuven. Xing's research has been supported by NSFC No. 11171177.

References

1. Aissi, H., Bazgan, C., Vanderpooten, D.: Approximating min-max (regret) versions of some polynomial problems. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 428–438. Springer, Heidelberg (2006)
2. Bárány, I., Fiala, T.: Többgépes ütemezési problémák közel optimális megoldása (in Hungarian). Szigma - Matematikai - Közgazdasági Folyóirat **15**, 177–191 (1982)

3. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
5. Gonzalez, T., Sahni, S.: Open shop scheduling to minimize finish time. Journal of the Association for Computing Machinery **23**, 665–679 (1976)
6. Jackson, J.R.: An extension of Johnson's results on job-lot scheduling. Naval Research Logistics Quarterly **3**, 201–203 (1956)
7. Jansen, K., Solis-Oba, R., Sviridenko, M.: Makespan minimization in job shops: A linear time approximation scheme. SIAM Journal on Discrete Mathematics **16**, 288–300 (2003)
8. Johnson, S.M.: Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly **1**, 61–68 (1954)
9. Lenstra, J.K., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. Annals of Operations Research **1**, 343–362 (1977)
10. Lenstra, J.K., Rinnooy Kan, A.: Computational complexity of discrete optimization. Annals of Operations Research **4**, 121–140 (1979)
11. Mastrolilli, M., Svensson, O.: Hardness of approximating flow and job shop scheduling problems. Journal of the Association for Computing Machinery **58**, 20:1–20:32 (2011)
12. Nip, K., Wang, Z.: Combination of two-machine flow shop scheduling and shortest path problems. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 680–687. Springer, Heidelberg (2013)
13. Nip, K., Wang, Z., Talla Nobibon, F., Leus, R.: A Combination of Flow Shop Scheduling and the Shortest Path Problem. Journal of Combinatorial Optimization **29**, 36–52 (2015)
14. Schmidt, J.P., Siegel, A., Srinivasan, A.: Chernoff-hoeffding bounds for applications with limited independence. SIAM Journal on Discrete Mathematics **8**, 223–250 (1995)
15. Sevastianov, S.V., Woeginger, G.J.: Makespan minimization in open shops: A polynomial time approximation scheme. Mathematical Programming **82**, 191–198 (1998)
16. Shmoys, D.B., Stein, C., Wein, J.: Improved approximation algorithms for shop scheduling problems. SIAM Journal on Computing **23**, 617–632 (1994)
17. Wang, Z., Cui, Z.: Combination of parallel machine scheduling and vertex cover. Theoretical Computer Science **460**, 10–15 (2012)
18. Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevast'janov, S.V., Shmoys, D.B.: Short shop schedules. Operations Research **45**, 288–294 (1997)

Complexity of Grundy Coloring and Its Variants

Édouard Bonnet[1,2], Florent Foucaud[3], Eun Jung Kim[1], and Florian Sikora[1(✉)]

[1] PSL, LAMSADE - CNRS UMR 7243, Université Paris-Dauphine, Paris, France
{edouard.bonnet,eunjung.kim,florian.sikora}@dauphine.fr
[2] Hungarian Academy of Sciences, Budapest, Hungary
[3] LIMOS - CNRS UMR 6158, Université Blaise Pascal, Clermont-ferrand, France
florent.foucaud@gmail.com

Abstract. The Grundy number of a graph is the maximum number of colors used by the greedy coloring algorithm over all vertex orderings. In this paper, we study the computational complexity of Grundy Coloring, the problem of determining whether a given graph has Grundy number at least k. We show that Grundy Coloring can be solved in time $O^*(2.443^n)$ on graphs of order n. While the problem is known to be solvable in time $f(k, w) \cdot n$ for graphs of treewidth w, we prove that under the Exponential Time Hypothesis, it cannot be computed in time $O^*(c^w)$, for any constant c. We also study the parameterized complexity of Grundy Coloring parameterized by the number of colors, showing that it is in FPT for graphs including chordal graphs, claw-free graphs, and graphs excluding a fixed minor.

Finally, we consider two previously studied variants of Grundy Coloring, namely Weak Grundy Coloring and Connected Grundy Coloring. We show that Weak Grundy Coloring is fixed-parameter tractable with respect to the weak Grundy number. In stark contrast, it turns out that checking whether a given graph has connected Grundy number at least k is NP-complete already for $k = 7$.

1 Introduction

A *k-coloring* of a graph G is a surjective mapping $\varphi : V(G) \to \{1, \dots, k\}$ and we say v is colored with $\varphi(v)$. A k-coloring φ is *proper* if any two adjacent vertices receive different colors in φ. The *chromatic number* $\chi(G)$ of G is the smallest k such that G has a k-coloring. Determining the chromatic number of a graph is the most fundamental problem in graph theory. Given a graph G and an ordering $\sigma = v_1, \dots, v_n$ of $V(G)$, the *first-fit algorithm* colors vertex v_i with the smallest color that is not present among the set of its neighbors within $\{v_1, \dots, v_{i-1}\}$. The *Grundy number* $\Gamma(G)$ is the largest k such that G admits a vertex ordering on which the first-fit algorithm yields a proper k-coloring. First-fit is presumably the simplest heuristic to compute a proper coloring of

F. Foucaud—This research was done while this author was a postdoctoral fellow at the Department of Mathematics of University of Johannesburg (South Africa) and at LAMSADE.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 109–120, 2015.
DOI: 10.1007/978-3-319-21398-9_9

a graph. In this sense, the Grundy number gives an algorithmic upper bound on the performance of any heuristic for the chromatic number. This notion was first studied by Grundy in 1939 in the context of digraphs and games [11], and formally introduced 40 years later by Christen and Selkow [8]. Many works have studied the first-fit algorithm in connection with on-line coloring algorithms, see e.g. [21]. A natural relaxation of this concept is the *weak Grundy number*, introduced by Kierstead and Saoub [17], where the obtained coloring is not asked to be proper. A more restricted concept is the one of *connected Grundy number*, introduced by Benevides et al. [3], where the algorithm is given an additional "local" restriction: at each step, the subgraph induced by the colored vertices must be connected.

The goal of this paper is to advance the study of the computational complexity of determining the Grundy number, the weak Grundy number and the connected Grundy number of a graph.

Let us introduce the problems formally. Let G be a graph and let $\sigma = v_1, \ldots, v_n$ be an ordering of $V(G)$. A (not necessarily proper) k-coloring $\varphi : V(G) \to \{1, \ldots, k\}$ of G is a *first-fit coloring with respect to* σ if for every vertex v_i and every color c with $c < \varphi(v_i)$, v_i has a neighbor v_j with $\varphi(v_j) = c$ for some $j < i$. In particular, $\varphi(v_1) = 1$. A vertex ordering $\sigma = v_1, \ldots, v_n$ is *connected* if for every i, $1 \leqslant i \leqslant n$, the subgraph induced by $\{v_1, \ldots, v_i\}$ is connected. A k-coloring $\varphi : V(G) \to \{1, \ldots, k\}$ is called the (i) *weak Grundy*, (ii) *Grundy*, (iii) *connected Grundy coloring* of G, respectively, if it is a first-fit coloring with respect to some vertex ordering σ such that (i) φ and σ has no restriction, (ii) φ is proper, (iii) φ is proper and σ is connected, respectively.

The maximum number of colors used in a (weak, connected, respectively) Grundy coloring is called the (*weak, connected*, respectively) Grundy number and is denoted $\Gamma(G)$ ($\Gamma'(G)$ and $\Gamma_c(G)$, respectively). In this paper, we study the complexity of computing these invariants.

GRUNDY COLORING
Input: A graph G, an integer k.
Question: Do we have $\Gamma(G) \geqslant k$?

The problems WEAK GRUNDY COLORING and CONNECTED GRUNDY COLORING are defined analogously.

Note that $\chi(G) \leqslant \Gamma(G) \leqslant \Delta(G) + 1$, where $\chi(G)$ is the chromatic number and $\Delta(G)$ is the maximum degree of G. However, the difference $\Gamma(G) - \chi(G)$ can be (arbitrarily) large, even for bipartite graphs. For example, the Grundy number of the tree of Figure 1 is 4, whereas its chromatic number is 2. Note that this is not the case for Γ_c for bipartite graphs, since $\Gamma_c(G) \leqslant 2$ for any bipartite graph G [3]. However, the difference $\Gamma_c(G) - \chi(G)$ can be (arbitrarily) large even for planar graphs [3].

Previous Results. GRUNDY COLORING remains NP-complete on bipartite graphs [14] and their complements [25] (and hence claw-free graphs and P_5-free graphs), on chordal graphs [23], and on line graphs [13]. Certain graph classes admit polynomial-time algorithms. There is a linear-time algorithm for Grundy Coloring on trees [15]. This result was extended to graphs of bounded treewidth by Telle and Proskurowski [24], which proposed a dynamic programming algorithm running in time $k^{O(w)}2^{O(wk)}n = O(n^{3w^2})$ for graphs of treewidth w (in other words, their algorithm is in FPT for parameter $k + w$ and in XP for parameter w).[1] A polynomial-time algorithm for P_4-laden graphs, which contains all cographs as a subfamily, was given in [2].

Note that Grundy Coloring admits a polynomial-time algorithm when the number k of colors is fixed [26], in other words, it is in XP for parameter k.

Grundy Coloring has polynomial-time constant-factor approximation algorithms for inputs that are interval graphs [12,21], complements of chordal graphs [12], complements of bipartite graphs [12] and bounded tolerance graphs [17]. In general, however, there is a constant $c > 1$ s.t. approximating Grundy Coloring within c is impossible unless $\mathsf{NP} \subseteq \mathsf{RP}$ [18]. It is not known if a polynomial-time $o(n)$-factor approximation algorithm exists.

When parameterized by the graph's order minus the number of colors, Grundy Coloring was shown to be in FPT by Havet and Sempaio [14].

Connected Grundy Coloring was introduced by Benevides *et al.* [3], who proved it to be NP-complete, even for chordal graphs and for co-bipartite graphs. Weak Grundy Coloring is NP-complete [10].

Our Results. As pointed out in [24], no (extended) monadic second order expression is known for the property "$\Gamma(G) \geqslant k$". Therefore it is not clear whether the algorithm of [24] can be improved, e.g. to an algorithm of running time $f(w) \cdot poly(n)$. Nevertheless, on general graphs, we show that Grundy Coloring can be solved in time $O^*(2.443^n)$.

As a lower bound to the positive algorithmic bounds, we show that under the Exponential Time Hypothesis (ETH) [16], an $O(c^w \cdot poly(n))$-time algorithm for Grundy Coloring does not exist (for any fixed constant c). Hence the exponent n cannot be replaced by the treewidth in our $O^*(2.443^n)$-time algorithm.

We also study the parameterized complexity of Grundy Coloring parameterized by the number of colors, showing that it is in FPT for graphs including chordal graphs, claw-free graphs, and graphs excluding a fixed minor.

Finally, we show that Weak Grundy Coloring and Connected Grundy Coloring exhibit opposite computational behavior when viewed through the lense of parameterized complexity (for the parameter "number of colors"). While Weak Grundy Coloring is shown to be FPT on general graphs, Connected Grundy Coloring is NP-complete even when $k = 7$, i.e. does not belong to XP (it is the only of the three studied problems to be in this case). Note that the known NP-hardness proof for Connected Grundy Coloring was only for an unbounded number of colors [3].

[1] The first running time is not explicitly stated in [24] but follows from their metatheorem. The second one is deduced by the authors of [24] from the first one by bounding k by $w \log_2 n + 1$.

Due to space constraints, some proofs are deferred to the full version of the paper [6].

2 Preliminaries

We defer many (classic) technical definitions to the full version [6], and only give the ones related to Grundy colorings. Given a graph G, a *colored witness* of height ℓ, or simply called an ℓ-witness, is a subgraph G' of G, which comes with a partition $\mathcal{W} = W_1 \uplus \cdots \uplus W_\ell$ of $V(G')$ such that for every i in $1, \ldots, \ell$ (1) $W_i \neq \emptyset$, and (2) W_i is an independent dominating set of $G[W_i \cup \cdots \cup W_\ell]$. The cell W_i under $\mathcal{W}$ is called the *color class* of color i. A witness G' of height ℓ is said to be *minimal* if for every $u \in V(G')$, $G' - u$ with the partition $\mathcal{W}|_{V(G')-\{u\}}$ is not an ℓ-witness.

Observation 1. *For any graph G, $\Gamma(G) \geqslant k$ if and only if G allows a minimal k-witness.*

Observation 2. *A minimal k-witness has a vertex of degree $k-1$ (the root), order at most 2^{k-1}, and is included in the distance-k neighborhood of the root.*

By these observations, k-Grundy Coloring can be solved by checking, for every subset of 2^{k-1} vertices, if it contains a k-witness as an induced subgraph:

Corollary 3 ([26]). Grundy Coloring *can be solved in time $f(k)n^{2^{k-1}}$, i.e.* Grundy Coloring *parameterized by the number k of colors is in* XP.

Observation 4. *In any Grundy coloring of G, a vertex with degree d cannot be colored with color $d+2$ or larger.*

Proposition 5. *Let G be a graph with a minimal Grundy coloring achieving color k and let W be the corresponding minimal witness. Then, if a vertex u of W is colored with $k' < k$, u has a neighbor colored with some color $k'', k'' > k'$.*

Proof. If not, one could remove u from the witness, a contradiction.

Lemma 6. *Let G be a graph and let G' be the corresponding minimal ℓ-witness with the partition $\mathcal{W} := W_1 \uplus \cdots \uplus W_\ell$. Then, W_i is an independent set which dominates the set $\bigcup_{j \in [i+1,\ell]} W_j$ (and no proper subset of W_i has this property). In particular, W_1 is a minimal independent dominating set of $V(G')$.*

For each $i \in [l]$, let t_i be a rooted tree. We define $v[t_1, t_2, \ldots, t_l]$ as the tree rooted at node v where v is linked to the root of each tree t_i. The set $(T_k)_{k \geqslant 1}$ is a family of rooted trees (known as *binomial trees*) defined as follows (see Figure 1 for an illustration):

- T_1 consist only of one node (incidentally the root), and
- $\forall k \geqslant 1$, $T_{k+1} = v[T_1, T_2, \ldots, T_k]$.

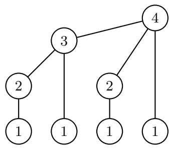


Fig. 1. The binomial tree T_4, where numbers denote the color of each vertex in a first-fit proper coloring with largest number of colors

In a tree T_k with root v, for each $i \in [k]$, $v(i)$ denotes the root of T_i (i.e. the i-th child of v).

We now show a useful lemma about Grundy colorings of the tree T_k.

Lemma 7. *The Grundy number of T_k is k. Moreover, there are exactly two Grundy colorings achieving color k, and a unique coloring if we impose that the root is colored k.*

The following result of Chang and Hsu [7] will prove useful:

Theorem 8 ([7]). *Let G be a graph on n vertices for which every subgraph H has at most $d|V(H)|$ edges. Then $\Gamma(G) \leqslant \log_{d+1/d}(n) + 2$.*

3 Grundy Coloring: Algorithms and Complexity

3.1 An Exact Algorithm

A straightforward way to solve Grundy Coloring is to enumerate all possible orderings of the vertex set and to check whether the greedy algorithm uses at least k colors. This is a $\Theta(n!)$-time algorithm. A natural question is whether there is a faster exact algorithm. We now give such an algorithm.

We rely on two observations: (a) in a colored witness, every color class W_i is an independent dominating set in $G[\bigcup_{j \geqslant i} W_j]$ (Lemma 6), and (b) any independent dominating set is a maximal independent set (and vice versa). The algorithm is obtained by dynamic programming over subsets, and uses an algorithm which enumerates all maximal independent sets.

Theorem 9. Grundy Coloring *can be solved in time $O^*(2.44225^n)$.*

Proof. Let $G = (V, E)$ be a graph. We present a dynamic programming algorithm to compute $\Gamma(G)$. For simplicity, given $S \subseteq V$, we denote the Grundy number of the induced subgraph $G[S]$ by $\Gamma(S)$. We recursively fill a table $\Gamma^*(S)$ over the subset lattice $(2^V, \subseteq)$ of V in a bottom-up manner starting from $S = \emptyset$. The base case of the recursion is $\Gamma^*(\emptyset) = 0$. The recursive formula is given as

$$\Gamma^*(S) = \max\{\Gamma^*(S \setminus X) + 1 \mid X \subseteq S \text{ is an independent dominating set of } G[S]\}.$$

Now let us show by induction on $|S|$ that $\Gamma^*(S) = \Gamma(S)$ for all $S \subseteq V$. The assertion trivially holds for the base case. Consider a nonempty subset $S \subseteq V$;

by induction hypothesis, $\Gamma^*(S') = \Gamma(S')$ for all $S' \subset S$. Let X be a subset of S achieving $\Gamma^*(S) = \Gamma^*(S \setminus X) + 1$ and X' be the set of the color class 1 in the ordering achieving the Grundy number $\Gamma(S)$.

Let us first see that $\Gamma^*(S) \leqslant \Gamma(S)$. By induction hypothesis we have $\Gamma^*(S \setminus X) = \Gamma(S \setminus X)$. Consider a vertex ordering σ on $S \setminus X$ achieving $\Gamma(S \setminus X)$. Augmenting σ by placing all vertices of X at the beginning of the sequence yields a (set of) vertex ordering(s). Since X is an independent set, the first-fit algorithm gives color 1 to all vertices in X, and since X is also a dominating set for $S \setminus X$, no vertex of $S \setminus X$ receives color 1. Therefore, the first-fit algorithm on such ordering uses $\Gamma(S \setminus X) + 1$ colors. We deduce that $\Gamma(S) \geqslant \Gamma(S \setminus X) + 1 = \Gamma^*(S \setminus X) + 1 = \Gamma^*(S)$.

To see that $\Gamma^*(S) \geqslant \Gamma(S)$, we first observe that $\Gamma(S \setminus X') \geqslant \Gamma(S) - 1$. Indeed, the use of the optimal ordering of S ignoring vertices of X' on $S \setminus X'$ yields the color $\Gamma(S) - 1$. We deduce that $\Gamma(S) \leqslant \Gamma(S \setminus X') + 1 = \Gamma^*(S \setminus X') + 1 \leqslant \Gamma^*(S \setminus X) + 1 = \Gamma^*(S)$.

As a minimal independent dominating set is a maximal independent set, we can estimate the computation of the table by restricting X to the family of maximal independent sets of $G[S]$. On an n-vertex graph, one can enumerate all maximal independent sets in time $O(1.44225^n)$ [20]. Checking whether a given set is a minimal independent set is polynomial and thus, the number of execution steps is dominated (up to a polynomial factor) by the number of recursion steps taken. This is

$$\sum_{i=0}^{n} \binom{n}{i} \cdot 1.44225^i = (1 + 1.44225)^n. \square$$

We leave as an open question to improve this running time. However, we note that the *fast subset convolution* technique [4] does not seem to be directly applicable.

3.2 Lower Bound on the Treewidth Dependency

Let us recall that GRUNDY COLORING is known to be in XP for the parameter treewidth, but its membership in FPT remains open.

The following result is inspired by ideas in [19] for proving near-optimality of known algorithm on bounded treewidth graphs. Unlike [19] which is based on the *Strong* ETH, our result is based on the ETH.

Theorem 10. *Under the ETH, for any constant c,* GRUNDY COLORING *is not solvable in time $O^*(c^w)$ on graphs with feedback vertex set number (and hence treewidth) at most w.*

3.3 Grundy Coloring on Special Graph Classes

For each fixed k, GRUNDY COLORING can be solved in polynomial time [26] and thus GRUNDY COLORING parameterized by the number of colors is in XP.

However, it is unknown whether it is in FPT for this parameter. We will next show several positive results for H-minor-free, chordal and claw-free graphs. Note that Grundy Coloring is NP-complete on chordal graphs [23] and on claw-free graphs [25].

We first observe that the XP algorithm of [24] implies a pseudo-polynomial-time algorithm on apex-minor-free graphs (such as planar graphs).

Proposition 11. Grundy Coloring *is* $n^{O(\log^2 n)}$*-time solvable on apex-minor-free graphs.*

Proposition 12. Grundy Coloring *parameterized by the number of colors is in* FPT *for the class of graphs excluding a fixed graph* H *as a minor.*

Proof. Notice that G contains a k-witness H as an induced subgraph if and only if $\Gamma(G) \geqslant k$. We can check, for every k-witness H, whether the input graph G contains H as an induced subgraph. By Observation 1, it suffices to test only the minimal k-witnesses. The number of minimal k-witnesses is bounded by some function of k and H-Induced Subgraph Isomorphism is in FPT when parameterized by $|V(H)|$ on graphs excluding H as a minor [9]. Therefore, one can check if $\Gamma(G) \geqslant k$ by solving H-Induced Subgraph Isomorphism for all minimal k-witnesses H. □

Proposition 13. *Let* $\mathcal{C}$ *be a graph class for which every member* G *satisfies* $tw(G) \leqslant f(\Gamma(G))$ *for some function* f*. Then,* Grundy Coloring *parameterized by the number of colors is in* FPT *on* $\mathcal{C}$*. In particular,* Grundy Coloring *is in* FPT *on chordal graphs.*

Proof. Since Grundy Coloring is in FPT for parameter combination of the number of colors and the treewidth [24], the first claim is immediate. Moreover $\omega(G) \leqslant \Gamma(G)$, hence if $tw(G) \leqslant f(\omega(G))$ we have $tw(G) \leqslant f(\Gamma(G))$. For any chordal graph G, $tw(G) = \omega(G) - 1$ [5]. □

Proposition 14. Grundy Coloring *can be solved in time* $O\left(nk^{\Delta^{k+1}}\right) = n\Delta^{\Delta^{O(\Delta)}}$ *for graphs of maximum degree* Δ*.*

Proof. Observation 2 implies that one can enumerate every distance-k-neighbourhood of each vertex, test every k-coloring of this neighborhood, and check if it is a valid Grundy k-coloring. Every such neighborhood has size at most $\Delta^{k+1} \leqslant \Delta^{\Delta+3}$ since by Observation 4, $k \leqslant \Delta + 2$. There are at most k^x k-colorings of a set of x elements. □

Corollary 15. *Let* $\mathcal{C}$ *be a graph class for which every member* G *satisfies* $\Delta(G) \leqslant f(\Gamma(G))$ *for some function* f*. Then,* Grundy Coloring *parameterized by the number of colors is in* FPT *for graphs in* $\mathcal{C}$*. In particular, this holds for the class of claw-free graphs.*

Proof. Straightforward by Proposition 14. Moreover, let G be a claw-free graph, and consider a vertex v of degree $\Delta(G)$. Since G is claw-free, the subgraph induced by the neighbors of v has independence number at most 2, and hence $\Gamma(G) \geqslant \chi(G) \geqslant \chi(N(v)) \geqslant \frac{\Delta(G)}{2}$. □

4 Weak and Connected Grundy Coloring

Among the three versions of Grundy Coloring we consider in this paper, Weak Grundy Coloring is the least constrained while Connected Grundy Coloring appears to be the most constrained one. This intuition turns out to be true when it comes to their parameterized complexity. When parameterized by the number of colors, Weak Grundy Coloring is in FPT while Connected Grundy Coloring does not belong to XP.

We recall that Weak Grundy Coloring is NP-complete [10].

Theorem 16. Weak Grundy Coloring *parameterized by number of colors is in* FPT*.*

The FPT-algorithm is based on the idea of *color-coding* by Alon et al. [1]. The height of a minimal witness for $\Gamma' \geqslant k$ is bounded by a function of k. Since those vertices of the same color do not need to induce an independent set, a random coloring will identify a *colorful* minimal witness with a good probability.

We also remark that the approach used to prove Theorem 16 does not work for Grundy Coloring because there is no control on the fact that a color class is an independent set.

Minimal connected Grundy k-witnesses, contrary to minimal Grundy k-witnesses (Observation 2), have arbitrarily large order: for instance, the cycle C_n of order n ($n > 4$, n odd) has a Grundy 3-witness of order 4, but its unique *connected* Grundy 3-witness is of order n: the whole cycle.

Observe that $\Gamma_c(G) \leqslant 2$ if and only if G is bipartite. Hence, Connected Grundy Coloring is polynomial-time solvable for any $k \leqslant 3$. However, we will now show that this is not the case for larger values of k, contrary to Grundy Coloring (Corollary 3). Hence, the parameterized version of the problem does not belong to XP.

Theorem 17. Connected Grundy Coloring *is* NP*-hard even for $k = 7$.*

Proof. We give a reduction from 3-SAT 3-OCC, an NP-complete restriction of 3-SAT where each variable appears in at most three clauses [22], to Connected Grundy Coloring with $k = 7$. We first give the intuition of the reduction. The construction consists of a tree-like graph of constant order (resembling binomial tree T_6) whose root is adjacent to two vertices of a K_6 (this constitutes W) and contains three special vertices a_4, a_{21}, and a_{24} (which will have to be colored with colors 1, 3, and 2 respectively), a connected graph P_1 which encodes the variables and a path P_2 which encodes the clauses. One in every three vertices of P_2 is adjacent to a_4, a_{21} and a_{24}. To achieve color 7, we will need to color those vertices with color strictly greater than 3. This will be possible if and only if the assignment corresponding to the coloring of P_1 satisfies all the clauses.

We now formally describe the construction. Let $\phi = (X = \{x_1, \ldots, x_n\}, \mathcal{C} = \{C_1, \ldots, C_m\})$ be an instance of 3-SAT 3-OCC where no variable appears always as the same literal. $P_1 = (\{i_1, i_2, v\} \cup \{v_i, \overline{v_i} \mid i \in [n]\}, \{\{i_1, i_2\}, \{i_2, v\}\} \cup \{\{v, v_i\} \cup \{v, \overline{v_i}\} \cup \{v_i, \overline{v_i}\} \mid i \in [n]\})$ consists of n triangles sharing the vertex v.

$P_2 = (\{p_j \mid j \in [3m-1]\}, \{\{p_j, p_{j+1}\} \mid j \in [3m-2])$ consists of a path of length $3m-1$. For each $j \in [m]$ and $i \in [n]$, $c_j \overset{def}{=} p_{3j-1}$ is adjacent to v_i if x_i appears positively in C_j, and is adjacent to $\overline{v_i}$ if x_i appears negatively in C_j. For each $j \in [m]$, c_j is adjacent to a_4, a_{21}, and a_{24}.

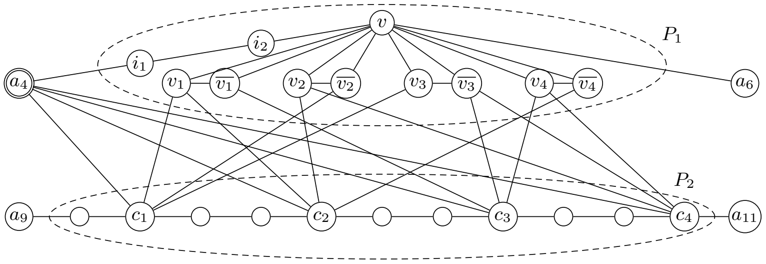


Fig. 2. P_1 and P_2 for the instance $\{x_1 \vee \neg x_2 \vee x_3\}, \{x_1 \vee x_2 \vee \neg x_4\}, \{\neg x_1 \vee x_3 \vee x_4\}, \{x_2 \vee \neg x_3 \vee x_4\}$

Intuitively, setting a literal to true consists of coloring the corresponding vertices with 3. Therefore, a clause C_j is satisfied if c_j has a 3 among its neighbors. To actually satisfy a clause, one has to color c_j with 4 or higher. Thus, c_j must also see a 2 in its neighborhood. We will show that the unique way of doing so is to color p_{3j-2} with 2, so all the clauses have to be checked along the path P_2.

We give, in Figure 3, a coloring of P_1 corresponding to a truth assignment of the instance SAT formula. One can check that when going along P_2 all the c_j's are colored with color 4.

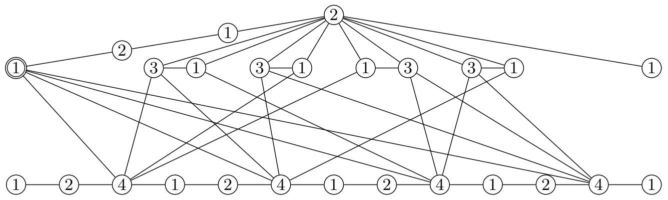


Fig. 3. A connected Grundy coloring such that all the c_j's are colored with color at least 4

The constant gadget W is depicted in Figure 4. The waves between a_4 and a_6 and between a_9 and a_{11} correspond, respectively, to the gadgets encoding the variables (P_1) and the clauses (P_2) described above and drawn in Figure 2. A connected Grundy coloring achieving color 7 is given in Figure 5 provided that going from a_9 to a_{11} can be done without coloring any vertex c_j with color 2 or less.

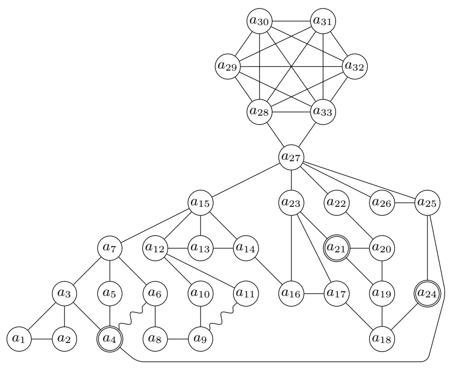


Fig. 4. The constant gadget. The doubly-circled vertices are adjacent to all the c_j's ($j \in [m]$).

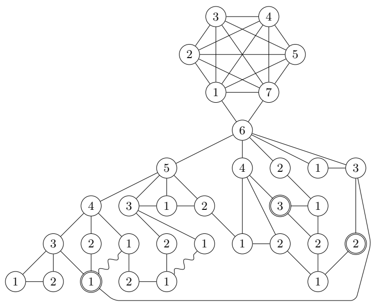


Fig. 5. A connected Grundy coloring of the constant gadget achieving color 7. The order is given by the sequence $(a_i)_{1\leqslant i\leqslant 33}$.

In the following claims, we use extensively Observation 1 which states that a vertex with degree d gets color at most $d+1$. We observe that coloring a vertex of degree d with color $d+1$ is useful only if we want to achieve color $d+1$. Indeed, otherwise, the vertex has all its neighbors already colored and cannot be used in the sequel. Moreover, if one wants to color a neighbor y of a vertex x in order to color x with a higher color, y cannot receive a color greater than its degree $d(y)$. Hence, the only vertices that could achieve color k are vertices of degree at least $k-1$ having at least one neighbor of degree at least $k-1$.

In the sequel, we call *doubly-circled vertices* the special vertices a_4, a_{21} and a_{24}, as they are doubly-circled in our figures.

Claim 17.A. *To achieve color* 7*,* a_{27} *needs to be colored with color* 6 *(while for all* $i \in [28, 33]$*,* a_i *is still uncolored).*

Claim 17.B. *Vertices* a_{26}*,* a_{22}*,* a_{25}*,* a_{23}*,* a_{15} *must receive color* 1*,* 2*,* 3*,* 4*,* 5 *respectively.*

Claim 17.C. *Vertex* a_7 *must receive color* 4*.*

Claim 17.D. *Vertex* a_3 *must receive color* 3*.*

Claim 17.D has further consequences: we must start the connected Grundy coloring by giving colors 1 and 2 to a_1 and a_2. The only follow-up, for connectivity reasons, is then to color a_3 with color 3 and a_4 with color 1. Thus, vertices a_5 and a_6 has to be colored with colors 2 and 1 respectively (so that a_7 can be colored 4). As, by Claim 17.B, a_{25} must receive color 3, a_{24} must receive color 2 (since a_4 has already color 1), so a_{18} must be colored 1.

Claim 17.E. *Vertex* a_{21} *must receive color* 3*.*

Claim 17.F. *The unique way of coloring a_{11} with color 1 without coloring any vertex c_j with color 1, 2, or 3 is to color all the c_j's for each $j \in [m]$.*

We remark that opposite literals are adjacent, so for each $i \in [n]$, only one of v_i and $\overline{v_i}$ can be colored with color 3. We interpret coloring v_i with 3 as setting x_i to true and coloring $\overline{v_i}$ with 3 as setting x_i to false.

Claim 17.G. *To color each c_j ($j \in [m]$) of the path P_2 with a color at least 4, the SAT formula must be satisfiable.*

So, to achieve color 7 in a connected Grundy coloring, the SAT formula must be satisfiable. The reverse direction consists of completing the coloring by giving a_{13} color 1 and a_{14} color 2, as shown in Figure 3 and Figure 5.

5 Concluding Remarks and Questions

We presented several positive and negative results concerning Grundy Coloring and two of its variants. To conclude this article, we suggest some questions which might be useful as a guide for further studies.

There is a gap between the $f(k, w) \cdot n$ (and XP) algorithm of [24] and the lower bound of Theorem 10. Is Grundy Coloring in FPT when parameterized by treewidth? Two simpler questions are whether there is a better $f(k, w)poly(n)$ algorithm (for example with $f(k, w) = k^{O(w)}$), and whether Grundy Coloring is in FPT when parameterized by the feedback vertex set number (it is easy to see that it is the case when parameterized by the vertex cover number).

Grundy Coloring (parameterized by the number of colors) is in XP, and we showed it to be in FPT on many important graph classes. Yet, the question whether it is in FPT or W[1]-hard remains unsolved. A perhaps more accessible research direction is to settle this question on bipartite graphs.

It would also be interesting to determine the (classic) complexity of Grundy Coloring on interval graphs. Also, we saw that the algorithm of [24] implies a pseudo-polynomial algorithm for planar (even apex-minor-free) graphs, making it unlikely to be NP-complete on this class. Is there a polynomial-time algorithm?

Concerning Connected Grundy Coloring, we showed that it becomes NP-complete for $k = 7$. As Connected Grundy Coloring is polynomial-time solvable for $k \leqslant 3$, its complexity status for $4 \leqslant k \leqslant 6$ and/or on restricted graph classes remains open.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM **42**(4), 844–856 (1995)
2. Araujo, J., Sales, C.L.: On the Grundy number of graphs with few P_4's. Discrete Applied Mathematics **160**(18), 2514–2522 (2012)
3. Benevides, F., Campos, V., Dourado, M., Griffiths, S., Morris, R., Sampaio, L., Silva, A.: Connected greedy colourings. In: Pardo, A., Viola, A. (eds.) LATIN 2014. LNCS, vol. 8392, pp. 433–441. Springer, Heidelberg (2014)

4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: Proc. 39th Annual ACM Symposium on Theory of Computing, STOC 2007, pp. 67–74. ACM (2007)
5. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica **11**(1–2), 1–21 (1993)
6. Bonnet, E., Foucaud, F., Kim, E.J., Sikora, F.: Complexity of grundy coloring and its variants (2014). CoRR, abs/1407.5336
7. Chang, G.J., Hsu, H.-C.: First-fit chromatic numbers of d-degenerate graphs. Discrete Mathematics **312**(12–13), 2088–2090 (2012)
8. Christen, C.A., Selkow, S.M.: Some perfect coloring properties of graphs. Journal of Combinatorial Theory, Series B **27**(1), 49–59 (1979)
9. Flum, J., Grohe, M.: Fixed-parameter tractability, definability, and model-checking. SIAM Journal on Computing **31**(1), 113–145 (2001)
10. Goyal, N., Vishwanathan, S.: NP-completeness of undirected Grundy numbering and related problems. Manuscript (1997)
11. Grundy, P.M.: Mathematics and games. Eureka **2**, 6–8 (1939)
12. Gyárfás, A., Lehel, J.: On-line and first fit colorings of graphs. Journal of Graph Theory **12**(2), 217–227 (1988)
13. Havet, F., Maia, A.K., Yu, M.-L.: Complexity of greedy edge-colouring. Research report RR-8171, INRIA, December 2012
14. Havet, F., Sampaio, L.: On the Grundy and b-chromatic numbers of a graph. Algorithmica **65**(4), 885–899 (2013)
15. Hedetniemi, S.M., Hedetniemi, S.T., Beyer, T.: A linear algorithm for the Grundy (coloring) number of a tree. Congressus Numerantium **36**, 351–363 (1982)
16. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences **63**(4), 512–530 (2001)
17. Kierstead, H.A., Saoub, K.R.: First-fit coloring of bounded tolerance graphs. Discrete Applied Mathematics **159**(7), 605–611 (2011)
18. Kortsarz, G.: A lower bound for approximating Grundy numbering. Discrete Mathematics & Theoretical Computer Science **9**(1) (2007)
19. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, pp. 777–789. SIAM (2011)
20. Moon, J., Moser, L.: On cliques in graphs. Israel Journal of Mathematics **3**(1), 23–28 (1965)
21. Narayanaswamy, N.S., Babu, R.S.: A note on first-fit coloring of interval graphs. Order **25**(1), 49–53 (2008)
22. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
23. Sampaio, L.: Algorithmic aspects of graph colouring heuristics. PhD thesis, University of Nice-Sophia Antipolis, November 2012
24. Telle, J.A., Proskurowski, A.: Algorithms for vertex partitioning problems on partial k-trees. SIAM Journal on Discrete Mathematics **10**(4), 529–550 (1997)
25. Zaker, M.: The Grundy chromatic number of the complement of bipartite graphs. Australasian Journal of Combinatorics **31**, 325–329 (2005)
26. Zaker, M.: Results on the Grundy chromatic number of graphs. Discrete Mathematics **306**(23), 3166–3173 (2006)

On the Complexity of the Minimum Independent Set Partition Problem

T.-H. Hubert Chan[1], Charalampos Papamanthou[2], and Zhichao Zhao[1](✉)

[1] Department of Computer Science, The University of Hong Kong, Hong Kong, China
{hubert,zczhao}@cs.hku.hk

[2] Department of Electrical and Computer Engineering, UMIACS University of Maryland, College Park, USA
cpap@umd.edu

Abstract. We consider the Minimum Independent Set Partition Problem (MISP) and its dual (MISPDual). The input is a multi-set of N vectors from $\{0, 1\}^n$, where $U := \{1, \dots, n\}$ is the index set. In MISP, a threshold k is given and the goal is to partition U into a minimum number of subsets such that the projected vectors on each subset of indices have multiplicity at least k, where the multiplicity is the number of times a vector repeats in the (projected) multi-set. In MISPDual, a target number χ is given instead of k, and the goal is to partition U into χ subsets to maximize k such that each projected vector appears at least k times.

The problem is inspired from applications in private voting verification. Each of the N vectors corresponds to a voter's preference for n contests. The n contests are partitioned into χ subsets such that each voter receives a verifiable tracking number for each subset. For each subset of contests, each voter's tracking number together with the votes for that subset is released in some public bulletin, which can be verified by each voter. The multiplicity k of the vectors' projection onto each subset of indices ensures that the bulletin for each subset of contests satisfies the standard privacy notion of k-anonymity.

In this paper, we show strong inapproximability results for both problems. For MISP, we show the problem is hard to approximate to within a factor of $n^{1-\epsilon}$. For MISPDual, we show the problem is hard to approximate to within a factor of $N^{1-\epsilon}$. Here, ϵ can be any small constant. Note that factors n and N approximation are trivial for MISP and MISPDual respectively. Hence, our results imply that any polynomial-time algorithm can almost do no better than the trivial one.

1 Introduction

We study the Minimum Independent Set Partition problem (MISP) and its dual problem (MISPDual). This problem was raised by Wagner on cstheory.stackexchange [12] in the context of data privacy [6]. We first describe the problem and an application scenario.

This research is partially funded by a grant from Hong Kong RGC under the contract HKU719312E.

In MISP, a multi-set Y of N vectors in $\{0,1\}^n$ is given together with a multiplicity threshold k. Our goal is to partition the indices $[n]$ into minimum number χ of subsets such that the projection of Y on each subset has multiplicity at least k.

The dual problem MISPDual is also of interest, in which a multi-set Y of vectors is also given. However, the target number χ of parts is given, and the goal is to return a χ-partition of the indices $[n]$ such that the minimum multiplicity k of the projected vectors is maximized.

Application Scenario. The problem is motivated by privacy in voting verification. We have N voters, each of whom is voting for n contests (with $\{0,1\}$ voting). To verify that all votes have been counted, each voter gets assigned a verifiable tracking number during voting. Then, there is a public bulletin board where all pairs of tracking numbers and votes are posted (where names of voters are withheld) such that each voter can verify that his votes are correct using his own tracking number. This could provide verifiability, but it is well-known in the privacy community that simply replacing a user's name with a random id cannot achieve privacy [6], since a voter might be uniquely identified by the way he votes in the n contests.

An expensive solution would be for each voter to get a separate tracking number for each contest, but this would increase the space complexity to store n numbers for each voter. Observe that if k is the minimum of the number of minority votes over all n contests, this expensive solution achieves the standard notion of k-anonymity [11].

To obtain a tradeoff between the space complexity of each voter and the anonymity parameter, one solution is: after receiving all votes, partition the n contests into some small number χ of subsets such that within each subset of contests, each voter has at least $k-1$ other voters who vote in exactly the same way in that subset of contests, for some parameter k. In the public bulletin board, the χ subsets of contests are released independently. Each voter needs to store only χ tracking numbers (one for each subset of contests), and k-anonymity is achieved.

The case for MISP corresponds to the scenario when a parameter k is given, and the goal is the minimize the number χ of subsets to achieve k-anonymity. For the dual problem MISPDual, the number χ of subsets is given, and the goal is to partition the contests into χ subsets such that the anonymity parameter k is maximized. Hence, it is of interest to investigate the complexity and hardness of approximation for these problems.

Our Results and Techniques. We prove strong inapproximability results for both problems MISP and MISPDual. We first give a reduction from graph coloring, which is NP-hard; in graph coloring, each vertex is assigned a color such that no two adjacent vertices receive the same color. Intuitively, each index in $[n]$ stands for a vertex, while the vectors capture the properties of the graph coloring problem. In our construction, a valid coloring corresponds to a partition with multiplicity k, while an invalid coloring corresponds to one with multiplicity 1.

The inapproximability of graph coloring implies that the approximation hardness of MISP and MISPDual with $\chi \geq 3$ is at least $n^{1-\epsilon}$ and $N^{1-\epsilon}$ respectively.

However, we show that MISPDual with $\chi = 2$ is much harder than graph coloring with $\chi = 2$. Observe that deciding if a graph is 2-colorable can be solved in polynomial time. Hence, to show the hardness of MISPDual with $\chi = 2$, we need new reduction techniques. We give a novel reduction from the NP-hard problem 3-SAT. Similar to graph coloring, some indices stand for variables and their negations. Intuitively, one subset stands for "true" and the other stands for "false". In order to show approximation hardness, for any threshold k, our reduction is carefully constructed such that a satisfiable assignment corresponds to a partition with multiplicity at least k, while an unsatisfiable formula corresponds to an instance such that any 2-partition has multiplicity only 1. This gap property allows us to prove that it is NP-hard to approximate MISPDual within factor $N^{1-\epsilon}$.

Our strong inapproximability results imply that there can be no efficient approximation algorithms for the problems MISP and MISPDual in their most general form. However, in real-world applications, the instances might have special structures that facilitate useful heuristic algorithms, which we leave as future research directions.

1.1 Historical Overview on Inapproximability

NP-Completeness has been developed in the 1970s [2,9]. Its success motivated the study of approximation algorithms. The first such paper was by Johnson [8]. He considered the problems Max SAT, Independent Set, Coloring and Set Cover. Several approximation algorithms have been proposed for these problems in this paper.

The design and analysis of approximation algorithms have grown since then. Several problems are shown to admit polynomial time approximation schemes (PTAS), meaning that they can be approximated as close to the optimum as possible. It was known from the very beginning of approximation algorithms that some problems do not admit PTAS. For instance, coloring can not be approximated within $\frac{4}{3} - \epsilon$, since 3-coloring is NP-hard. However, the inapproximabilities for many hard problems remains unknown.

Modern theory of inapproximability starts from the development of PCP systems, which are proved in [1]. Unlike conventional NP-hardness reduction, PCP systems can be used more readily to achieve inapproximability hardness. Based on PCP systems, several strong inapproximability have been proved since then, e.g., MAX 3SAT [7], Set Cover [5] and Coloring [4,13]. In particular, one of our reductions is based on the hardness of graph coloring [13].

Other Vector Partition Problems. Onn and Schulman [10] have also considered vector partition problems in which the input is also a collection of vectors. However, the goal is to partition the vectors (as opposed the coordinate index set) to maximize some convex objective function on the sum of vectors in each part. They showed that if both the dimension and the number of parts are fixed, the problem can be solved in strongly polynomial time.

2 Problem Definition

We give the formal definition of the Minimum Independent Set Partition Problem (MISP). The input is a positive integer n, a multi-set $Y := \{y_1, y_2, \ldots, y_N\}$ of N vectors in $\{0,1\}^n$, and a multiplicity threshold k. We use $U := [n] = \{1, 2, \ldots, n\}$ to denote the set of indices.

Given a vector y and subset $I \subseteq U$ of indices, we use $y|_X$ to denote the projection of vector y on I. For instance, for $y = (0,1,0,1,0,1)$ and $I = \{2,4,5\}$, $y|_I = (1,1,0)$. Given a multi-set Y, the projection of Y on I is a multi-set defined similarly $Y|_I := \{y|_I : y \in Y\}$.

A subset $I \subseteq U$ of indices is *k-independent* (with respect to Y) if each vector in the multi-set $Y|_I$ has multiplicity at least k, where multiplicity denotes the number of times a vector in $Y|_I$ repeats. A partition $\{I_1, I_2, \ldots, I_\chi\}$ of U is k-independent if each part I_i is k-independent.

The goal is to find the smallest integer χ and partition U into χ subsets $I_1, \ldots, I_\chi$ such that each partition I_i is k-independent.

Dual Problem. We also describe a dual version of the problem that we call MISPDual. Similarly, a multi-set Y of vectors are given, and a target number χ of partitions is given instead of k. The goal is to maximize k and partition the indices U into χ subsets $I_1, \ldots, I_\chi$ such that each I_i is k-independent.

3 General Reduction Schema

In this section, we reduce from the problem of graph coloring to MISP (and MISPDual with $\chi \geq 3$); in a *valid* coloring of an undirected graph, each vertex is assigned a color such that no two adjacent vertices receive the same color. We convert from an undirected graph $G = (V, E)$ to a multi-set of vectors such that a valid coloring corresponds to satisfying some fixed multiplicity threshold k while an invalid coloring leads to multiplicity 1. The use of this "k vs 1"-gap will be clear in the proof of the hardness of MISPDual. Because graph coloring is hard to approximate [13], our reduction readily implies the approximation hardness of MISP (and MISPDual with $\chi \geq 3$).

Our reduction depends on an arbitrarily chosen parameter $k > 1$ that is the same as the given threshold in MISP or may depend on the graph size $n = |V|$ in MISPDual. The index set is $U := [n]$. The multi-set Y consists of $N = k(n+1) + \binom{n}{2} + (k-1)|\overline{E}|$ vectors in $\{0,1\}^n$, where $\overline{E}$ is the edges in the complement graph of G. We also use $u \in V$ to denote an index of a vector. Let MISP(G, k) be the instance reduced from graph G with parameter k. The vectors in MISP(G, k) are defined as follows.

(I) An all-0's vector, and the n vectors in the standard basis (each having exactly one non-zero coordinate). Each of these vectors are repeated k times. There are $k(n+1)$ such vectors.

(II) Vectors of exactly two non-zero coordinates. There are $\binom{n}{2}$ such vectors.

(III) For each $(u, v) \notin E$, the vector with exactly two non-zero entries at indices u and v. Each of these vectors are repeated $(k-1)$ times. There are $(k-1)|\overline{E}|$ such vectors.

Figure 1 contains an example of the vectors for graph $G = (V = \{a, b, c, d\}, E = \{\{a, b\}, \{a, c\}, \{a, d\}\})$ and $k = 3$. Observe that parts (I) and (II) only depend on the size of graph G and k.

Note that a coloring of the graph gives a partition on U (and vice versa) in a natural way, where vertices having the same color corresponds to a subset of indices. Next we prove the relationship between colorings and partitions.

Theorem 1. *For any $k > 1$ and graph G, G has a valid χ-coloring iff* $\mathsf{MISP}(G, k)$ *has a k-independent χ-partition. If G does not have any valid χ-coloring, then any χ-partition of* $\mathsf{MISP}(G, k)$ *is not 2-independent.*

Proof. When G has a valid χ-coloring, we can induce a χ-partition from the coloring. We prove it is k-independent. Given a subset I of indices, consider the projected vectors in each part of the reduction.

Each projected vector in part (I) appears at least k times by the construction. Each projected vector in (III) appears at least k times since it repeats $k-1$ times in (III) and we can find a same one in (II).

For a vector in part (II), it depends on the indices u and v at which the entries are non-zero. If at most one of them is included in I, then the projected vector already appears k times in (I); otherwise, both u and v are included in I.

Two vertices u and v can be included in the same part I only if they are not neighbors in G; hence, the projected vector appears once from (II) and $k - 1$ times from (III). This proves the "only if" part.

On the other hand, if a χ-partition is 2-independent, then we induce a χ-coloring for G from the partition. We claim the coloring is valid. For any vertices u, v with the same color, we have a vector in (II) with u, v in the same part I (the part corresponding to their color). Such vector appears only once in (II). It appears in (III) at least once, since the partition is 2-independent. Hence u, v cannot be neighbours in G, thus it is a valid coloring. Notice k-independent implies 2-independent. This proves the "if" part and the contrapositive proves the second statement.

Theorem 2. *The inapproximability of* MISP *is $n^{1-\epsilon}$ for arbitrarily small $\epsilon > 0$, unless* $\mathsf{P} = \mathsf{NP}$*; this means that if a k-independent partition has minimum number of parts χ, it is* NP*-hard to return a k-independent partition with at most $n^{1-\epsilon} \cdot \chi$ parts. Moreover, the result holds for any constant $k \geq 2$.*

Proof. We want to show a reduction from a coloring instance G to an instance of MISP. Use the "reduction schema" in Theorem 1 with $k \geq 2$ to get a multi-set Y, which is a MISP instance with threshold k.

It is immediate from Theorem 1 that the minimum χ such that there is a k-independent partition with χ parts in the MISP instance is the same as the chromatic number of G (the minimum number of colors needed to color G).

Thus, the inapproximability of graph coloring can be applied to MISP. The inapproximability of chromatic number is $n^{1-\epsilon}$, by [13], meaning that it is NP-hard to approximate chromatic number within a factor of $n^{1-\epsilon}$. Hence, it is also NP-hard to approximate MISP within a factor of $n^{1-\epsilon}$.

4 Approximation Hardness of MISPDual

In this section, we show that the dual problem of maximizing the multiplicity of the projections into partitions with $\chi \geq 3$ is hard to approximate. In Section 5, we show that even for $\chi = 2$, the problem is hard.

Theorem 3. *For arbitrarily small constant $\epsilon > 0$, there is no polynomial time algorithm that approximates* MISPDual *within a factor of $N^{1-\epsilon}$, where N is the number of vectors in the given multi-set Y; moreover this result holds for any constant $\chi \geq 3$, unless* P=NP*.*

Remark 1. We comment on choosing "n vs N" as the parameter to express approximation hardness. In MISP, a trivial solution is to partition U into n singletons, and hence, it is natural to compare with the trivial solution with approximation ratio n. Hence, inapproximability within factor $n^{1-\epsilon}$ is a strong indicator that no efficient algorithm would exist.

In MISPDual, since any partition would give multiplicity 1, and the maximum possible multiplicity is the number N of vectors, inapproximability within factor $N^{1-\epsilon}$ indicates that there is no efficient algorithm. Observe that we can also derive n^C hardness for MISPDual for any constant C.

Proof. We use the fact [3] that the problem of deciding whether a graph is χ-colorable is NP-complete for any $\chi \geq 3$. We reduce the problem of deciding whether a graph G is χ-colorable to MISPDual, such that for a "YES" instance, the multiplicity of MISPDual solution is at least k, otherwise the multiplicity is at most 1. Later we will set $k = n^C$ for some large enough constant $C = \Omega(\frac{1}{\epsilon})$.

Given a graph G, we use the "reduction schema" in Theorem 1 with $k = n^C$ to get a multi-set Y, which is a MISPDual instance with the same χ (target number of parts). Suppose the graph is χ-colorable. From Theorem 1, we know that the MISPDual has a solution with multiplicity at least k. On the other hand, if the graph is not χ-colorable, then MISPDual only has solutions with multiplicity 1, since otherwise it will contradict Theorem 1.

Note that the gap between "NO" and "YES" instances is 1 vs k.

We next prove no polynomial algorithm can approximate MISPDual within a factor better (smaller) than $k = n^C$. Note that the size N of Y is at most $k(n+1)+\binom{n}{2}+(k-1)|\overline{E}| \leq n^{C+10}$, hence this will imply no polynomial algorithm can approximate MISPDual within a factor better than $N^{\frac{C}{C+10}}$.

Suppose there is an algorithm $\mathcal{A}$ that can approximate MISPDual within a factor better than k. Then, we can decide whether a graph is χ-colorable by examining if the multiplicity is greater than 1. Hence, it is NP-hard to approximate MISPDual within a factor better than $k = n^C > N^{\frac{C}{C+10}}$. Note that for constant C, this is a polynomial-time reduction.

Setting C large enough such that $\frac{C}{C+10} > 1 - \epsilon$ gives the result.

5 Improved Approximation Hardness of MISPDual

This is the most technical part of the paper. In view of Section 4, it is natural to ask whether MISPDual with $\chi = 2$ is polynomial-time solvable, as deciding if a graph is 2-colorable has an easy solution.

In this section, we answer this question negatively. We show strong inapproximability result for MISPDual with $\chi = 2$. Observe that the reduction from graph coloring no longer works. To derive such a result, we need some problem with binary choice to tackle 2-partition. It turns out that 3-SAT does the job. In our construction the two parts correspond to "true" and "false" literals. At the same time, "true" and "false" literals are distinguishable via additional indices. The inapproximability comes from the fact that any satisfiable assignment corresponds to a 2-partition with high multiplicity, while any non-satisfiable assignment corresponds to a 2-partition with low multiplicity. In particular, we prove the following result.

Theorem 4. *For arbitrarily small constant $\epsilon > 0$, there is no polynomial algorithm that approximates* MISPDual *with $\chi = 2$ within a factor of $N^{1-\epsilon}$, unless* P=NP.

Proof. We use the fact that 3-SAT is NP-hard [9]. We construct a reduction from 3-SAT to MISPDual with $\chi = 2$. Consider an instance of 3-SAT: $C = \wedge_{i=1}^{l} C_i = \wedge_{i=1}^{l}(c_{i,1} \vee c_{i,2} \vee c_{i,3})$, with l clauses and p distinct variables.

Here $c_{i,j}$ can be x or $\neg x$. Without loss of generality, we assume that x and $\neg x$ do not appear in the same clause. It is obvious that $p \leq 3l$, and we further assume that $p, l \geq 2$ to avoid trivial cases. The property of our reduction is that a satisfiable 3-SAT instance corresponds a MISPDual solution with multiplicity at least k (later fixed to be $l^{\Omega(\frac{1}{\epsilon})}$), while a non-satisfiable 3-SAT corresponds to a MISPDual solution with multiplicity at most 1. Notice that the gap "1 vs k" is used to derive the inapproximability result.

We next give the construction for the reduction from 3-SAT to MISPDual with $\chi = 2$. We need a parameter $k \geq 2$ to be fixed later, which will be polynomially related to l. We denote the resulting MISPDual instance by MISPDual(C, k), where C is the 3-SAT instance and k is the parameter.

Our reduction will generate a multi-set Y of vectors from $\{0, 1\}^{(1+l+2p)}$, with index set $U := [l + 1 + 2p]$. The first l indices are identification indices and are denoted by $[1..l]$. The $(l+1)$-th index is the separation index and is denoted by $(l + 1)$. The last $2p$ indices correspond to literals (and their negations) and are denoted by the literals, e.g., x or $\neg x$. The use of identification and separation indices will become clear in the proof.

NOTATION. To simplify description, coordinates not mentioned are 0.

There are four parts of vectors as below:

(I) There are $2k$ vectors:
The 1st vector is the vector with the first l coordinates being 1.
The 2nd to the k-th vectors are the vectors with the first $l+1$ coordinates being 1.
The $(k+1)$-st vector is the vector with the $(l+1)$-st coordinate being 1.
The remaining $k-1$ vectors are all zero vectors.
The use of (I) is to force the identification indices $1..l$ to be in different part from the separation index $l+1$ in a "good" partition. Notice that some $(0,1)$ will appear only once otherwise.

(II) There are $(2k+1)p$ vectors. For each variable x, we have $(2k+1)$ vectors described as below:

(II.x) The first k vectors are the vectors with coordinates $(x, \neg x)$ being $(0,1)$.
The next k vectors are the vectors with coordinates $(x, \neg x)$ being $(1,0)$.
The last vector is a vector with indices $(x, \neg x)$ being $(1,1)$.

The use of (II) is to force x and $\neg x$ to be apart. Since there will be only one $(1,1)$ if the two indices are put together. In a "good" partition, literals setting to be "true" are supposed to be within the identification indices' (the first l indices) partition, while the "false" are in the separation index's (the $l+1$-st index) partition.

(III) There are $(3k+1)l$ vectors. For each clause $C_i = x \vee y \vee z$ (with literals x, y and z), we have $3k+1$ vectors:

(III.i) The first k vectors are the vectors with the i-th coordinate set to 1 and coordinates $(\neg y, \neg z)$ set to 1.
The next k vectors are the vectors with the i-th coordinate set to 1 and the coordinates $(\neg x, \neg z)$ set to 1.
The next k vectors are the vectors with the i-th coordinate set to 1 and the coordinates $(\neg x, \neg y)$ set to 1.
The last vector is the vector with the i-th coordinate set to 1 and the coordinates $(\neg x, \neg y, \neg z)$ set to 1.
Note that for all the $(3k+1)$ vectors, the i-th coordinate is set to 1.

The use of (III) is to force the variables to satisfy the constraints. Notice that if a clause is not satisfied, then all the indices $\neg x, \neg y, \neg z$ are on the "true" side (together with the first l indices), causing $(1,1,1)$ to appear only once in the projection onto the coordinates $(\neg x, \neg y, \neg z)$. On the other hand, as long as the not all indices $\neg x, \neg y, \neg z$ are included on the "true" side, any vector will appear at least k times. Notice the use of identification indices (the first l indices) here. With different identification indices, clauses will not affect each other.

(IV) There are lk vectors. For each clause $C_i = x \vee y \vee z$ there are k vectors as follows.

(IV.i) The k vectors are the same with the coordinates $(\neg x, \neg y, \neg z)$ set to 1.

Notice that in (IV) the identifier columns are set to 0, which is different from (III). The idea is to handle the situation when in (III.i) all $\neg x, \neg y, \neg z$ are partitioned into the "false" side. If this happens, the vector (projected on the "false" side with the $(l+1)$-st index) will repeat at least k times. Figure 2 (in appendix) gives an example for $(x \vee y \vee z) \wedge (\neg y \vee \neg x \vee w)$ with $k = 2$.

It remains to show that a satisfiable assignment corresponds to an k-independent partition, while a non-satisfiable assignment corresponds to a MISPDual instance such that any 2-partition is not 2-independent.

Lemma 1. *For all $k > 1$ and 3-SAT instance C, if C has a satisfiable assignment, then* $\mathsf{MISPDual}(C, k)$ *has a k-independent 2-partition.*

Proof. Give a satisfiable assignment, we partition the indices set U into 2 subsets T and F as follows. The first l indices $[1..l]$ are included in T, and the $(l+1)$-st index is in F. For each literal x, if $x = \mathit{true}$ then the index x is included in T and the index $\neg x$ is included in F; otherwise, the index $\neg x$ is in T and the index x is in F.

We next consider the vectors in each part projected on T and F.

Claim. In (I), each vector appears at least k times on both T and F.

Proof. First we consider the each vector in (I) projected on T. By construction, in the first l coordinates, each of the all 1's and all 0's vectors is repeated k times, and other coordinates are all set to 0.

For the projections on F, only the $(l+1)$-st index has non-zero values and it contains exactly k 1's and k 0's. Hence, in (I), each projected vector repeats at least k times.

Claim. In (II.x), each vector appears at least k times on both T and F.

Proof. It can be seen that the only non-zero values are at indices x and $\neg x$. At both x and $\neg x$, we have more than k 0's and k 1's.

By the construction we know that x and $\neg x$ are assigned to different parts. In each part, the only non-zero coordinate is repeated at least k times, for each of the two values 0 and 1.

Claim. In (III.i), each vector appears at least k times on both T and F.

Proof. We denote the i-th clause by $C_i = x \vee y \vee z$, where x, y, z can be a variable or its negation. By construction, at least 1 of $\neg x, \neg y, \neg z$ is in F, since it is a satisfiable assignment. For instance, suppose $\neg z$ is in F; other situations follow the same argument.

Consider projections on F. Since the first l indices are not in F, we can find at least k same vectors in (III.i) and (IV.i) (in case $\neg x, \neg y, \neg z \in F$).

Now consider the projections on T. Vectors in (III.i) projected on T only differ at indices $\neg x$ and $\neg y$. It can be seen from the construction that no matter which part each of the indices $\neg x$ and $\neg y$ goes, each projected vector still appears at least k times. Hence, result of Claim 5 follows.

Claim. In (IV), each vector appears at least k times on both T and F.

Proof. This follows immediately from the construction.

The result of Lemma 1 follows, since each projected vector repeats at least k times.

Lemma 2. *For all $k > 1$ and 3-SAT instance C, if* $\mathsf{MISPDual}(C, k)$ *has a* 2*-independent* 2*-partition, then C has a satisfiable assignment.*

Proof. We first argue that if the 2-partition is 2-independent, then the identification (first l) indices and the separation ($l + 1$-st) index should be in different subsets. Similarly, x and $\neg x$ should be in different subsets. Then, an assignment is derived (such that literals on the same side as the identification indices are set to true) and analyzed.

Claim. Each of the indices $[1..l]$ is in the subset different from the subset containing the index $l + 1$.

Proof. Note that the only 1's at index $l+1$ happens in vectors 2 to $k+1$. Suppose on the contrary that some index in $j \in [1..l]$ is in the same subset at index $l+1$. Then, at the coordinates $(j, l + 1)$, the projection $(0, 1)$ will appear only once due to vector $k + 1$. This contradicts 2-independence.

We denote T as the subset containing $[1..l]$, and F as the other subset F.

Claim. For each literal x, x and $\neg x$ are in different subsets.

Proof. Notice that we assume that no x and $\neg x$ appear in the same clause. As a result, there will be no vector with coordinates $(x, \neg x)$ being $(1, 1)$ in (I,III,IV). Such a vector appears only once in (II). The result follows as the partition is 2-independent.

From this point it is obvious that we should assign *true* to the literals in T and $false$ to the literals in F. Next we prove that this is indeed a satisfying assignment.

Claim. Every clause C_i is satisfied by the above assignment.

Proof. Suppose $C_i = x \vee y \vee z$ is not satisfied. Then, it must be the case that $\neg x, \neg y, \neg z \in T$. We consider the vectors in (III.i) projected on T. From the construction, in (III.i) there will be exactly one vector with coordinates $(\neg x, \neg y, \neg z)$ being $(1, 1, 1)$.

We argue that this vector projected on T does not appear anywhere else. To see this, note that the identification indices are included in T, which is different from all other parts except (III.i). In (III.i), such vector (projected on T) only appears once, and hence the result follows.

This completes the proof of Lemma 2.

The following corollary is the contrapositive of Lemma 2.

Corollary 1. *For all $k > 1$ and 3-SAT instance C, if C does not have any satisfiable assignment, then any* 2*-partition for* $\mathsf{MISPDual}(C, k)$ *is not* 2*-independent.*

At this point, we can see that there is a gap of 1 vs k, meaning that to distinguish satisfiable 3-SAT from unsatisfiable ones, we only need to distinguish between multiplicity 1 and k. Hence, any polynomial algorithm that approximates MISPDual within a factor better than k will imply P=NP.

We can set $k = l^C$ for some large enough constant C, and observing that $N \leq l^{C+10}$, we conclude that there is no polynomial algorithm with approximation ratio better than $N^{\frac{C}{C+10}}$.

Choosing C large enough (depending on ϵ) completes the proof of Theorem 4.

Appendix

	a	b	c	d
(I)	0	0	0	0
	0	0	0	0
	0	0	0	0
	1	0	0	0
	1	0	0	0
	1	0	0	0
	0	1	0	0
	0	1	0	0
	0	1	0	0
	0	0	1	0
	0	0	1	0
	0	0	1	0
	0	0	0	1
	0	0	0	1
	0	0	0	1
(II)	1	1	0	0
	1	0	1	0
	1	0	0	1
	0	1	1	0
	0	1	0	1
	0	0	1	1
(III). (b, c)	0	1	1	0
	0	1	1	0
(III). (b, d)	0	1	0	1
	0	1	0	1
(III). (c, d)	0	0	1	1
	0	0	1	1

Fig. 1. $G = (V = \{a, b, c, d\}, E = \{\{a, b\}, \{a, c\}, \{a, d\}\})$ with $k = 3$

	Identifiers		Separator	x	¬ x	y	¬ y	z	¬ z	w	¬ w
(I)	1	1	0								
	1	1	1								
	0	0	1								
	0	0	0								
(II. x)				0	1						
				0	1						
				1	0						
				1	0						
				1	1						
(II. y)						0	1				
						0	1				
						1	0				
						1	0				
						1	1				
(II. z)								0	1		
								0	1		
								1	0		
								1	0		
								1	1		
(II. w)										0	1
										0	1
										1	0
										1	0
										1	1
(III. 1)	1	0			0		1		1		
	1	0			0		1		1		
	1	0			1		0		1		
	1	0			1		0		1		
	1	0			1		1		0		
	1	0			1		1		0		
	1	0			1		1		1		
(III. 2)	0	1		1		0					1
	0	1		1		0					1
	0	1		0		1					1
	0	1		0		1					1
	0	1		1		1					0
	0	1		1		1					0
	0	1		1		1					1
(IV. 1)	0	0			1		1		1		
	0	0			1		1		1		
(IV. 2)	0	0		1		1					1
	0	0		1		1					1

Fig. 2. $(x \vee y \vee z) \wedge (\neg y \vee \neg x \vee w)$ with $k = 2$; unspecified entries are 0

Acknowledgments. We would like to thank David Wagner for posting the problem online [12] and for useful discussions.

References

1. Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of np. J. ACM **45**(1), 70–122 (1998)
2. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC 1971, pp. 151–158. ACM, New York (1971)
3. Dailey, D.P.: Uniqueness of colorability and colorability of planar 4-regular graphs are np-complete. Discrete Mathematics **30**(3), 289–293 (1980)
4. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. In: Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity, pp. 278–287 (1996)
5. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652 (1998)
6. Ganta, S.R., Kasiviswanathan, S.P., Smith, A.: Composition attacks and auxiliary information in data privacy. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, pp. 265–273. ACM, New York (2008)
7. Håstad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)
8. Johnson, D.S.: Approximation algorithms for combinatorial problems. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, STOC 1973, pp. 38–49. ACM, New York (1973)
9. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
10. Onn, S., Schulman, L.J.: The vector partition problem for convex objective functions. Math. Oper. Res. **26**(3), 583–590 (2001)
11. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10**(5), 557–570 (2002)
12. Wagner, D.: Find index set partition that has large projections (2013). http://cstheory.stackexchange.com/questions/17562/find-index-set-partition-that-has-large-projections
13. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing **3**(6), 103–128 (2007)

Bivariate Complexity Analysis of Almost Forest Deletion

Ashutosh Rai(✉) and Saket Saurabh

The Institute of Mathematical Sciences, Chennai, India
{ashutosh,saket}@imsc.res.in

Abstract. In this paper we study a generalization of classic Feedback Vertex Set problem in the realm of multivariate complexity analysis. We say that a graph F is an l-forest if we can delete at most l edges from F to get a forest. That is, F is at most l edges away from being a forest. In this paper we introduce the Almost Forest Deletion problem, where given a graph G and integers k and l, the question is whether there exists a subset of at most k vertices such that its deletion leaves us an l-forest. We show that this problem admits an algorithm with running time $2^{\mathcal{O}(l+k)}n^{\mathcal{O}(1)}$ and a kernel of size $\mathcal{O}(kl(k+l))$. We also show that the problem admits a $c^{\mathbf{tw}}n^{\mathcal{O}(1)}$ algorithm on bounded treewidth graphs, using which we design a subexponential algorithm for the problem on planar graphs.

1 Introduction

In the field of graph algorithms, vertex deletion problems constitute a considerable fraction. In these problems we need to delete a small number of vertices such that the resulting graph satisfies certain properties. Many well known problems like Vertex Cover and Feedback Vertex Set fall under this category. Most of these problems are NP-complete due to a classic result by Lewis and Yannakakis [19]. The field of parameterized complexity tries to provide efficient algorithms for these NP-complete problems by going from the classical view of single-variate measure of the running time to a multi-variate one. It aims at getting algorithms of running time $f(k)n^{\mathcal{O}(1)}$, where k is an integer measuring some aspect of the problem. The integer k is called the *parameter*. In most of the cases, the solution size is taken to be the parameter, which means that this approach gives faster algorithms when the solution is of small size. For more background, the reader is referred to the monographs [4,7,20].

Recently, there has been a trend of exploiting other structural properties of the graph other than the solution size [2,3,6,14,18]. For further reading, reader may refer to the recent survey by Fellows et al. [5]. In an earlier work, Guo et al. [13] introduced the notion of *"distance from triviality"* which looked at structural parameterization as a natural way to deal with problems which are polynomial time solvable on some graph classes. They argued that we could ask the same problems on some other (bigger) graph class which is close to the graph class on which the problem is polynomial time solvable, but the parameter is the

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 133–144, 2015.
DOI: 10.1007/978-3-319-21398-9_11

closeness or the *distance* from the original graph class instead of the solution size.

In the same spirit, we introduce the notion of *distance from tractability*. We know that vertex deletion problems deal with deletion of vertices to get to some graph class. For example, the Vertex Cover problem deals with deleting vertices such that the resulting graph does not have any edge. Similarly, the well known Feedback Vertex Set problem talks about deleting vertices such that the resulting graph is a forest. What if we want to delete vertices so that the resulting graph class is *close* to the earlier graph class, while taking the *measure of closeness* as a parameter? This approach takes us from single variate parameterized algorithms to multivariate ones, which throws some light on the interplay between the parameters concerned. More precisely, they tell us about the trade-offs if we want to go away from the tractable version of a problem, and hence the term "distance from tractability". We also point out that the term can be a bit misleading, since in the case of this paper, we do find out tractable (FPT) algorithms for these problems when we consider both the parameters. But for want of a better term, we use it here.

There is already some work done which can be seen as examples of the notion of parameterizing by distance from tractability. For example, the Partial Vertex Cover and other related partial cover problems [1,10] come to mind, where after the deletion of a small set of vertices, the resulting graph is close to an edgeless graph. In these problems, the measure of closeness is the number of edges. Similarly, work has been done on vertex deletion to get a graph of certain treewidth, which can be looked as deletion of vertices to get a graph close to a forest, where the measure of the closeness is the treewidth of the graph.

The algorithms of the above mentioned kind show the correlation between the solution size and the distance from tractability. Let the distance from tractability be the parameter ℓ, and k be the number of vertices to be deleted. Suppose we have an algorithm with running time say $f(\ell)^{g(k)}n^{\mathcal{O}(1)}$, then is it possible to obtain an algorithm with running time $h_1(\ell)h_2(k)n^{\mathcal{O}(1)}$? That is, could we disentangle the function depending on both ℓ and k to a product of functions where each function depends only on one of ℓ and k. Answer to this question is *yes* if we take $f(\ell)^{g(\ell)}f(k)^{g(k)}n^{\mathcal{O}(1)}$. However, if we ask for an algorithm with running time $h(\ell)2^{\mathcal{O}(g(k))}n^{\mathcal{O}(1)}$ then it becomes interesting. This kind of question can be asked for several problems. For an example, it is known that the Treewidth-η-Deletion problem, where the objective is to test whether there exists a vertex subset of size at most k such that its deletion leaves a graph of treewidth at most η. For $\eta = 0$ and 1 this correspond to the Vertex Cover and Feedback Vertex Set problems, respectively. It is known that Treewidth-η-Deletion admits an algorithm with running time $f(\eta)^k n^{\mathcal{O}(1)}$ [9,15]. However, it is not known whether there exists an algorithm with running time $h(\eta)2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. Clearly, algorithms with running time $h(\ell)2^{\mathcal{O}(g(k))}n^{\mathcal{O}(1)}$ are more desirable.

The Feedback Vertex Set problem has been widely studied in the field of parameterized algorithms. A series of results have improved the running times to $\mathcal{O}^*(3.619^k)$ in deterministic setting [17] and $\mathcal{O}^*(3^k)$ in randomized setting [3], where the $\mathcal{O}^*$ notation hides the polynomial factors. Looking at this problem in the notion of distance from tractability, the number of edges comes to mind as a natural measure of distance. More precisely, we try to address the question of deleting vertices such that the resulting graph is l edges away from being a forest. We call such forests l-forests, and the problem ALMOST FOREST DELETION. The main focus of this paper is to design algorithm for this problem with parameters both l and k.

Our Results. We show that ALMOST FOREST DELETION can be solved in time $\mathcal{O}^*(5.0024^{(k+l)})$. We arrive at the result using the iterative compression technique which was introduced in [21] and a non-trivial measure which helps us in getting the desired running time. Then we explore the kernelization complexity of the problem, and show that ALMOST FOREST DELETION admits a polynomial kernel with $\mathcal{O}(kl(k+l))$ edges. For arriving at the result, we first make use of the Expansion Lemma and Gallai's theorem for reducing the maximum degree of the graph, and then we bound the size of the graph. It is easy to see that for a YES instances (G, k, l) of ALMOST FOREST DELETION, the treewidth of G is bounded by $k + l$. Since we have an algorithm of the form $\mathcal{O}^*(c^{(k+l)})$ on general graphs, the question of finding an $\mathcal{O}^*(c^{\mathbf{tw}})$ algorithm becomes interesting for bounded treewidth graphs. We answer this question affirmatively by giving an $\mathcal{O}^*(17^{\mathbf{tw}})$ algorithm for graphs which come with a tree decomposition of width $\mathbf{tw}$. This algorithm, along with the notion of bidimensionality gives rise to an algorithm for ALMOST FOREST DELETION on planar graphs running in time $2^{\mathcal{O}(\sqrt{l+k})}n^{\mathcal{O}(1)}$. Our methods are based on the known methods to solve the FEEDBACK VERTEX SET problem.

2 Preliminaries

For a graph G, we denote the set of vertices of the graph by $V(G)$ and the set of edges of the graph by $E(G)$. For a set $S \subseteq V(G)$, the *subgraph of* G *induced by* S is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{(u, v) \in E(G) : u, v \in S\}$, and the subgraph obtained after deleting S is denoted as $G - S$. If H is a subgraph of G, we write $H \subseteq G$ and for two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, by $G_1 \cup G_2$, we denote the graph $(V_1 \cup V_2, E_1 \cup E_2)$. All vertices adjacent to a vertex v are called neighbours of v and the set of all such vertices is called the neighbourhood of v. A *k-flower* in a graph is a set of k cycles which are vertex disjoint except for one vertex v, which is shared by all the cycles in the set. The vertex v is called *center* of the flower and the cycles are called the *petals* of the flower. A *forest* is a graph which does not contain any cycle. An l-forest is a graph which is at most l edges away from being a forest, i.e. the graph can be transformed into a forest by deleting at most l edges. For a connected component C of a graph, we call the quantity $|E(G[C])| - |C| + 1$ the *excess of* C and denote it by $\mathsf{ex}(C)$. It can also be equivalently defined as the minimum number of edges we need to delete from the connected component to get to a tree. For a graph G, let $\mathcal{C}$ be the set of its connected components. We define the excess of the graph, $\mathsf{ex}(G)$ as follows.

$$\mathsf{ex}(G) = \sum_{C \in \mathcal{C}} \mathsf{ex}(C)$$

As in the case of components, this measure can be equivalently defined as the minimum number of edges we need to delete from G to to get to a forest. It is easy to see that a graph G is an l-forest if and only if $\mathsf{ex}(G) \leq l$. For $X \subseteq V(G)$ such that $G - X$ is an l-forest, we call X an l-forest deletion set of G. We denote $\{1, \ldots, n\}$ by $[n]$. We define the Almost Forest Deletion problem as follows.

Almost Forest Deletion	
Input:	A graph G, integers k and l.
Parameter(s):	l, k
Question:	Does there exist $X \subseteq V(G)$ such that $\lvert X \rvert \leq k$ and $G - X$ is an l-forest?

Observation 1. *Let G' be a subgraph of G. If G is an l-forest, then so is G'.*

Observation 2. *If G is an l-forest, it has at most $V(G) - 1 + l$ edges.*

Lemma 3. *Let G be a graph. If there exists a vertex v such that v is not part of any cycle in G, then* $\mathsf{ex}(G - \{v\}) = \mathsf{ex}(G)$. *Furthermore, if v is part of a cycle in G, then* $\mathsf{ex}(G - \{v\}) \leq \mathsf{ex}(G) - 1$.

Lemma 4. *Let $X \subseteq V(G)$ be a set of vertices of G which do not belong to any cycle. Then, G is an l-forest if and only if $G - X$ is an l-forest.*

Lemma 5. *Any l-forest can have at most l edge disjoint cycles.*

Kernelization. A *kernelization* algorithm for a parameterized language L is a polynomial time procedure which takes as input an instance $(x, k_1, \ldots, k_l)$, where k_i's are the parameters and returns an instance $(x', k'_1, \ldots, k'_l)$ such that $(x, k_1, \ldots, k_l) \in L$ if and only if $(x', k'_1, \ldots, k'_l) \in L$ and $\lvert x' \rvert \leq h(k_1, \ldots, k_l)$ and $k'_i \leq g(k_1, \ldots, k_l)$ for all $i \in [l]$, for some computable functions h, g. The returned instance is said to be a *kernel* for L and the function h is said to be the *size of the kernel.*

Treewidth. Let G be a graph. A *tree-decomposition* of a graph G is a pair $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ such that

– $\cup_{t \in V(\mathbb{T})} X_t = V(G)$,
– for every edge $xy \in E(G)$ there is a $t \in V(\mathbb{T})$ such that $\{x, y\} \subseteq X_t$, and
– for every vertex $v \in V(G)$ the subgraph of $\mathbb{T}$ induced by the set $\{t \mid v \in X_t\}$ is connected.

The *width* of a tree decomposition is $\max_{t \in V(\mathbb{T})} |X_t| - 1$ and the *treewidth* of G is the minimum width over all tree decompositions of G and is denoted by $\mathbf{tw}(G)$.

A tree decomposition $(\mathbb{T}, \mathcal{X})$ is called a *nice tree decomposition* if $\mathbb{T}$ is a tree rooted at some node r where $X_r = \emptyset$, each node of $\mathbb{T}$ has at most two children, and each node is of one of the following kinds:

1. **Introduce node**: a node t that has only one child t' where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
2. **Forget node**: a node t that has only one child t' where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.
3. **Join node**: a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.
4. **Base node**: a node t that is a leaf of $\mathbb{T}$, is different than the root, and $X_t = \emptyset$.

Notice that, according to the above definition, the root r of $\mathbb{T}$ is either a forget node or a join node. It is well known that any tree decomposition of G can be transformed into a nice tree decomposition maintaining the same width in linear time [16]. We use G_t to denote the graph induced by the vertex set $\cup_{t'} X_{t'}$, where t' ranges over all descendants of t, including t. By $E(X_t)$ we denote the edges present in $G[X_t]$.

3 An $O^*(c^{(l+k)})$ Algorithm for ALMOST FOREST DELETION

In this section we will present a $c^{(l+k)} n^{\mathcal{O}(1)}$ algorithm for ALMOST FOREST DELETION. We use the well known technique of iterative compression and arrive at the desired running time after defining a non-trivial measure.

Given an instance (G, k, l) of ALMOST FOREST DELETION, let $V(G) = \{v_1, \ldots, v_n\}$ and define vertex sets $V_i = \{v_1, \ldots, v_i\}$, and let the graph $G_i = G[V_i]$. We iterate through the instances (G_i, k, l) starting from $i = k + 1$. For the i^{th} instance, we try to find an l-forest deletion set $\hat{S}_i$ of size at most k, with the help of a *known* l-forest deletion set S_i of size at most $k + 1$. Formally, the compression problem we address is the following.

ALMOST FOREST DELETION COMPRESSION	
Input:	A graph G, an l-forest deletion set S of G of size at most $k+1$, integers k and l.
Parameter(s):	k, l
Question:	Does there exist $X \subseteq V(G)$ such that $|X| \leq k$ and $G - X$ is an l-forest?

Lemma 6. *If* ALMOST FOREST DELETION COMPRESSION *can be solved in $f(k, l)n^c$ time, then* ALMOST FOREST DELETION *can be solved in $f(k, l)n^{c+1}$ time.*

For designing an algorithm for Almost Forest Deletion Compression, let the input instance be (G, S, k, l). We guess a subset $Y \subseteq S$ with the intention of picking these vertices in our hypothetical solution for this instance and not picking the rest of the vertices of S in the solution. We delete the set Y from the graph and decrease k by $|Y|$. We then check if the graph $G[S \setminus Y]$ is an l-forest and if it is not, then reject this guess of Y as a spurious guess. Suppose that $G[S \setminus Y]$ is indeed an l-forest. Then, it remains only to check if there is an l-forest deletion set S' of the size $k' = k - |Y|$ which is disjoint from $S \setminus Y$, and $G - (Y \cup S')$ is an l-forest. More precisely, we have an instance of AFDDC, which is defined as follows.

Almost Forest Deletion Disjoint Compression (AFDDC)	
Input:	A graph G, an l-forest deletion set S of G, integers k and l.
Parameter(s):	k, l
Question:	Does there exist $X \subseteq V(G)$ such that $X \cap S = \emptyset$, $\vert X\vert \leq k$ and $G - X$ is an l-forest?

To solve the problem, we first design a set of reduction rules.

Reduction Rule 1. *If there exists a vertex v of degree at most 1 in the graph, delete it.*

Reduction Rule 2. *If there exists $v \in V(G) \setminus S$ such that $G[S \cup \{v\}]$ is not an l-forest, delete v and decrease k by 1.*

Reduction Rule 3. *If there exists a vertex $v \in V(G) \setminus S$ of degree two, such that at least one of its neighbours is in $V(G) \setminus S$, then delete v and put a new edge between its neighbours (even if they were already adjacent). If both of v's edges are to the same vertex, delete v and put a new self loop on the adjacent vertex (even if it has self loop(s) already).*

It is easy to see that after the exhaustive applications of reduction rules 1-3, if there exists a vertex of degree at most 1 in $G - S$, then it has at least 2 neighbours in S.

Now we are ready to describe our algorithm for AFDDC. Given an input instance (G, S, k, l) of AFDDC, we first apply reduction rules 1, 2 and 3 exhaustively. If $k < 0$, then we return that the given instance is a No instance.

Now, we look for a vertex v of degree at most 1 in $G - S$ and we branch by either including v in our solution or excluding it. More precisely, we call the algorithm recursively on $(G - \{v\}, S, k - 1, l)$ and $(G, S \cup \{v\}, k, l)$. If one of the recursive call returns Yes, then we say that the instance was a Yes instance. If there does not exist a vertex of degree at most 1 in $G - S$, then there must be a vertex v which is part of a cycle. In this case we branch on this vertex, and call the algorithm recursively on $(G - \{v\}, S, k - 1, l)$ and $(G, S \cup \{v\}, k, l)$ as we did in the previous case. This concludes the description of the algorithm. The correctness of the algorithm follows from the correctness of reduction rules and the fact that the branching is exhaustive.

To analyze the running time of the algorithm, we define a measure $\phi(I)$ for the input instance $I = (G, S, k, l)$ as follows.

$$\phi(I) = \alpha k + \beta \mathsf{cc}(S) + \gamma(l - \mathsf{ex}(G[S])) + \delta(\mathsf{ex}(G - S))$$

Here, $\mathsf{cc}(S)$ denotes the number of connected components of $G[S]$ and α, β, γ, δ are positive constants such that $\delta > \beta$. We will assume these properties for now, and will fix the values of these constants later.

Lemma 7. *None of the reduction rules 1-3 increases the measure $\phi(I)$.*

Lemma 8. AFDDC *can be solved in time $\mathcal{O}^*((4.0024)^k(5.0018)^l)$.*

Proof. When be branch on a vertex v of degree at most 1 in $G - S$, the measure drops by α in the first branch. In the other branch, depending on whether v's neighbours in S belong to different components or to the same component, the measure drops by at least β or γ. This gives us branching factors of (α, β) and (α, γ). Branching on a vertex v, which is part of a cycle in $G - S$, gives us a branching factor of $(\alpha + \delta, \delta - \beta)$. We ran a numerical program to find values of α, β, γ and δ, which optimize the running time of the algorithm. Putting $\alpha = 1.45$, $\beta = 1.35$, $\gamma = 1.35$ and $\delta = 1.9$ gives us the worst case running time of $(4.0024)^k(5.0018)^l n^{\mathcal{O}(1)}$. □

Given Lemma 8, the algorithm for ALMOST FOREST DELETION COMPRESSION runs in time $\mathcal{O}(\sum_{i=0}^{k} \binom{k+1}{i} \cdot (4.0024)^i(5.0018)^l \cdot n^{\mathcal{O}(1)}) = \mathcal{O}^*(5.0024^{(k+l)})$. Here, the factor of $\binom{k+1}{i}$ is for the guesses we make for the set S. Finally applying Lemma 6, we get the following theorem.

Theorem 9. ALMOST FOREST DELETION *can be solved in $\mathcal{O}^*(5.0024^{(k+l)})$ time.*

4 $\mathcal{O}(kl(k+l))$ Kernel for ALMOST FOREST DELETION

In this section, we give the kernelization algorithm for ALMOST FOREST DELETION. First we give a set of reduction rules which help us bound the size of the output instance. Throughout the section, we apply the reduction rules in order, that is, while applying a reduction rule we assume that all the reduction rules stated previously in the section have been applied exhaustively.

Reduction Rule 4. *If there exists a vertex v of degree at most 1 in the graph, delete it.*

Reduction Rule 5. *If there exists a vertex $v \in V(G)$ of degree two then delete v and put a new edge between its neighbours (even if they were already adjacent). If both of v's edges are to the same vertex, delete v and put a new self loop on the adjacent vertex (even if it has self loop(s) already).*

Reduction Rule 6. *If any edge has multiplicity more that $l+2$, then delete all but $l+2$ copies of that edge.*

Given an instance (G, k, l) of Almost Forest Deletion, we apply reduction rules 4-6 exhaustively. Observe that after the application of these reduction rules, the graph has degree at least 3, as all the vertices of degrees 1 and 2 are taken care of by Reduction Rule 4 and Reduction Rule 5 respectively.

Lemma 10. *If a graph G has minimum degree at least 3, maximum degree at most d, and an l-forest deletion set of size at most k, then it has less than $2l + k(d+1)$ vertices and less than $2kd + 3l$ edges.*

Lemma 10 gives rise to the following reduction rule immediately.

Reduction Rule 7. *After the application of reduction rules 4, 5 and 6 exhaustively, if either $|V(G)| \geq 2l + k(d+1)$ or $|E(G)| \geq 2kd + 3l$, where d is the maximum degree of the graph, return that the given instance is a* No *instance.*

After this, all that is left is to reduce the maximum degree of the graph. After the exhaustive application of reduction rules 4, 5 and 6, if the maximum degree of the graph is already bounded by $(k+l)(3l+8)$ then we already have a kernel with $\mathcal{O}(kl(k+l))$ vertices and $\mathcal{O}(kl(k+l))$ edges. Hence we assume, for the rest of the section, that after the exhaustive application of reduction rules 4- 7, there exists a vertex v with degree greater than $(k+l)(3l+8)$. We need one more reduction rule before we proceed further.

Reduction Rule 8. *If there is a vertex v with more than l self loops, delete v and decrease k by 1.*

We now try to reduce the high degree vertices. The idea is that either a high degree vertex participates in many cycles (and contributes many excess edges) and hence should be part of the solution, or only a small part of its neighbourhood is relevant for the solution. We formalize these notions by use of Gallai's theorem to find flowers and applying a set of reduction rules. Given a set $T \subseteq V(G)$, by T-path we mean set of paths of positive length with both endpoints in T.

Theorem 11 (Gallai, [11]). *Given a simple graph G, a set $T \subseteq V(G)$ and an integer s, one can in polynomial time find either*

- *a family of $s+1$ pairwise vertex-disjoint T-paths, or*
- *a set B of at most $2s$ vertices, such that in $G - B$ no connected component contains more than one vertex of T.*

We would want to have the neighborhood of a high degree vertex as the set T for applying Gallai's theorem and for detecting flowers. But we need to be careful, as the graph in its current form contains multiple edges and self loops. Let v be a vertex with high degree. The vertices in $N(v)$ which have at least two parallel edges to v can be greedily picked to form a petal of the flower. Let L be the set of vertices in $N(v)$ which have at least two parallel edges to v.

Reduction Rule 9. *If $|L| > k + l$, delete v and decrease k by* 1.

Let $\widehat{G}$ be the graph $G - L$ with all parallel edges replaced with single edges, and all self loops removed. It is not hard to show that finding an f-flower in G centered at v is equivalent to finding an $f - |L|$ flower in $\widehat{G}$ centered at v for any $f \geq |L|$. Now we apply Gallai's theorem on $\widehat{G}$ with $T = N(v)$ and $s = k+l-|L|$. If the theorem returns a collection of vertex disjoint T-paths, then it is easy to see that they are in one to one correspondence with cycles including v, and hence can be considered petals of the flower centered at v.

Reduction Rule 10. *If the application of Gallai's theorem returns a flower with more than s petals, then delete v and decrease k by* 1.

We now deal with the case when the application of Gallai's theorem returns a set B of at most $2(k+l-|L|)$ vertices, such that in $\widehat{G}-B$ no connected component contains more than one vertex of T. Let $Z = B \cup L$. Clearly, $|Z| \leq 2(k+l) - |L|$. Now we look at the set of connected components of $\widehat{G} - (Z \cup \{v\})$. Let us call this set C.

Reduction Rule 11. *If more than $k+l$ components of C contain a cycle, then return that the instance is a* No *instance.*

Lemma 12. *After applying reduction rules* $4-11$ *exhaustively, there are at least* $2(l+2)(k+l)$ *components in C which are trees and connected to v with exactly one edge.*

Proof. The number of self loops on v is bounded by l due to Reduction Rule 8. Number of edges from v to Z is bounded by $|B|+(l+2)|L| \leq 2(k+l-|L|)+(l+2)|L| = 2(k+l)+l|L| \leq (k+l)(l+2)$. As degree of v is greater than $(k+l)(3l+8)$, at least $(k+l)(3l+8)-(k+l)(l+2)-l \geq (k+l)(2l+5)$ connected components in C have exactly one vertex which is is neighbour of v. Out of these, the number of connected components containing cycles is bounded by $k+l$ by Reduction Rule 11. Hence, at least $2(l+2)(k+l)$ connected components are trees and are connected to v by exactly one edge. □

Before we proceed further, we state the Expansion Lemma. Let G be a bipartite graph with vertex bipartition (A, B). For a positive integer q, a set of edges $M \subseteq E(G)$ is called a q-expansion of A into B if every vertex of A is incident with exactly q edges of M, and exactly $q|A|$ vertices in B are incident to M.

Lemma 13 (Expansion Lemma, [8]). *Let $q \geq 1$ be a positive integer and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$ and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there is a q-expansion of X into Y and no vertex in Y has a neighbor outside X, that is, $N(Y) \subseteq X$. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

Let D the set of connected components of C which are trees and connected to v with exactly one edge. We have shown that $|D| \geq 2(l+2)(k+l)$. Now we construct an auxiliary bipartite graph H as follows. In one partition of H, we have a vertex for every connected component in D, and the other partition is Z. We put an edge between $A \in D$ and $v \in Z$ if some vertex of A is adjacent to v. Since every connected component in D is a tree and has only one edge to v, some vertex in it has to have a neighbour in Z, otherwise Reduction Rule 1 would apply. Now we have that $|Z| \leq 2(k+l)$ and every vertex in D is adjacent to some vertex in Z, we may apply Expansion Lemma with $q = l+2$. This means, that in polynomial time, we can compute a nonempty set $\widehat{Z} \subseteq Z$ and a set of connected components $\widehat{D} \subseteq D$ such that:

1. $N_G(\bigcup_{D \in \widehat{D}} D) = \widehat{Z} \cup \{v\}$, and
2. Each $z \in \widehat{Z}$ will have $l+2$ private components $A_z^1, A_z^2, \ldots A_z^{l+2} \in \widehat{D}$ such that $z \in N_G(A_z^i)$ for all $i \in [l+2]$. By private we mean that the components $A_z^1, A_z^2, \ldots A_z^{l+2}$ are all different for different $z \in \widehat{Z}$.

Lemma 14. *For any l-forest deletion set X of G that does not contain v, there exists an l-forest deletion set X' in G such that $|X'| \leq |X|$, $X' \cap (\bigcup_{A \in \widehat{D}} A) = \emptyset$ and $\widehat{Z} \subseteq X'$.*

Now we are ready to give the final reduction rule.

Reduction Rule 12. *Delete all edges between v and $\bigcup_{A \in \widehat{D}} A$ and put $l+2$ parallel edges between v and z for all $z \in \widehat{Z}$.*

Theorem 15. Almost Forest Deletion *admits a kernel with $\mathcal{O}(kl(k+l))$ vertices and $\mathcal{O}(kl(k+l))$ edges.*

Proof. First we show that either we have a kernel of the desired size or one of the reduction rules 4-12 apply. So, we only need to define a measure which is polynomial in the size of the graph and show that each of the reduction rules decrease the measure by a constant. We define the measure of a graph G to be $\phi(G) = 2|V(G)| + |E_{\leq l+2}|$, where $E_{\leq l+2}$ is set of edges with multiplicity at most $l+2$. Then we show that each of the reduction rules either terminate the algorithm or decrease the measure by a constant. □

5 An $O^*(c^{\mathbf{tw}})$ Algorithm for Almost Forest Deletion

In this section, we first design an algorithm, which given an instance (G,k) of Almost Forest Deletion along with a tree decomposition of G of width at most $\mathbf{tw}$, solves it in time $O^*(c^{\mathbf{tw}})$. Then, using that algorithm, we give a subexponential time algorithm for Almost Forest Deletion on planar graphs.

Theorem 16. *Given an instance (G,k,l) of* Almost Forest Deletion *along with tree decomposition of G of width at most $\mathbf{tw}$, it can be solved in $\mathcal{O}^*(c^{\mathbf{tw}})$ time.*

Proof. For solving the problem in desired running time, we do dynamic programming on the tree decomposition of G in a bottom-up manner, while storing partial solution for each node of the tree. We use representative sets to store the information about connectivity of the partial solutions. This can be done using graphic matroid of a clique on vertices which are mapped to the tree node. We also need to store the information about the extra edges. But since in an l-forest it does not matter where exactly the extra edges are, we can make use of this fact and solve the problem efficiently. □

Theorem 17 (Planar Extended Grid Theorem [12,22]). *Let t be a non-negative integer. Then every planar graph G of treewidth at least $\frac{9}{2}t$ contains $\boxplus_t$ as a minor. Furthermore, for every $\epsilon > 0$ there exists an $\mathcal{O}(n^2)$ algorithm that, for a given n-vertex planar graph G and integer t, either outputs a tree decomposition of G of width at most $(\frac{9}{2} + \epsilon)t$, or returns that $\boxplus_t$ is a minor of G, where $\boxplus_t$ denotes a grid of dimension $t \times t$.*

Lemma 18. *Let X be an l-forest deletion set of $\boxplus_t$ of size at most k, then $t \leq \sqrt{l + 3k} + 1$.*

Theorems 16 and 17, along with Lemma 18 are combined to get the subexponential time algorithm on planar graphs.

Theorem 19. ALMOST FOREST DELETION *can be solved in $2^{\mathcal{O}(\sqrt{l+k})}n^{\mathcal{O}(1)}$ time on planar graphs.*

6 Conclusions

In this paper we studied ALMOST FOREST DELETION and obtained a polynomial kernel as well as a single exponential time algorithm for the problem. It would be interesting to study other classical problems from this view-point of distance from tractability using a suitable measure of distance.

References

1. Amini, O., Fomin, F.V., Saurabh, S.: Implicit branching and parameterized partial cover problems. J. Comput. Syst. Sci. **77**(6), 1159–1171 (2011)
2. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. Comput. J. **51**(3), 255–269 (2008)
3. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22–25, 2011, pp. 150–159 (2011)
4. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer (2013)

5. Fellows, M.R., Jansen, B.M.P., Rosamond, F.A.: Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. Eur. J. Comb. **34**(3), 541–566 (2013)
6. Fellows, M.R., Lokshtanov, D., Misra, N., Mnich, M., Rosamond, F.A., Saurabh, S.: The complexity ecology of parameters: An illustration using bounded max leaf number. Theory Comput. Syst. **45**(4), 822–848 (2009)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series). Springer-Verlag New York Inc., Secaucus (2006)
8. Fomin, F.V., Lokshtanov, D., Misra, N., Philip, G., Saurabh, S.: Hitting forbidden minors: approximation and kernelization. In: 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10–12, 2011, Dortmund, Germany, pp. 189–200 (2011)
9. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar f-deletion: approximation, kernelization and optimal FPT algorithms. In: 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20–23, 2012, pp. 470–479 (2012)
10. Fomin, F.V., Lokshtanov, D., Raman, V., Saurabh, S.: Subexponential algorithms for partial cover problems. Inf. Process. Lett. **111**(16), 814–818 (2011)
11. Gallai, T.: Maximum-minimum stze und verallgemeinerte faktoren von graphen. Acta Mathematica Academiae Scientiarum Hungarica **12**(1–2), 131–173 (1964)
12. Gu, Q., Tamaki, H.: Improved bounds on the planar branchwidth with respect to the largest grid minor size. Algorithmica **64**(3), 416–453 (2012)
13. Guo, J., Hüffner, F., Niedermeier, R.: A structural view on parameterizing problems: distance from triviality. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 162–173. Springer, Heidelberg (2004)
14. Kawarabayashi, K., Mohar, B., Reed, B.A.: A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width. In 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25–28, 2008, Philadelphia, PA, USA, pp. 771–780 (2008)
15. Kim, E.J., Langer, A., Paul, C., Reidl, F., Rossmanith, P., Sau, I., Sikdar, S.: Linear kernels and single-exponential algorithms via protrusion decompositions. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 613–624. Springer, Heidelberg (2013)
16. Kloks, T.: Treewidth, Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
17. Kociumaka, T., Pilipczuk, M.: Faster deterministic feedback vertex set. Inf. Process. Lett. **114**(10), 556–560 (2014)
18. Komusiewicz, C., Niedermeier, R., Uhlmann, J.: Deconstructing intractability - A multivariate complexity analysis of interval constrained coloring. J. Discrete Algorithms **9**(1), 137–151 (2011)
19. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. J. Comput. Syst. Sci. **20**(2), 219–230 (1980)
20. Niedermeier, R.: Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications). Oxford University Press, USA (2006)
21. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. **32**(4), 299–301 (2004)
22. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Comb. Theory, Ser. B **62**(2), 323–348 (1994)

Approximation Algorithms

Improved Approximation Algorithms for Min-Max and Minimum Vehicle Routing Problems

Wei Yu and Zhaohui Liu(✉)

Department of Mathematics, East China University of Science and Technology, Shanghai 200237, China
{yuwei,zhliu}@ecust.edu.cn

Abstract. Given an undirected weighted graph $G = (V, E)$, a set $C_1, C_2, \ldots, C_k$ of cycles is called a *cycle cover* of V' if $V' \subset \cup_{i=1}^{k} V(C_i)$ and its cost is the maximum weight of the cycles. The Min-Max Cycle Cover Problem(MMCCP) is to find a minimum cost cycle cover of V with at most k cycles. The Rooted Min-Max Cycle Cover Problem(RMMCCP) is to find a minimum cost cycle cover of $V \backslash D$ with at most k cycles and each cycle contains one vertex in D. The Minimum Cycle Cover Problem(MCCP) aims to find a cycle cover of V of cost at most λ with minimum number of cycles. We propose approximation algorithms for the MMCCP, RMCCP and MCCP with ratios 5, 6 and 24/5, respectively. Our results improve the previous algorithms in term of both approximation ratios and running times. Moreover, we transform a ρ-approximation algorithm for the TSP into approximation algorithms for the MMCCP, RMCCP and MCCP with ratios 4ρ, $4\rho + 1$ and 4ρ, respectively.

Keywords: Vehicle routing · Cycle cover · Traveling salesman problem · Approximation algorithm

1 Introduction

In the last two decades, considerable research attention has been devoted to the following fundamental vehicle routing problem. Given a fleet of k vehicles and a general network, there is exactly one customer located at each vertex. Each vehicle has to start from some vertex to visit some customers and return to the same vertex. There is a travel cost for each pair of vertices that obeys the triangle inequality. The goal is to find a routing for the vehicles to collectively visit all the customers such that the maximum traveling cost of the vehicles is minimum. If described by graph theoretic language, the above problem is to cover all the vertices of an undirected weighted graph with at most k cycles such that the maximum weight of the cycles is minimum. It is called the Min-Max Cycle Cover Problem(MMCCP) in the literature(see [15]). In the rooted version, called the Rooted Min-Max Cycle Cover Problem(RMMCCP), the objective is to use at most k rooted cycles, i.e., cycles contain one vertex of a given depot set

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 147–158, 2015.
DOI: 10.1007/978-3-319-21398-9_12

of vertices, to cover the non-depot vertices such that the maximum weight of the cycles is minimum. In the MMCCP and RMMCCP, if an upper bound $\lambda > 0$ is given for the weight of each cycle and the goal is to minimize the number of cycles used to cover the vertices, we obtain the Minimum Cycle Cover Problem(MCCP) and the Rooted Minimum Cycle Cover Problem(RMCCP), respectively.

The above-mentioned vehicle routing problems and their variants find numerous applications in operations research and computer science. RMMCCP and MMCCP were introduced by Even et al. [6] to model "Nurse Station Location Problem". Campbell et al. [4] illustrated how disaster relief efforts can be improved by efficient algorithms for min-max cycle/path cover problems. Xu et al. [15] described some applications of cycle cover problems in wireless sensor networks. For more practical examples involving min-max and minimum vehicle routing problems we refer to [1,14,16–18] and the references therein.

Unfortunately, all the problems RMMCCP, MMCCP, RMCCP, MCCP are NP-hard since they are extensions of the well-know Traveling Salesman Problem. Therefore, previous results mainly focus on devising approximation algorithms with good performance ratios.

1.1 Previous Works

Xu et al. [18] showed that both the MMCCP and the RMMCCP cannot be approximated within ratio 4/3, unless P=NP. Xu and Wen [16] gave an inapproximability bound of 20/17 for the single-depot RMMCCP. By the NP-completeness of the well-known Hamiltonian Cycle Problem, both the MCCP and the RMCCP can not be approximated within ratio 2.

For the MMCCP, a closely related problem, called the Min-Max Tree Cover Problem(MMTCP), can be obtained by replacing cycles with trees. On the one hand, the optimal value of the MMTCP can not be greater than that of the MMCCP. On the other hand, by duplicating each edge of a feasible solution of the MMTCP we obtain a feasible solution of MMCCP with the objective value doubled. Therefore, any α-approximation algorithm for the MMTCP implies a 2α-approximation algorithm for the MMCCP. Even et al. [6] and Arkin et al. [1] developed independently 4-approximation algorithms for the MMTCP by different algorithmic techniques. Khani and Salavatipour [10] give an improved 3-approximation algorithm, which implies a 6-approximation algorithm for the MMCCP. Xu et al. [18] also derived an approximation algorithm with the same ratio 6. These algorithms were improved to a 16/3-approximation algorithm by Xu et al. [15].

For the RMMCCP, Xu et al. [18] proposed a 7-approximation algorithm. Later, Xu et al. [15] improved the approximation ratio to 19/3. For the single-depot RMMCCP, Frederickson et al. [7] achieved a better ratio of $\rho+1$, where ρ is the approximation ratio of the best available algorithm for the Traveling Salesman Problem. By using the well-known Christofides' Algorithm[5] this implies a 5/2-approximation algorithm. Moreover, Nagamochi [11], Nagamochi and Okada [11,13] obtained better results on a special case of the RMMCCP where the graph is the metric closure of a tree.

In the MCCP, by replacing cycles with trees we derive the Minimum Tree Cover Problem(MTCP), which is also named as Bounded Tree Cover Problem in [10]. Since a cycle of weight at most λ can be splitted into two paths(which are also trees) of weight at most $\frac{\lambda}{2}$ by removing properly two edges, two times the optimal value of the MCCP can not be less than the optimal value of the corresponding MTCP with the upper bound on the weight of the trees reset to $\frac{\lambda}{2}$. On the other hand, by doubling the edges of a feasible solution of the MTCP with the revised upper bound we obtain a feasible solution of the MCCP. Therefore, any α-approximation algorithm for the MTCP implies a 2α-approximation algorithm for the MCCP. Arkin et al. [1] developed a 6-approximation algorithm for the MCCP. This result is also implied by the 3-approximation algorithm for the MTCP in the same paper. Khani and Salavatipour [10] gave an improved 5/2-approximation algorithm for the MTCP, which indicates a 5-approximation algorithm for the MCCP.

The approximability of the RMCCP is far less understood. All the existing results focus on the case of one single depot vertex, which are called the Distance Constrained Vehicle Routing Problem by Nagarajan and Ravi [14]. The authors proposed a $\min\{\log n, \log \lambda\}$-approximation algorithm for the general problem and a 2-approximation algorithm for the problem defined on the metric closure of a tree. Recently, Friggstad and Swamy [8] obtained an improved $\frac{\log \lambda}{\log \log \lambda}$-approximation algorithm for the general problem.

1.2 Our Results and Techniques

In this paper we focus on the MMCCP, RMMCCP and MCCP. Our main contributions are fourfold. Firstly, we obtain a 5-approximation algorithm for the MMCCP, which improves the previous best 16/3-approximation algorithm by Xu et al. [15]. Meanwhile, this algorithm also improves the running time from $O(n^5 \log \sum_{e \in E} w(e))$ to $O(n^3 \log \sum_{e \in E} w(e))$, where n is the number of vertices, E is the edge set of the graph and $w(e)$ is the weight of edge e. Moreover, we transform a ρ-approximation algorithm for the TSP into a 4ρ-approximation algorithm for the MMCCP, which implies further improvement on the performance ratio for the problem defined on some special metrics(e.g. Euclidean metric). Secondly, we show that any α-approximation algorithm for the MMCCP implies an $(\alpha + 1)$-approximation algorithm for the RMMCCP. This indicates a 6-approximation algorithm for the RMMCCP, beating the 19/3-approximation algorithm in [15] in term of both performance ratio and running time. Thirdly, we devise a 24/5-approximation algorithm for the MCCP with running time $O(n^4)$. In contrast, the previous best 5-approximation algorithm by Khani and Salavatipour [10] runs in $O(n^5)$ time. Lastly, we introduce a new matching-based upper bound analysis for the MMCCP which proves to be more efficient than the strategy of doubling tree edges in the literature.

The rest of the paper is organized as follows. We formally state the problems and give some preliminary results in Section 2. We deal with the MMCCP and RMMCCP in Section 3 and Section 4, respectively. The MCCP is treated in Section 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 $B > 0$, $G[B]$ is 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, matching) 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 $w_T(H)$. A cycle C is also called a tour on $V(C)$. The weight of a path or cycle is also called its length. A cycle(path, tree) containing only one vertex and no edges is a trivial cycle(path, tree) and its weight is defined as zero.

For a subset V' of V, a set $C_1, C_2, \ldots, C_k$ of cycles(some of them may be trivial cycles) is called a *cycle cover* of V' if $V' \subset \cup_{i=1}^{k} V(C_i)$. And the cost of this cycle cover is defined as $\max_{1 \le i \le k} w(C_i)$, i.e., the maximum weight of the cycles. Particularly, a cycle cover of V is simply called a cycle cover. By replacing cycles with trees we can define *tree cover* and its cost similarly.

Now we formally state the problems.

Definition 1. *In the Min-Max Cycle Cover Problem(MMCCP), we are given an undirected complete graph $G = (V, E)$ with a metric nonnegative weight function w on E and a positive integer k, the goal is to find a minimum cost cycle cover with at most k cycles.*

Definition 2. *In the Rooted Min-Max Cycle Cover Problem(RMMCCP), we are given an undirected complete graph $G = (V, E)$ with a metric nonnegative weight function w on E, a depot set $D \subset V$, and a positive integer k, the objective is to find a minimum cost cycle cover of $V \backslash D$ with at most k cycles such that each cycle contains exactly one vertex of D.*

Definition 3. *In the Minimum Cycle Cover Problem(MCCP), we are given an undirected complete graph $G = (V, E)$ with a metric nonnegative weight function w on E and a positive λ, the aim is to find a cycle cover of cost at most λ such that the number of cycles in the cycle cover is minimum.*

Note that in the above problem definitions we assume that the graph is complete. This involves no loss of generality, since we can take the metric closure of a connected graph G if it is not complete. When G is not connected, we simply consider the corresponding problems defined on each connected component. We also suppose w.l.o.g. that the weight of the edges and λ are integers.

Given an instance of the MMCCP(RMMCCP, MCCP), OPT indicates the 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 cycle-splitting and tree-decomposition results are very useful in the design and analysis of our algorithms.

Lemma 1. *[1, 7, 18] Given a tour C on V' and $B > 0$, we can split the tour into $\lceil \frac{w(C)}{B} \rceil$ paths of length at most B such that each vertex is located at exactly one path in $O(|V'|)$ time.*

Lemma 2. *[6, 10] 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 $T_1, T_2, \ldots, T_k$ with $w(T_i) \leq 2B$ for each $i = 1, 2, \ldots, k$ in $O(|V(T)|)$ time.*

3 Min-Max Cycle Cover

In this section we first show how to transform a ρ-approximation algorithm for the TSP into a 4ρ-approximation algorithm for the MMCCP. Next we give a 5-approximation algorithm for the MMCCP with running time $O(n^3 \log \sum_{e \in E} w(e))$.

As in the previous results, an α-approximation algorithm can be derived in two phases. First, we guess a objective value λ. If $OPT \leq \lambda$, we succeed in constructing a cycle cover with at most k cycles of cost no more than $\alpha\lambda$. Second, by a binary search in $[0, \sum_{e \in E} w(e)]$ we find the minimum value λ^* such that a cycle cover with at most k cycles whose cost is at most $\alpha\lambda^*$ can be constructed. This cycle cover is an α-approximate solution since $\lambda^* \leq OPT$ by definition. Since the second phase is standard, we focus on the first phase.

Given a ρ-approximation algorithm for the TSP, our first algorithm is described below.

Algorithm $MMCCP(\rho, \lambda)$

Step 1. Delete all the edges with weight greater than $\frac{\lambda}{2}$ in G. The resulted graph $G[\frac{\lambda}{2}]$ has p connected components $F_1, F_2, \ldots, F_p$.

Step 2. For each $i = 1, 2, \ldots, p$, find a ρ-approximate tour TSP_i on $V(F_i)$ and split it into $k_i = \max\{\lceil \frac{w(TSP_i)}{2\rho\lambda} \rceil, 1\}$ paths of length no more than $2\rho\lambda$ by Lemma 1.

Step 3. Connect the two end vertices of each path constructed in Step 2 to obtain $\sum_{i=1}^{p} k_i$ cycles which constitute a cycle cover. If $\sum_{i=1}^{p} k_i \leq k$, return this cycle cover; otherwise, return failure.

Let $C_1^*, C_2^*, \ldots, C_{k'}^*$ with $k' \leq k$ be all the vertex-disjoint optimum cycles. First, we give two observations that are also noted in [18] and [15]. For any $e = (u, v) \in C_i^*$, the weight of C_i^* consists of $w(e)$ and the weight of a path P from u to v. By the triangle inequality, $w(P) \geq w(e)$. This implies $OPT \geq w(C_i^*) \geq 2w(e)$. So we have

Observation 1. *If $OPT \leq \lambda$, then $w(e) \leq \frac{\lambda}{2}$ for each $e \in \cup_{i=1}^{k'} E(C_i^*)$.*

By this observation the vertex set of each optimum cycle is contained entirely in exactly one of $V(F_1), V(F_2), \ldots, V(F_p)$. As a consequence, the optimum cycles whose vertex sets are contained in $V(F_i)$ constitute a cycle cover of $V(F_i)$. Moreover, the cost of this cycle cover of $V(F_i)$ is at most OPT since the length of each optimum cycle is no more than OPT.

Observation 2. *If $OPT \leq \lambda$, the optimum cycles can be partitioned into p groups such that the $i^{th}(i = 1, 2, \ldots, p)$ group consisting of $k_i^* \geq 1$ optimum cycles is a cycle cover of $V(F_i)$ with cost at most λ.*

This observation leads to an upper bound on the length of the tour on $V(F_i)$.

Lemma 3. *If $OPT \leq \lambda$, $w(TSP_i) \leq \rho(2k_i^* - 1)\lambda$ for $i = 1, 2, \ldots, p$.*

Proof. For each $i = 1, 2, \ldots, p$, by Observation 2 we have $k_i^* \geq 1$ optimum cycles that constitute a cycle cover of $V(F_i)$. Since F_i is a connected component we can add a set $E_i \subset E(F_i)$ of $k_i^* - 1$ edges to these cycles to obtain a connected subgraph H_i. After that we double all the edges of E_i in H_i to obtain a Eulerian graph H_i' on $V(F_i)$. By shortcutting the repeated vertices of the Eulerian tour of H_i' we generate a tour T_i on $V(F_i)$. Due to $E_i \subset E(F_i)$ and Step 1, $w(e) \leq \frac{\lambda}{2}$ for all $e \in E_i$, which implies $w(E_i) \leq (k_i^* - 1) \cdot \frac{\lambda}{2}$. So

$$w(T_i) \leq w(H_i') \leq k_i^* \lambda + 2w(E_i) \leq k_i^* \lambda + 2(k_i^* - 1) \cdot \frac{\lambda}{2} = (2k_i^* - 1)\lambda, \qquad (1)$$

where the second inequality follows from $OPT \leq \lambda$.

Since TSP_i is a ρ-approximate solution, we have $w(TSP_i) \leq \rho w(T_i)$. Combining this inequality with (1) proves the lemma. □

Now we are ready to prove the following lemma.

Lemma 4. *If $OPT \leq \lambda$, Algorithm $MMCCP(\rho, \lambda)$ returns a cycle cover with at most k cycles whose cost is at most $4\rho\lambda$ in polynomial time.*

Proof. For each $i = 1, 2, \ldots, p$, TSP_i is a tour on $V(F_i)$. By Lemma 1, each vertex of $V(F_i)$ is contained in some path splitted from TSP_i and hence included in some cycle constructed in Step 3. Consequently, the set of cycles in Step 3 constitute a cycle cover. Since $OPT \leq \lambda$, by Lemma 3 we have

$$\sum_{i=1}^{p} k_i = \sum_{i=1}^{p} \max\{\lceil \frac{w(TSP_i)}{2\rho\lambda} \rceil, 1\} \leq \sum_{i=1}^{p} \max\{\lceil k_i^* - \frac{1}{2} \rceil, 1\} \leq \sum_{i=1}^{p} k_i^* = k' \leq k\,,$$

where the second inequality follows from the integrality of $k_i^* \geq 1$ and the last inequality holds since k' is the number of cycles used by the optimal solution.

Therefore, the algorithm returns the cycle cover generated in Step 3. To see that the cost of this cycle cover is at most $4\rho\lambda$, it is sufficient to note that all the paths derived in Step 2 have a length of at most $2\rho\lambda$ and the weight of the edge connecting the two end vertices of each path cannot exceed the length of the path due to the triangle inequality.

As for the time complexity, Step 1 takes $O(n^2)$ time. In Step 2, finding the approximate tour can be done in polynomial time since we run a polynomial time approximation algorithm for the TSP, and the cycle-splitting procedure takes $O(n)$ time by Lemma 1. Step 3 can also be completed in $O(n)$ time. To sum up, Algorithm $MMCCP(\rho, \lambda)$ runs in polynomial time. □

Using Lemma 4 at most $\log \sum_{e \in E} w(e)$ times to conduct a binary search we obtain the following theorem.

Theorem 1. *Given a ρ-approximation algorithm for the TSP, there is a 4ρ-approximation algorithm for the MMCCP.*

This theorem implies good approximation algorithms for the MMCCP defined on some special metrics. Particularly, by the PTAS for the Euclidean TSP given by Arora [2] we have the following result.

Corollary 1. *For any $\epsilon > 0$, there is a $(4+\epsilon)$-approximation algorithm for the MMCCP defined on any fixed d-dimensional Euclidean space.*

Remark 1. Karakawa et al. [9] proved a stronger version of Lemma 2 for trees on a Euclidean space, which can derive by a similar approach in [15] approximation algorithms for MMCCP defined on a d-dimensional Euclidean space with ratios $5.208(d=2)$ and $5.237(d \geq 3)$, respectively.

In what follows, we plug in Christofides' Algorithm in Step 2 of Algorithm $MMCCP(\rho, \lambda)$ and make a refined analysis to obtain a 5-approximation algorithm. The modified algorithm is described as follows:

Algorithm $MMCCP(\lambda)$

```
Step 1. Delete all the edges with weight greater than $\frac{\lambda}{2}$ in $G$. The resulted graph
    $G[\frac{\lambda}{2}]$ has $p$ connected components $F_1, F_2, \ldots, F_p$.
Step 2. For each $i = 1, 2, \ldots, p$, compute $MST(F_i)$ and determine the set $S_i$
    of vertices in $V(F_i)$ that are of odd degree in $MST(F_i)$. Find a minimum
    weight perfect matching $M_i$ on $S_i$ and add $M_i$ to $MST(F_i)$ to obtain an
    Eulerian graph $G_i$ on $V(F_i)$. Shortcut the repeated vertices of the Eulerian
    tour of $G_i$ to obtain a tour $TSP_i$ on $V(F_i)$. Split the tour into at most
    $k_i = \max\{\lceil \frac{w(TSP_i)}{\frac{5}{2}\lambda} \rceil, 1\}$ paths of length no more than $\frac{5}{2}\lambda$ by Lemma 1.
Step 3. Connect the two end vertices of each path constructed in Step 2 to
    obtain $\sum_{i=1}^{p} k_i$ cycles which constitute a cycle cover. If $\sum_{i=1}^{p} k_i \leq k$, return
    this cycle cover; otherwise, return failure.
```

Lemma 5. *If $OPT \leq \lambda$, $w(TSP_i) \leq (\frac{5}{2}k_i^* - 1)\lambda$ for $i = 1, 2, \ldots, p$.*

Proof. For each $i = 1, 2, \ldots, p$, we construct subgraphs H_i, H_i' and tour T_i in exactly the same way as in the proof of Lemma 3. Since H_i is a connected subgraph of F_i we have

$$w_T(F_i) \leq w(H_i) \leq k_i^* \lambda + w(E_i) \leq k_i^* \lambda + (k_i^* - 1) \cdot \frac{\lambda}{2} = \left(\frac{3}{2}k_i^* - \frac{1}{2}\right)\lambda, \tag{2}$$

where the second inequality follows from $OPT \leq \lambda$ and the third inequality holds by $w(e) \leq \frac{\lambda}{2}$ for all $e \in E_i$.

By shortcutting we can transform tour T_i into a tour T_i' on $V(M_i)$ with $w(T_i') \leq w(T_i)$ due to the triangle inequality. It is well known that T_i' can be decomposed into two edge-disjoint perfect matching on $V(M_i)$. Thus by the optimality of M_i we have

$$w(M_i) \leq \frac{1}{2}w(T_i') \leq \frac{1}{2}w(T_i) \leq \left(k_i^* - \frac{1}{2}\right)\lambda, \tag{3}$$

where the last inequality follows from (1). Therefore, by (2) and (3) we obtain

$$w(TSP_i) \leq w_T(F_i) + w(M_i) \leq \left(\frac{3}{2}k_i^* - \frac{1}{2}\right)\lambda + \left(k_i^* - \frac{1}{2}\right)\lambda = \left(\frac{5}{2}k_i^* - 1\right)\lambda. \square$$

It can be seen that finding TSP_i for each $i = 1, 2, \ldots, p$ takes $O(|V(F_i)|^3)$, which dominates the time complexity of Algorithm $MMCCP(\lambda)$. Consequently, the algorithm runs in $\sum_{i=1}^{p} O(|V(F_i)|^3)) = O(\sum_{i=1}^{p} |V(F_i)|^3) = O(n^3)$ time.

By Lemma 5 and a similar proof to Lemma 4 we derive the following lemma.

Lemma 6. *If $OPT \leq \lambda$, Algorithm $MMCCP(\lambda)$ returns a cycle cover with at most k cycles whose cost is at most 5λ in $O(n^3)$ time.*

Using this lemma to perform a binary search we obtain

Theorem 2. *There is a 5-approximation algorithm for the MMCCP that runs in $O(n^3 \log \sum_{e \in E} w(e))$ time.*

Combining Theorem 1 and Theorem 2 we have

Theorem 3. *Given a ρ-approximation algorithm for the TSP, there exists a* $\min\{4\rho, 5\}$*-approximation algorithm for the MMCCP.*

4 Rooted Min-Max Cycle Cover

One can transform an α-approximation algorithm for the MMCCP into an $(\alpha+1)$-approximation algorithm for the RMMCCP as follows. First, by ignoring the depot set D we obtain an instance of the MMCCP. Then we run the α-approximation algorithm for this instance to obtain a cycle cover $C_1, C_2, \ldots, C_{\bar{k}}$ of $V \backslash D$ with $\bar{k} \leq k$. Next for each $i = 1, 2, \ldots, \bar{k}$ we choose an arbitrary vertex v_i from C_i with $(u_i', v_i), (u_i'', v_i) \in E(C_i)$, determine $d_i \in D$ such that $w(v_i, d_i) = \min_{d \in D} w(v_i, d)$ and derive a cycle C_i' with $E(C_i') = (E(C_i) \setminus \{(u_i', v_i), (u_i'', v_i)\}) \cup \{(u_i', d_i), (u_i'', d_i)\}$. Then $C_1', C_2', \ldots, C_{\bar{k}}'$ is a feasible cycle cover for the RMMCCP. To show this is indeed an $(\alpha+1)$-approximation algorithm, we only need two facts: (i)the optimal value of the instance of the MMCCP can not exceed OPT, i.e., the optimal value of the original instance of the RMMCCP; (ii)$w(C_i') \leq w(C_i) + 2w(v_i, d_i)$ and $w(v_i, d_i) \leq OPT/2$ for each i. The first inequality follows from the triangle inequality. The second one holds because each v_i must locate in the same optimum cycle with some depot $d_i' \in D$ of weight no more than OPT, one of the two paths along the optimum cycle between v and d_i' is of length at most $OPT/2$. By the triangle inequality, $w(v_i, d_i') \leq OPT/2$ and hence $w(v_i, d_i) \leq w(v_i, d_i') \leq OPT/2$.

Therefore, all the results on the MMCCP can be applied to the RMMCCP with a loss of 1 in the approximation ratio.

Theorem 4. *Given a ρ-approximation algorithm for the TSP, there exists a* $\min\{4\rho+1, 6\}$*-approximation algorithm for the RMMCCP. Particularly, there is an* $O(n^3 \log \sum_{e\in E} w(e))$ *time 6-approximation algorithm for the RMMCCP.*

5 Minimum Cycle Cover

In this section we give approximation algorithms for the MCCP. First, we show how to apply the results for the MMCCP in Section 3 to obtain approximation algorithms with the same ratio. After that we propose an algorithm with better performance ratio for the MCCP.

Recall that in the MCCP, $\lambda > 0$ is given in advance and the aim is to find a cycle cover of cost at most λ such that the number of cycles is minimum. To turn Algorithm $MMCCP(\rho, \lambda)$ into an approximation algorithm for the MCCP, we need only split the tour TSP_i into paths of length at most $\frac{\lambda}{2}$ instead of $2\rho\lambda$ in Step 2. Moreover, in Step 3 we always return the cycle cover of cost at most λ.

Algorithm $MCCP(\rho)$

Step 1. Delete all the edges with weight greater than $\frac{\lambda}{2}$ in G. The resulted graph $G[\frac{\lambda}{2}]$ has p connected components $F_1, F_2, \ldots, F_p$.

Step 2. For each $i = 1, 2, \ldots, p$, find a ρ-approximate tour TSP_i on $V(F_i)$ and split it into $k_i = \max\{\lceil \frac{w(TSP_i)}{\frac{\lambda}{2}} \rceil, 1\}$ paths of length at most $\frac{\lambda}{2}$ by Lemma 1.

Step 3. Connect the two end vertices of each path constructed in Step 2 to obtain $\sum_{i=1}^{p} k_i$ cycles which constitute a cycle cover. Return this cycle cover.

By a similar analysis to Algorithm $MMCCP(\rho, \lambda)$ one can show the above algorithm is a 4ρ-approximation algorithm for the MCCP. A counterpart to Theorem 2 can also be established. So we have

Theorem 5. *Given a ρ-approximation algorithm for the TSP, there exists a* $\min\{4\rho, 5\}$*-approximation algorithm for the MCCP. Particularly, there exists a 5-approximation algorithm for the MCCP that runs in* $O(n^3)$ *time.*

Next we present a 24/5-approximation algorithm for the MCCP that runs in $O(n^4)$ time. In contrast, Khani and Salavatipour [10] gave a 5/2-approximation algorithm for the Minimum Tree Cover Problem that runs in $O(n^5)$ time, which implies a 5-approximation for the MCCP with the same running time. Our algorithm adopts a similar approach to the algorithm for the Min-Max Tree Cover Problem also proposed in [10]. However, we make a refined analysis on cycles instead of trees which leads to an improved approximation ratio and simplify the algorithm to obtain a better running time.

The basic idea of the algorithm is as follows. First, we delete all the edges with weight greater than $\frac{\lambda}{5}$ to obtain the graph $G[\frac{\lambda}{5}]$. Let $F_1, F_2, \ldots, F_l$ be the connected components of $G[\frac{\lambda}{5}]$ with $w_T(F_i) \le \frac{\lambda}{2} (i = 1, 2, \ldots, l)$, called *light components*. The rest of connected components $F_{l+1}, F_{l+2}, \ldots, F_{l+h}$ of $G[\frac{\lambda}{5}]$ with $w_T(F_i) > \frac{\lambda}{2} (i = l+1, l+2, \ldots, l+h)$ are called *heavy components*. Next we construct a tree cover of cost at most $\frac{\lambda}{2}$. Since $w_T(F_i) \le \frac{\lambda}{2}$ for $i = 1, 2, \ldots, l$ we choose the minimum spanning trees of some light components as the trees in the final tree cover. For the minimum spanning trees of the other light components we connect them properly to the heavy components, which results in h *modified heavy components* $F'_{l+1}, F'_{l+2}, \ldots, F'_{l+h}$. And for each modified heavy component we decompose its minimum spanning tree into a set of trees of weight at most $\frac{\lambda}{2}$ by Lemma 2 and put them into the final tree cover. Lastly, for each tree of the tree cover we double all the edges to obtain a Eulerian graph and shortcut the repeated vertices of the Eulerian tour to derive a cycle cover of cost at most λ.

To guide the choice of the minimum spanning trees of light components to be connected to heavy components, we define $w_{\min}(F_i)$ with $1 \le i \le l$ as the minimum weight of edges in G with one vertex in $V(F_i)$ and the other vertex in $\cup_{s=l+1}^{l+h} V(F_s)$ and construct a bipartite graph.

Definition 4. *Given an integer a with $0 \le a \le l$, the bipartite graph H_a has l light vertices $u_1, u_2, \ldots, u_l$, a null vertices $x_1, x_2, \ldots, x_a$ and $l - a$ heavy vertices $y_1, y_2, \ldots, y_{l-a}$. For all $i = 1, 2, \ldots, l$ and $j = 1, 2, \ldots, a$, there is an edge (u_i, x_j) of weight 0. For $i = 1, 2, \ldots, l$, if $w_{\min}(F_i) \le \frac{\lambda}{2}$ we add an edge (u_i, y_j) of weight $w_T(F_i) + w_{\min}(F_i)$ to H_a for each $j = 1, 2, \ldots, l - a$. There are no other edges in H_a.*

Now we formally describe our algorithm below.

Algorithm *MCCP*

```
Step 1. Delete all the edges with weight greater than $\frac{\lambda}{5}$ in $G$ to obtain $G[\frac{\lambda}{5}]$ with
        light components $F_1, F_2, \ldots, F_l$ and heavy components $F_{l+1}, F_{l+2}, \ldots, F_{l+h}$.
Step 2. For $a = 0, 1, \ldots, l$, set $T_a := \emptyset$ and $S_a := \emptyset$.
  (i) Find a minimum weight perfect matching $M_a$ in $H_a$(if there is no perfect
        matching in $H_a$ set $a := a + 1$ and go to Step 2);
  (ii) If $(u_i, x_j) \in M_a$, put $MST(F_i)$ into $T_a$;
  (iii) If $(u_i, y_j) \in M_a$, connect $MST(F_i)$ to some heavy component by the edge
        corresponding to $w_{\min}(F_i)$. This results in $h$ modified heavy components
        $F'_{l+1}, F'_{l+2}, \ldots, F'_{l+h}$. For each $s = l+1, l+2, \ldots, l+h$, decompose $MST(F'_s)$
        by Lemma 2 into a set of trees of weight at most $\frac{\lambda}{2}$ and put them into $T_a$.
  (iv) For each tree in $T_a$, 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. Put this cycle into $S_a$.
Step 3. Among all the nonempty $S_a$, return the one contains the minimum num-
        ber of cycles.
```

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

Let $OPT = k$ and C_1^*, C_2^*, ..., C_k^* be the vertex-disjoint optimum cycles. Given a cycle C, if $V(F_i) \cap V(C) \ne \emptyset$ for some i with $1 \le i \le l + h$, we say that F_i is *incident* to C or C 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, $k = k_l + k_h + k_b$.

Next we analyze the execution of the algorithm for $a = a'$ with $0 \leq a' \leq l$, where a' is the number of light components incident to at least one light cycle, and bound the number of cycles in $S_{a'}$ which is identical to the number of trees in $T_{a'}$. If we define F_0 as an empty light component which contains neither a vertex nor an edge, we can assume without loss of generality that $F_0, F_1, \ldots, F_{a'}$ are the light components incident to at least one light cycle. An edge of weight greater than $\frac{\lambda}{5}$ is referred to as a *long edge*. Since an optimum cycle(particularly a light cycle) is of length no more than λ, it cannot contain more than four long edges and hence $a' \leq 4k_l$, which implies

Lemma 7. *For $a = a'$, Step 2(ii) of Algorithm MCCP generates $a' \leq 4k_l$ trees.*

Lemma 8. *For $a = a'$, $\sum_{s=l+1}^{l+h} w_T(F'_s) \leq \frac{6}{5}(k_h + k_b)\lambda$.*

Lemma 9. *For $a = a'$, Step 2(iii) of Algorithm MCCP generates at most $\frac{24}{5}(k_h + k_b)$ trees.*

By Lemma 7 and Lemma 9 we deduce

Lemma 10. $|S_{a'}| = |T_{a'}| \leq \frac{24}{5}k$.

By his lemma and a simple analysis of the complexity of Algorithm *MCCP* we have

Theorem 6. *There is a $\frac{24}{5}$-approximation algorithm for the MCCP that runs in $O(n^4)$ time.*

Combining this theorem with Theorem 5 we obtain

Theorem 7. *Given a ρ-approximation algorithm for the TSP, there exists a $\min\{4\rho, \frac{24}{5}\}$-approximation algorithm for the MCCP.*

Acknowledgments. 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 number 11171106, 11301184, 11301475.

References

1. Arkin, E.M., Hassin, R., Levin, A.: Approximations for minimum and min-max vehicle routing problems. Journal of Algorithms **59**, 1–18 (2006)
2. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. Journal of the ACM **45**, 753–782 (1998)

3. Bhattacharya, B., Hu, Y.: Approximation algorithms for the multi-vehicle scheduling problem. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 192–205. Springer, Heidelberg (2010)
4. Campbell, A.M., Vandenbussche, D., Hermann, W.: Routing for relief efforts. Transportation Science **42**, 127–145 (2008)
5. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA (1976)
6. Even, G., Garg, N., Koemann, J., Ravi, R., Sinha, A.: Min-max tree covers of graphs. Operations Research Letters **32**, 309–315 (2004)
7. Frederickson, G.N., Hecht, M.S., Kim, C.E.: Approximation algorithms for some routing problems. SIAM Journal on Computing **7**(2), 178–193 (1978)
8. 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)
9. Karakawa, S., Morsy, E., Nagamochi, H.: Minmax tree cover in the euclidean space. Journal of Graph Algorithms and Applications **15**, 345–371 (2011)
10. Khani, M.R., Salavatipour, M.R.: Approximation algorithms for min-max tree cover and bounded tree cover problems. Algorithmica **69**, 443–460 (2014)
11. Nagamochi, H.: Approximating the minmax rooted-subtree cover problem. IEICE Transactions on Fundamentals of Electronics **E88–A**, 1335–1338 (2005)
12. Nagamochi, H., Okada, K.: Polynomial time 2-approximation algorithms for the minmax subtree cover problem. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 138–147. Springer, Heidelberg (2003)
13. Nagamochi, H., Okada, K.: Approximating the minmax rooted-tree cover in a tree. Information Processing Letters **104**, 173–178 (2007)
14. Nagarajan, V., Ravi, R.: Approximation algorithms for distance constrained vehicle routing problems. Networks **59**(2), 209–214 (2012)
15. Xu, W., Liang, W., Lin, X.: Approximation algorithms for Min-max Cycle Cover Problems. IEEE Transactions on Computers (2013). doi:10.1109/TC.2013.2295609
16. Xu, Z., Wen, Q.: Approximation hardness of min-max tree covers. Operations Research Letters **38**, 408–416 (2010)
17. Xu, Z., Xu, L., Li, C.-L.: Approximation results for min-max path cover problems in vehicle routing. Naval Research Logistics **57**, 728–748 (2010)
18. Xu, Z., Xu, L., Zhu, W.: Approximation results for a min-max location-routing problem. Discrete Applied Mathematics **160**, 306–320 (2012)

Improved Approximation Algorithms for the Maximum Happy Vertices and Edges Problems

Peng Zhang[1](✉), Tao Jiang[2,3], and Angsheng Li[4]

[1] School of Computer Science and Technology, Shandong University, Jinan 250101, China
algzhang@sdu.edu.cn

[2] Department of Computer Science and Engineering, University of California, Riverside, CA 92521, USA
jiang@cs.ucr.edu

[3] MOE Key Lab of Bioinformatics and Bioinformatics Division, TNLIST/Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

[4] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
angsheng@ios.ac.cn

Abstract. The Maximum Happy Vertices (MHV) problem and the Maximum Happy Edges (MHE) problem are two fundamental problems arising in the study of the homophyly phenomenon in large scale networks. Both of these two problems are NP-hard. Interestingly, the MHE problem is a natural generalization of Multiway Uncut, the complement of the classic Multiway Cut problem. In this paper, we present new approximation algorithms for MHV and MHE based on randomized LP-rounding techniques. Specifically, we show that MHV can be approximated within $\frac{1}{\Delta+1}$, where Δ is the maximum vertex degree, and MHE can be approximated within $\frac{1}{2} + \frac{\sqrt{2}}{4} f(k) \geq 0.8535$, where $f(k) \geq 1$ is a function of the color number k. These results improve on the previous approximation ratios for MHV, MHE as well as Multiway Uncut in the literature.

1 Introduction

Homophyly [5, Chapter4] is one of the basic laws governing the structures of large scale networks, which states that edges in a network tend to connect nodes with the same or similar attributes. For example, in a social network, people are more likely to connect with people they like, as the old proverb says, "birds of a feather flock together".

As another example, Li et al. [9] recently conducted an interesting experiment to predict the keywords of a paper from the citation network in high energy

P. Zhang—Work was done while the first author was visiting at the University of California - Riverside, USA.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 159–170, 2015.
DOI: 10.1007/978-3-319-21398-9_13

physics theory [1] by using the homophyly law. The network consists of 27, 770 vertices (*i.e.*, papers) and 352, 807 directed edges (*i.e.*, citations). However, only 1, 214 papers have keywords annotated by their authors. The task was to predict the keywords of the remaining papers. By the homophyly law, papers within a small community of the network should share common keywords. The prediction algorithm used by [9] is as follows. (1) Find a small community for each paper, if any. After this step, there are 20, 310 papers found in communities, and only 1, 409 papers from amongst them have keywords. (2) Extract the most popular 10 keywords from the known keywords in each community as the remarkable common attributes of this community. (3) For every paper in a community, predict that all or some of the 10 remarkable common keywords of this community are the keywords of the paper by checking whether the keywords appear in either the title or the abstract of the paper. Surprisingly, this simple rule successfully finds keywords for 14, 123 (70%) un-annotated papers. The experiment suggests that real networks do satisfy the homophyly law, and that the homophyly law could be used as a principle for predicting common attributes in a large network.

In a network where the homophyly law holds but some vertices have unknown attributes, as in the case of the above example where keywords are considered as attributes, one may consider the natural question of how to assign (or predict) attributes so that the homophyly law is followed to the greatest degree. Following this idea, and identifying attributes with colors, Li and Zhang [10] recently introduced two interesting maximization problems in terms of graph coloring. For simplicity, they focused on the case that each vertex has only one color.

Definition 1. The Maximum Happy Vertices (MHV) problem.

(Instance) We are given an undirected graph $G = (V, E)$ with vertex weights $\{w_v\}$, a color set $C = \{1, 2, \cdots, k\}$, and a partial vertex coloring function $c\colon V \mapsto C$. That is, c assigns colors only to a part of the vertices in V.

(Goal) A vertex is happy *if it shares the same color with all its neighbors. The goal is to color all the uncolored vertices such that the total weight of happy vertices is maximized.*

Definition 2. The Maximum Happy Edges (MHE) problem.

(Instance) We are given an undirected graph $G = (V, E)$ with edge weights $\{w_e\}$, a color set $C = \{1, 2, \cdots, k\}$, and a partial vertex coloring function $c\colon V \mapsto C$.

(Goal) An edge is happy *if its two endpoints have the same color. The goal is to color all the uncolored vertices such that the total weight of happy edges is maximized.*

Remarks. (i) When every vertex (resp., edge) has unit weight, the MHV (resp., MHE) problem is to maximize the total number of happy vertices (resp., edges). (ii) The partial vertex coloring function c is given in the input. When a vertex v has a color specified by function c, we also say that vertex v has a

[1] http://snap.stanford.edu/data/cit-HepTh.html.

pre-specified color (that is, $c(v)$). The MHV and MHE problems actually ask for a total vertex coloring. (iii) The coloring in MHV and MHE is completely different from the well-known Graph Coloring problem, which asks to color all vertices with the minimum number of colors such that each edge has its two endpoints with different colors.

The MHV and MHE problems can also be viewed as two classification problems. Given a set of objects to be classified and a set of colors, a classification problem can be depicted as from a very high level assigning a color to each object in a way that is consistent with some observed data or structure [1,7]. In our problems, the observed structure is homophyly.

1.1 Related Work

MHV and MHE are two quite natural and fundamental algorithmic problems. Surprisingly, as introduced before, they arise only very recently from the study of network homophyly [10]. Li and Zhang [10] proved that both MHV and MHE are already NP-hard even if the color number k is fixed. More precisely, when $k \geq 3$, MHV and MHE are NP-hard. When $k = 2$, MHV and MHE are polynomial time solvable. Li and Zhang [10] proposed a $\frac{1}{2}$-approximation algorithm for (the unit weight version of) MHE based on a combinatorial partitioning strategy. For (the unit weight version of) MHV, they gave two approximation algorithms. One algorithm is based on a greedy approach, whose approximation ratio is $\frac{1}{k}$. The other algorithm is based on a subset-growth technique, whose approximation ratio is $\Omega(\Delta^{-3})$, where Δ is the maximum vertex degree in the input graph.

The MHE problem is closely related to the Multiway Uncut problem [8], which is the complement of the classic Multiway Cut problem [2–4,6,11].

Definition 3. The Multiway Uncut problem.

(Instance) An undirected graph $G = (V, E)$ with edge weights $\{w_e\}$, and a terminal set $S = \{s_1, s_2, \cdots, s_k\}$.

(Goal) Find a partition $\{V_1, V_2, \cdots, V_k\}$ of V such that for each i, s_i is contained in V_i, and the total weights of edges not cut by the partition is maximized.

The goal of the Multiway Uncut problem is equivalent to coloring all the non-terminal vertices such that the total weight of happy edges is maximized. From the viewpoint of coloring, in the Multiway Uncut problem there is only one vertex s_i that has the pre-specified color i, for each $1 \leq i \leq k$. So, Multiway Uncut is just a special case of the MHE problem. The current best approximation ratio for Multiway Uncut is 0.8535, due to Langberg et al. [8].

Given an undirected graph $G = (V, E)$ with costs defined on edges and a terminal set $S \subseteq V$, the Multiway Cut problem asks for a set of edges with the minimum total cost such that its removal from graph G separates all terminals in S from one another. The Multiway Cut problem is NP-hard even if there are only three terminals and each edge has a unit cost [4]. The current best approximation ratio known for the Multiway Cut problem is 1.2965 [11]. Since the optimization goals of Multiway Cut and MHE are completely different (minimization vs. maximization), the approximation results for Multiway Cut do not *directly* extend to MHE.

1.2 Our Results

In this paper, we give improved approximation algorithms for MHV and MHE based on randomized rounding in linear programming. Specifically, we show that MHV can be approximated within $\frac{1}{\Delta+1}$, and MHE can be approximated within $\frac{1}{2}+\frac{\sqrt{2}}{4}f(k)$, where $f(k)=\frac{(1-1/k)\sqrt{k(k-1)}+1/\sqrt{2}}{k-1+1/2k}\geq 1$. These results significantly improve on the previous approximation ratios for MHV and MHE in [10]. Our randomized rounding approach is motivated by the work of Kleinberg and Tardos [7] on the uniform Metric Labeling problem and the work of Langberg et al. [8] on the Multiway Uncut problem. However, our approximation ratio analyses require nontrivial extension. From a high-level viewpoint, the analyses of the randomized rounding scheme in [7,8] were performed in an edge-by-edge manner. In contrast, our analysis for the MHV problem considers a group of vertices at each time, and this extension essentially requires the structural properties of the MHV problem.

Since Multiway Uncut is a special case of MHE, the above results also means that the Multiway Uncut problem can be approximated within $\frac{1}{2}+\frac{\sqrt{2}}{4}f(k)$. For fixed values of k, this ratio improves upon the result 0.8535 in [8]. For example, when $k=3,4,5$ and 10, our ratios are 0.8818, 0.8739, 0.8694, and 0.8611, respectively. We get this improvement because we unite a simple randomized algorithm and the randomized rounding procedure in [8]. When k approaches infinity, the ratio tends to $\frac{1}{2}+\frac{\sqrt{2}}{4}=0.8535\cdots$, coinciding with the ratio given in [8].

Notations. Throughout the paper, we use OPT to denote the optimal value of an optimization problem and OPT_f to denote the optimal value of the corresponding fractional problem (when linear programming is involved).

2 Algorithms for MHV

Let Δ be the maximum vertex degree in the input graph. In this section, we show that the MHV problem can be approximated within $\frac{1}{\Delta+1}$ in polynomial time by randomized LP-rounding. Li and Zhang [10] gave a simple greedy $\frac{1}{k}$-approximation algorithm for the MHV problem, which is briefly shown as Algorithm $\mathcal{G}$ below. Therefore, to achieve the $\frac{1}{\Delta+1}$-approximation, we can safely assume $k\geq\Delta+1$.

Algorithm $\mathcal{G}$.

1 Just color all the uncolored vertices in the same color. Since there are k colors, we obtain k vertex colorings for graph G.

2 Output the coloring that has the largest total weight of happy vertices.

Lemma 1. *Without loss of generality, we may assume that* $k\geq\Delta+1$. □

The linear programming relaxation for the MHV problem is shown as (LP-V) below. To explain that (LP-V) is really an LP-relaxation for MHV, consider its corresponding integer linear program. That is, the variable constraint is replaced by the constraint $x_v, x_v^i, y_v^i \in \{0,1\}, \forall i, \forall v$.

Here, variable y_v^i indicates whether vertex v is colored in i, x_v^i indicates whether v is happy by color i, and x_v indicates whether v is happy. Constraint (2) is concerned with vertices that have pre-specified colors. Then, constraint (1) says that each vertex must be colored and can have only one color. In constraint (3), the notation $B(v)$ means the *ball* centered at vertex v, *i.e.*, the set of vertex v itself and all its neighbors. By constraint (3), vertex v is happy by color i only when all the vertices in $B(v)$ are colored in i. Note that constraint (3) can be replaced by the linear constraint $x_v^i \le y_u^i, \forall i, \forall v, \forall u \in B(v)$.

$$
\begin{aligned}
\max \quad & \sum_{v\in V} w_v x_v && && \text{(LP-V)}\\
\text{s.t.} \quad & \sum_i y_v^i = 1, && \forall v && (1)\\
& y_v^i = 1, && \forall i, \forall v \text{ s.t. } c(v) = i && (2)\\
& x_v^i = \min_{u\in B(v)} \{y_u^i\}, && \forall i, \forall v && (3)\\
& x_v = \sum_i x_v^i, && \forall v && (4)\\
& x_v, x_v^i, y_v^i \ge 0, && \forall i, \forall v &&
\end{aligned}
$$

We mention that in the integer version of (LP-V), any vertex can be happy in only one color. For otherwise suppose for some vertex v, we have $x_v^{i_1} = 1$ and $x_v^{i_2} = 1$, where $i_1 \neq i_2$. Then by constraint (3), we have $y_u^{i_1} = 1$ and $y_u^{i_2} = 1$ for any vertex $u \in B(v)$, but this immediately contradicts constraint (1). The similar property holds for LP-relaxation (LP-V), as shown in the following Lemma 2.

Lemma 2. *In any feasible solution to (LP-V), we have* $0 \le x_v \le 1$ *for any vertex* v.

Proof. Suppose for some vertex v we have $x_v = \sum_{i=1}^k x_v^i > 1$. Take any vertex $u' \in B(v)$. For this vertex u', by constraint (3), we have $\sum_i y_{u'}^i \ge \sum_i \min_{u\in B(v)}\{y_u^i\} = \sum_i x_v^i > 1$. This is in contradiction with constraint (1). □

The straightforward strategy that colors vertex u by color i with probability y_u^i would yield an integral solution with poor approximation ratio. Instead, we use the rounding technique proposed by Kleinberg and Tardos [7] to round a fractional solution to (LP-V). The algorithm is shown as Algorithm $\mathcal{R}$. In step 3 of the algorithm, the notation $[k]$ denotes the set $\{1, 2, \cdots, k\}$.

Algorithm $\mathcal{R}$.

```
1 Solve (LP-V) to obtain an optimal solution $(x, y)$.
2 while there exists some uncolored vertex do
```

```
3      Pick a color $i \in [k]$ uniformly at random.
4      Pick a parameter $\rho \in [0, 1]$ uniformly at random.
5      For each uncolored vertex $v$, if $y_v^i \geq \rho$, then color $v$ in $i$.
6  endwhile
```

Langberg et al. [8] used the same rounding technique as in Algorithm $\mathcal{R}$ for the Multiway Uncut problem. We adopt the high-level idea of the analyses in [7] and [8]. However, both the analyses in [7] and [8] for the randomized rounding scheme were performed in an edge-by-edge manner. We non-trivially extend the analyses of [7,8] to Algorithm $\mathcal{R}$ for the MHV problem below. In contrast, our analysis considers a group of vertices at each time, and this extension essentially requires the structural properties of the MHV problem. We begin with a simple assumption.

Lemma 3. *Without loss of generality, we may assume that $|B(v)| \geq 2$ for every vertex v.*

Proof. By definition, if $|B(v)| = 1$, then v is an isolated vertex. In this case, either v is already happy (It has a pre-specified color), or v can become happy (It is uncolored. Then we just color v in any color). So, all the isolated vertices can be safely removed from the graph without affecting the analysis of approximation ratio of the algorithm $\mathcal{R}$. $\square$

In Algorithm $\mathcal{R}$, each execution of steps 3 to 5 is called a *round*. The algorithm may iterate steps 3 to 5 for many rounds. If a ball $B(v)$ contains no colored vertices, then $B(v)$ is called a *blank* ball. If a ball $B(v)$ contains only one color, then $B(v)$ is called a *monochrome* ball.

Lemma 4. *Consider a ball $B(v)$ which is blank at the beginning of some round. Then the probability that in this round all (uncolored) vertices in $B(v)$ are colored in the same color is $\frac{x_v}{k}$.*

Proof. Fix some color i. The probability that color i is picked is $\frac{1}{k}$. All vertices in $B(v)$ are colored in i if and only if $\rho \leq \min_{u \in B(v)}\{y_u^i\} = x_v^i$. So, conditioned on the event that color i is picked, the probability that all vertices in $B(v)$ is colored in i is x_v^i.

By the above analysis, the probability that all vertices in $B(v)$ are colored in the same color is $\sum_i \frac{1}{k} x_v^i = \frac{x_v}{k}$. $\square$

Remarks. Let $B(v)$ be a ball which is blank at the beginning of some round. It is easy to see that even if parts of the vertices in $B(v)$ are colored in this round, it is still possible that all vertices in $B(v)$ are colored in the same color in the course of the algorithm (due to the subsequent round(s)). This probability is beyond Lemma 4 and is omitted in the analysis of the algorithm. See the proof of Theorem 1. Of course, the approximation ratio proved in Theorem 1 holds even with this omission.

Lemma 5. *Consider a ball $B(v)$ which is blank at the beginning of some round. Then the probability that in this round some vertex in $B(v)$ is colored is $\leq \frac{1}{k}(\Delta + 1 - x_v)$.*

Proof. By step 5 of Algorithm $\mathcal{R}$, there is a vertex in $B(v)$ that will be colored in i by the current round, if and only if ρ falls in the interval $[0, \max_{u \in B(v)}\{y_u^i\}]$. So, the probability that in this round some vertex in $B(v)$ is colored is

$$\sum_i \frac{1}{k} \max_{u \in B(v)} \{y_u^i\} = \frac{1}{k} \sum_i \max_{u \in B(v)} \{y_u^i\}. \tag{5}$$

On the other hand, for any color i, we have $\min_{u \in B(v)}\{y_u^i\} + \max_{u \in B(v)}\{y_u^i\} \leq \sum_{u \in B(v)} y_u^i$, since $|B(v)| \geq 2$ by Lemma 3. This implies

$$\begin{aligned} &\sum_i \left(\min_{u \in B(v)} \{y_u^i\} + \max_{u \in B(v)} \{y_u^i\} \right) \\ \leq &\sum_i \sum_{u \in B(v)} y_u^i = \sum_{u \in B(v)} \sum_i y_u^i \underset{(1)}{=} |B(v)| \leq \Delta + 1. \end{aligned}$$

Therefore, we have

$$\sum_i \max_{u \in B(v)} \{y_u^i\} \leq \Delta + 1 - \sum_i \min_{u \in B(v)} \{y_u^i\} \underset{(3),(4)}{=} \Delta + 1 - x_v. \tag{6}$$

By (5) and (6), the probability that in this round some vertex in $B(v)$ is colored is $\leq \frac{1}{k}(\Delta + 1 - x_v)$. □

Lemma 6. *Consider a blank ball $B(v)$. The probability that all vertices in $B(v)$ are colored in the same round (and hence in the same color) is $\geq \frac{x_v}{\Delta+1-x_v}$.*

Proof. Let p be the probability defined in the lemma. Fix the r-th round ($r \geq 1$) in the execution course of the algorithm. Define $N_{<r}$ as the event that all vertices in $B(v)$ are not colored *before* the r-th round, and A_r the event that all vertices in $B(v)$ are colored in the r-th round. Since the random variables i and ρ in Algorithm $\mathcal{R}$ are chosen independently across rounds, we have

$$p = \sum_{r=1}^{\infty} \Pr[N_{<r}] \cdot \Pr[A_r | N_{<r}]. \tag{7}$$

We first calculate the probability $\Pr[N_{<r}]$. Let t be a round ($t \geq 1$). Define N_t as the event that all vertices in $B(v)$ are not colored *in* the t-th round. Then

$$\begin{aligned} \Pr[N_{<r}] &= \Pr[N_{<r-1}] \Pr[N_{r-1} | N_{<r-1}] \\ &= (\Pr[N_{<r-2}] \Pr[N_{r-1} | N_{<r-2}]) \Pr[N_{r-1} | N_{<r-1}] \\ &= \cdots \\ &= \Pr[N_{<1}] \Pr[N_1 | N_{<1}] \Pr[N_2 | N_{<2}] \Pr[N_3 | N_{<3}] \cdots \Pr[N_{r-1} | N_{<r-1}] \\ &= \prod_{t=1}^{r-1} \Pr[N_t | N_{<t}], \end{aligned} \tag{8}$$

where the last equality holds since, by the given condition in the lemma, $B(v)$ is a blank ball (at the beginning of the algorithm), and hence we have $\Pr[N_{<1}] = 1$. Note that we also have $\Pr[N_1|N_{<1}] = \Pr[N_1]$.

Define E_t as the event that there exists a vertex in $B(v)$ that is colored in the t-th round. Then we have $\Pr[N_t|N_{<t}] = 1 - \Pr[E_t|N_{<t}]$. By Lemma 5, we obtain $\Pr[N_t|N_{<t}] \geq 1 - \frac{\Delta+1-x_v}{k}$. Consequently, we get

$$\Pr[N_{<r}] \underset{(8)}{\geq} \left(1 - \frac{\Delta+1-x_v}{k}\right)^{r-1}. \tag{9}$$

By Lemma 4, we know

$$\Pr[A_r|N_{<r}] = \frac{x_v}{k}. \tag{10}$$

Now we can give our estimation of the probability p:

$$p \underset{(7),(9),(10)}{\geq} \sum_{r=1}^{\infty} \left(1 - \frac{\Delta+1-x_v}{k}\right)^{r-1} \cdot \frac{x_v}{k} = \frac{x_v}{\Delta+1-x_v}. \qquad \square$$

The following Lemma 7 is used to analyze the probability that a vertex v is happy whose ball $B(v)$ is a monochrome ball. While it is the counterpart of Lemma 6 for a vertex whose ball is a blank ball, we have to be careful for the analysis in Lemma 7, since in a monochrome ball, there is already a used color. Although the two proofs are similar, the proof of Lemma 7 is more complicated than that of Lemma 6. The proof of Lemma 7 is omitted here due to space limitation and will be given in the full version.

Lemma 7. *Consider a monochrome ball $B(v)$ with pre-specified color i^*. The probability that all the uncolored vertices in $B(v)$ are colored in i^* in the same round is $\geq \frac{x_v}{\Delta - x_v}$.*

Define $\mathcal{B}_0$ as the set of vertices whose $B(v)$'s are blank at the beginning of Algorithm $\mathcal{R}$, and $\mathcal{B}_1$ the set of vertices whose $B(v)$'s are monochrome at the beginning of Algorithm $\mathcal{R}$.

Lemma 8. *For vertex $v \notin \mathcal{B}_0 \cup \mathcal{B}_1$, we have $x_v = 0$.*

Proof. Since $v \notin \mathcal{B}_0 \cup \mathcal{B}_1$, by definition, there are at least two different pre-specified colors in $B(v)$. Suppose u_1 and u_2 are two vertices in $B(v)$ with two pre-specified colors i_1 and i_2, respectively. Without loss of generality, we can assume that $i_1 = 1$ and $i_2 = 2$.

For vertex u_1, we have $y_{u_1}^1 = 1$ and $y_{u_1}^2 = \cdots = y_{u_1}^k = 0$. So, for each $2 \leq i \leq k$, we have $\min_{u \in B(v)}\{y_u^i\} = 0$. Then consider color 1. Since $y_{u_2}^2 = 1$, we know $y_{u_2}^1 = 0$. Therefore, $\min_{u \in B(v)}\{y_u^1\} = 0$. This means $x_v = \sum_i x_v^i = 0$. $\square$

Theorem 1. *Algorithm $\mathcal{R}$ is a $\frac{1}{\Delta+1}$-approximation algorithm for MHV.*

Proof. Let random variable SOL represent the total weight of happy vertices found by Algorithm $\mathcal{R}$. Obviously, only vertices in $\mathcal{B}_0$ and $\mathcal{B}_1$ may be happy. So, we have

$$\begin{aligned}
\mathrm{E}[SOL] = &\sum_{v\in\mathcal{B}_0} w_v \Pr[B(v) \text{ is finally colored in only one color}] + \\
&\sum_{v\in\mathcal{B}_1} w_v \Pr[B(v) \text{ is finally colored in only one color}] \\
\geq &\sum_{v\in\mathcal{B}_0} \frac{w_v x_v}{\Delta+1-x_v} + \sum_{v\in\mathcal{B}_1} \frac{w_v x_v}{\Delta - x_v} \quad \text{(by Lemmas 6 and 7)} \\
\geq &\frac{1}{\Delta+1} \sum_{v\in\mathcal{B}_0\cup\mathcal{B}_1} w_v x_v \underset{\text{(LM8)}}{=} \frac{1}{\Delta+1}\sum_v w_v x_v \geq \frac{1}{\Delta+1} OPT,
\end{aligned}$$

where the first equality is due to the linearity of expectation. Note that at the first inequality, we omit the probability that all vertices in $B(v)$ are colored in the same color across rounds (see remarks after Lemma 4). □

Algorithm $\mathcal{R}$ for the MHV problem can be derandomized in polynomial time by the standard conditional expectation method. The derandomization details can be found in [7, Section5]. Hence, the MHV problem actually can be approximated deterministically within $\frac{1}{\Delta+1}$.

3 Algorithms for MHE

The following linear program (LP-E) is an LP-relaxation for the MHE problem. In the corresponding integer program of (LP-E), variable y_v^i indicates whether vertex v is colored in i, x_e^i indicates whether edge e is happy by color i (*i.e.*, its two endpoints are all colored in i), and x_e indicates whether edge e is happy.

$$\begin{aligned}
\max \quad & \sum_{e\in E} w_e x_e & & \text{(LP-E)} \\
\text{s.t.} \quad & \sum_i y_v^i = 1, & \forall v & \quad (11) \\
& y_v^i = 1, & \forall i, \forall v \text{ s.t. } c(v) = i & \quad (12) \\
& x_e^i = \min\{y_u^i, y_v^i\}, & \forall i, \forall e = (u,v) & \quad (13) \\
& x_e = \sum_i x_e^i, & \forall e & \\
& x_e, x_e^i, y_v^i \geq 0, & \forall i, \forall v, \forall e &
\end{aligned}$$

Constraint (12) describes the vertices that have pre-specified colors. Constraint (11) says that each vertex has exactly one color. Constraint (13) says that edge e is happy by color i only when both its two endpoints are colored in i. Note that constraint (13) is linear since it can be replaced by $x_e^i \leq y_u^i$

and $x_e^i \leq y_v^i$. Furthermore, by constraint (11), it is impossible for an edge to be simultaneously satisfied by two different colors. Therefore, (LP-E) is really an LP-relaxation for the MHE problem.

Let (x, y) be an optimal fractional solution to (LP-E). To obtain an integral solution, a straightforward LP-rounding technique is to color vertex v in i with probability y_v^i. We can show this strategy will generate an integral solution whose value could be as bad as $1/k$ times OPT_f(LP-E) (the fractional optimum of (LP-E)). This approximation is unsatisfactory, and thus we adopt the randomized rounding technique of Kleinberg and Tardos [7] again to round (x, y). The algorithm is essentially the same as Algorithm $\mathcal{R}$, except that in step 1 we solve (LP-E) instead of (LP-V). For simplicity, we still call the randomized rounding algorithm Algorithm $\mathcal{R}$.

Then Consider the following simple randomized algorithm, namely, Algorithm $\mathcal{P}$ for MHE.

```
Algorithm $\mathcal{P}$.
1 Pick a color $i \in [k]$ uniformly at random.
2 Color all the uncolored vertices in $i$.
```

The final algorithm for MHE is shown as Algorithm $\mathcal{A}$.

```
Algorithm $\mathcal{A}$.
1 With probability $\lambda$ run Algorithm $\mathcal{R}$, and with probability $1-\lambda$ run Algorithm $\mathcal{P}$.
2 Return the coloring found (by either $\mathcal{R}$ or $\mathcal{P}$) in step 1.
```

Let E_2 be the set of edges with both endpoints being given some pre-specified colors, E_1 the set of edges with only one endpoint being given a pre-specified color, and E_0 the set of edges with no endpoints that have pre-specified colors. Then (E_2, E_1, E_0) is a partition of the edge set E. The edges in E_2 can be further classified into two categories: happy edges and unhappy edges. Since Algorithm $\mathcal{A}$ can obtain all happy edges in E_2 (namely, the approximation ratio for this part of edges is 1), for simplicity of analysis, we may assume that there is no edges of E_2 in the input graph.

Let $e = (u, v)$ be an edge. Note that $0 \leq x_e = \sum_i x_e^i \leq \sum_i y_u^i = 1$. The following lemmas are known from [7,8].

Lemma 9 ([7,8]). *Let e be an edge in E_1. Then* $\Pr[e$ *is happy in* $\mathcal{R}] = x_e$.

Lemma 10 ([8]). *Let e be an edge in E_0. Then* $\Pr[e$ *is happy in* $\mathcal{R}] \geq \frac{x_e}{2-x_e}$.

Theorem 2. *The MHE problem can be approximated within $\frac{1}{2} + \frac{\sqrt{2}}{4} f(k)$, where k is the number of colors and $f(k) = \frac{(1-1/k)\sqrt{k(k-1)}+1/\sqrt{2}}{k-1+1/2k} \geq 1$.*

Proof. Let random variables R_1, P_1, A_1 be the total weight of happy edges in E_1 found by Algorithms $\mathcal{R}$, $\mathcal{P}$, and $\mathcal{A}$, respectively. Then, by Lemma 9, we have

$$\begin{aligned}
\mathrm{E}[A_1] &= (1-\lambda)\mathrm{E}[P_1] + \lambda \mathrm{E}[R_1] \\
&= (1-\lambda)\sum_{e\in E_1} w_e \Pr[e \text{ is happy in } \mathcal{P}] + \lambda \sum_{e\in E_1} w_e \Pr[e \text{ is happy in } \mathcal{R}] \\
&= (1-\lambda)\sum_{e\in E_1} \frac{w_e}{k} + \lambda \sum_{e\in E_1} w_e x_e = \sum_{e\in E_1}\left(\frac{1-\lambda}{k} w_e + \lambda w_e x_e\right) \\
&\geq \sum_{e\in E_1}\left(\frac{1-\lambda}{k} + \lambda\right) w_e x_e.
\end{aligned}$$

Let random variables R_0, P_0, A_0 be the total weight of happy edges in E_0 found by Algorithms $\mathcal{R}$, $\mathcal{P}$, and $\mathcal{A}$, respectively. Then, by Lemma 10, we have

$$\begin{aligned}
\mathrm{E}[A_0] &= (1-\lambda)\mathrm{E}[P_0] + \lambda \mathrm{E}[R_0] \\
&= (1-\lambda)\sum_{e\in E_0} w_e \Pr[e \text{ is happy in } \mathcal{P}] + \lambda \sum_{e\in E_0} w_e \Pr[e \text{ is happy in } \mathcal{R}] \\
&\geq (1-\lambda)\sum_{e\in E_0} w_e + \lambda \sum_{e\in E_0} \frac{w_e x_e}{2-x_e} = \sum_{e\in E_0}\left((1-\lambda) w_e + \frac{\lambda w_e x_e}{2-x_e}\right) \\
&\geq \sum_{e\in E_0, x_e>0}\left(\frac{1-\lambda}{x_e} + \frac{\lambda}{2-x_e}\right) w_e x_e \geq \sum_{e\in E_0, x_e>0}\left(\frac{1}{2} + \sqrt{\lambda(1-\lambda)}\right) w_e x_e \\
&= \sum_{e\in E_0}\left(\frac{1}{2} + \sqrt{\lambda(1-\lambda)}\right) w_e x_e,
\end{aligned}$$

where the last inequality holds since $\min_{x_e\in[0,1]}\{\frac{1-\lambda}{x_e} + \frac{\lambda}{2-x_e}\} \geq \frac{1}{2} + \sqrt{\lambda(1-\lambda)}$.

Let random variable A be the total weight of happy edges found by Algorithm $\mathcal{A}$. Therefore, we get

$$\begin{aligned}
\mathrm{E}[A] &= \mathrm{E}[A_1] + \mathrm{E}[A_0] \\
&\geq \sum_{e\in E_1}\left(\frac{1-\lambda}{k} + \lambda\right) w_e x_e + \sum_{e\in E_0}\left(\frac{1}{2} + \sqrt{\lambda(1-\lambda)}\right) w_e x_e \\
&\geq \min\left\{\frac{1-\lambda}{k} + \lambda, \frac{1}{2} + \sqrt{\lambda(1-\lambda)}\right\} \sum_{e\in E} w_e x_e.
\end{aligned}$$

Solving $\frac{1-\lambda}{k} + \lambda = \frac{1}{2} + \sqrt{\lambda(1-\lambda)}$, we get $\lambda = \frac{2k^2+2-3k+\sqrt{2k^4-2k^3}}{2(2k^2-2k+1)}$. (The other root of λ cannot lead to good approximation ratio and is omitted.) For our choice of λ, its value is between $0.8227\cdots$ and $0.8535\cdots$ when $k \geq 3$. (The problem is polynomial time solvable when $k = 2$). With this value of λ, the approximation ratio is $\frac{1}{2} + \frac{\sqrt{2}}{4} f(k)$, where $f(k) = \frac{(1-1/k)\sqrt{k(k-1)}+1/\sqrt{2}}{k-1+1/2k} \geq 1$. □

Algorithms $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{R}$ can all be derandomized in polynomial time by the conditional expectation method. So the result in Theorem 2 is a deterministic result.

Since Multiway Uncut is a special case of MHE, Theorem 2 also means that the Multiway Uncut problem can be approximated within $\frac{1}{2} + \frac{\sqrt{2}}{4} f(k)$. For fixed values of k, this ratio improves upon the ratio 0.8535 of [8]. For example, when $k = 3, 4, 5$, and 10, our ratios are 0.8818, 0.8739, 0.8694, and 0.8611, respectively. When k approaches infinity, the ratio tends to $\frac{1}{2} + \frac{\sqrt{2}}{4} = 0.8535 \cdots$, coinciding with the ratio given in [8]. We get this improvement because we make use of the randomized Algorithm $\mathcal{P}$, while [8] does not.

Acknowledgments. Peng Zhang is supported by the State Scholarship Fund of China, Natural Science Foundation of Shandong Province (ZR2012FZ002 and ZR2013FM030), and the Fundamental Research Funds of Shandong University (2015JC006).

Angsheng Li is supported by the hundred talent program of the Chinese Academy of Sciences, and the grand challenge program, *Network Algorithms and Digital Information*, Institute of Software, Chinese Academy of Sciences.

References

1. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA, USA (1984)
2. Buchbinder, N., Naor, J., Schwartz, R.: Simplex partitioning via exponential clocks and the multiway cut problem. In: Proc. STOC, pp. 535–544 (2013)
3. Calinescu, G., Karloff, H., Rabani, Y.: An improved approximation algorithm for multiway cut. Journal of Computer and System Sciences **60**(3), 564–574 (2000)
4. Dahlhaus, E., Johnson, D., Papadimitriou, C., Seymour, P., Yannakakis, M.: The complexity of multiterminal cuts. SIAM Journal on Computing **23**, 864–894 (1994)
5. Easley, D., Kleinberg, J.: Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press (2010)
6. Karger, D., Klein, P., Stein, C., Thorup, M., Young, N.: Rounding algorithms for a geometric embedding of minimum multiway cut. Mathematics of Operations Research **29**(3), 436–461 (2004)
7. Kleinberg, J., Tardos, É.: Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. Journal of the ACM **49**(5), 616–639 (2002)
8. Langberg, M., Rabani, Y., Swamy, C.: Approximation algorithms for graph homomorphism problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 176–187. Springer, Heidelberg (2006)
9. Li, A., Li, J., Pan, Y.: Homophyly/kinship hypothesis: natural communities, and predicting in networks. Physica A **420**, 148–163 (2015)
10. Li, A., Zhang, P.: Algorithmic aspects of homophyly of networks. Manuscript (2012). arXiv:1207.0316
11. Sharma, A., Vondrák, J.: Multiway cut, pairwise realizable distributions, and descending thresholds. In: Proc. STOC, pp. 724–733 (2014)

An Approximation Algorithm for the Smallest Color-Spanning Circle Problem

Yin Wang[1,2](✉) and Yinfeng Xu[1,2]

[1] School of Management, Xi'an Jiaotong University, Xi'an 710049, China
[2] The State Key Lab for Manufacturing Systems Engineering, Xi'an 710049, China
yinaywang@stu.xjtu.edu.cn, yfxu@mail.xjtu.edu.cn

Abstract. To find a minimum radius circle in which at least one point of each color lies inside, we have researched the smallest enclosing circle problem for n points with m different colors. The former research proposed a π-approximation algorithm with a running time $O(n^2 + nm \log m)$. In this paper, we construct a color-spanning set for each point and find the smallest enclosing circle to cover all points of each color-spanning set. The approach to find each color-spanning set is based on the nearest neighbor points which have different colors. An approximation algorithm to compute the minimum diameter of the enclosing circle is proposed with the time of $O(nm \log n + n \log m)$ at most. The approximation ratio of our algorithm is less than 2. In conclusion, both approximation ratio and complexity are improved by our proposed algorithm.

Keywords: Computational geometry · Colored set · Approximation algorithm · The minimum diameter color-spanning set problem

1 Introduction

The motivation for the geometric facility location problem comes from the base site selection problem. Based on the arrangement of no different points, a couple of related optimization problems for a set s of n points have already been studied in the literature. Dobkin et al.[1] and Overmas et al.[2] treated the problems of finding minimum perimeter convex k-gons. Their $O(k^2 n \log n + k^5 n)$ Alogrithm result was inmproved to $O(n \log n + k^5 n)$ by Agarwal et al. [3]. Under the condition of $k \leq \frac{n}{2}$, Eppstein et al. [4] studied the smallest ploytopes and proposed a $O(n^2 \log n)$ time algorithms using the Farthest color Voronoi diagram.

Depending on the practical requirements of different applications, variations of problems have been proposed. As associated with different categories, all points are marked by different colors. However, sometimes the information like the locations are not known exactly (Beresfor et al. [5], Cheng et al. [6]). A continuous region which can be a disc model is proposed in difference papers (Wenqi Ju et al. [7]). Then, it is always the objective to find the boundary of a geometrical model to cover all properties.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 171–182, 2015.
DOI: 10.1007/978-3-319-21398-9_14

Abellanas et al.[8] showed an algorithm for smallest color-spanning objects of axis-parallel rectangle and the narrowest strip. Two constant factor approximation algorithms were proposed for the minimum perimeter of the convex hull by Wenqi Ju et al. [7]. And their algorithms cost time of $O(n^2 + nm \log m)$ and O$(\min\{n(n-m)^2, nm\,(n-m)\})$ with the ratio of π and $\sqrt{2}$ respectively. However, as well as some other variants of theses problems are based on the smallest plogons but Morton et al.[9] proved that such k-gons might not exist for $k > 7$.

However, the former research mostly focus on computing the smallest covering polygons with colored vertices. If the colored points are in general position, it is proved to be NP-hard for some problems (Fleischer and Xu [10], Wenqi Ju et al.[7]). The versions of constrained circle were proposed (Kamrmakar et al. [11]) and solved by the polynomial time algorithms. Zhang et al. [12] showed a brute force algorithm with $O(n^k)$ time for the minimum diameter color-spanning set problem(MDCS).

The problems above are often referred to as the color-spanning set problem (CSSP). Each imprecise point is modelled by a set of k points with m kind of given colors. Each point only has one of the m given colors and the possible location of each point is already known. The color-spanning set contain one point in each color set at least and its number of points is not less than m $(n \geq k \geq m)$. In this paper, we study the smallest color-spanning circle in general position and without the constrained of center and color numbers.

For multicolored point sets, there are solutions to several problems, such as the bichromatic closest, see e.g. Agarwal et al. [13]. Interestingly, the approach in paper of Eppstein et al. [14] for the smallest polytopes, like the group Steiner tree by Graf et al. [15], is also based on the nearest neighbors. In this paper, for each point, we cosider the nearest neighbor point which has different color firstly. Then we can find a color-spanning set based on these nearest neighbors. For every color-spanning set, we can find a circle to cover all points of the set. Our purpose is to find the minimum diameter of these circles.

2 Problem Statement and Notations

2.1 Problem Statement

In the color-spanning set problem, each point in the set $P = \{p_1, p_2, ..., p_n\}$ is associated with a color from a set of m colors ($m \leq n$). Let S_i be the set of points in the color class i $(i = 1, 2, ..., m)$. We use i to denote the color of the point p_{ij} and j to lable the differentiate points in S_i which have different position in the plane. A new point set named the color-spanning set is constituted such that at least one point of each color lies inside it.

Without loss of generality, we make the following assumption on general position: (1) each point has and only has one of the m given colors and there is no horizontal or vertical line passing through two or more points. (2) the number of colors is fixed. (3) considering the concision, the Euclidean distance between two points is supposed to be known as its position information, but the exact position of each point is uncertain.

Definition 1. We call a circle color-spanning if all points in this circle contains at least one of each color set S_i. The objective is to find the minimum diameter circle c_s to cover a color-spanning set.

Meanwhile, this circle should meet the following conditions: (1) covering all the points in color-spanning set, (2) having the minimum diameter, (3) containing all m given colors. From all above, we can define this problem to find the minimum diameter of this circle as the smallest color-spanning circle (SCSC) problem.

2.2 Notations

The two properties of every point are color and position. j is used to mark each point with unique position in the plane, since the exact position is unknown. Let p_{ij} denote the point with the color of i when its position is j, for $1 \leq i \leq m$ and $1 \leq j < n$.

Definition 2. Let V_{ij} denote a point set of all points in each color-spanning circle. For each point p_{ij}, we can constitute a color-spanning set V_{ij} with $m-1$ different colors from p_{ij}. Then we have the point set

$$V_{ij} = \{p_{ij}, p_1, \ldots, p_k, \ldots, p_{m-2}, p_{m-1}\}.$$

We assume the number of points in each V_{ij} is not less than m. The rank of points in each V_{ij} is denoted as k $(1 \leq k \leq m)$. Let each p_{ij} be the first one of V_{ij} and called as initial point. The distance between p_{ij} and other points of V_{ij} decides their rank k in each V_{ij}.

The points in each color-spanning set V_{ij} should satisfy two properties: different colors and shortest interval with p_{ij}. Satisfied all the property above, the point set can be regard as the color-spanning set V_{ij} of each point p_{ij}.

Definition 3. The intervals of p_{ij} and other points in V_{ij} constitute a sequence I_{ij}. Similarly, let the sequence I_{ij} as $I_{ij} = \{d_1, d_2, \ldots, d_{m-2}, d_{m-1}\}$. We denote a set of the nearest distinct color distance as I_{ij}.

An approach to find the candidates of a color-spanning set is searching the distance sequence I_{ij}. That is to say, the furthest distance of the nearest distinct color neighbors to p_{ij} is the maximum one in the sequence I_{ij} and denoted as follows: $d_{ij \to m-1} = \max\{I_{ij}\}$. With the hypothesis of its uniqueness, $d_{ij \to m-1}$ is accoresponding to the endpoint p_{m-1} in V_{ij}. By the analogy, $d_{m-2 \to ij}$ is the second furthest one. For the further computing, d_{ij} stands for the solution to the diameter of SCSC problem.

3 An Algorithm for the Color-Spanning Sets

Our aim is to cover all m given colors using the smallest circle. We conside one of the n points briefly. By that analogy, the others can be computed by the same way. Finally, the smallest one of these n color-spanning circles is our solution.

According to the definitions above, we can deduce that the color points in each color-spanning set should satisfy two constraints: closest to initial point p_{ij} in each color set s_i and cover all m colors.

As we know, V_{ij} is the color-spanning set of p_{ij}. The purpose of this paper is to find the smallest color-spanning circle. We present an algorithm to find the nearest distinct colored points of each point p_{ij}. As this purpose, the approach is based on the nearest distinct color distance I_{ij} for each p_{ij}. After finding the furthest distinct colored distance for each given point, the color-spanning set V_{ij} can be constituted. The processes are shown in Algorithm 1.

Algorithm 1. Contitute a color-spanning set for each p_{ij}

```
   For each initial point $p_{ij}$:
1: **for all** color sets except the one $p_{ij}$ belong to **do**
2:     **for all** points in each color set $S_x$ except the one $p_{ij}$ belong to **do** find the
   nearest point to initial point $p_{ij}$ in each color set
3:         **if** a point is the nearest one to $p_{ij}$ in color set $s_x$ **then**
4:             save this point as $p_x$ and add it to point set $V_{ij}$.
5:         **end if**
6:     **end for**
7: **end for**
```

For n points with m colors, the Algorithm 1 above can constitute a color-spanning point set for each p_{ij}.

Definition 4. With each I_{ij}, we can get the rainbow circles of this color-spanning set. The rainbow circle set of each color-spanning set V_{ij} is defined as a group of concentric circles. Seen in Fig.1 . Thus, we have the following result.

The rainbow circles set $C_{ij}(k, d_k) = \{c_1, c_2, \ldots, c_k, \ldots, c_{m-2}, c_{m-1}\}$ is a set of $m-1$ concentric circles with the center on p_{ij}. The radius of rainbow circle c_k is depended on the nearest distinct color distance d_k in each I_{ij}. For each p_{ij}, every colored circle c_k has two propertie as we proposed by Lemma 1 and Lemma 2 as below.

Lemma 1. There is k colors covered by the colored circle c_k.

Proof. The center of these circles is the initial point p_{ij}. The radiu of c_k is equal to $d_{ij \to m-1}$, which is belong to I_{ij}. If the distance between any point and p_{ij} is not longer than the radiu of c_k, the color of this point is covered by c_k. According to Algorithm 1, there is k points contain this condition. Besides, all these points have different colors as the definition of V_{ij}. □

We can know the c_{m-1} and c_{m-2} are the largest and second largest circle in each set of rainbow circles from definition 4. According to Lemma 1, the circle c_{m-1} of each p_{ij} contains all given m colors.

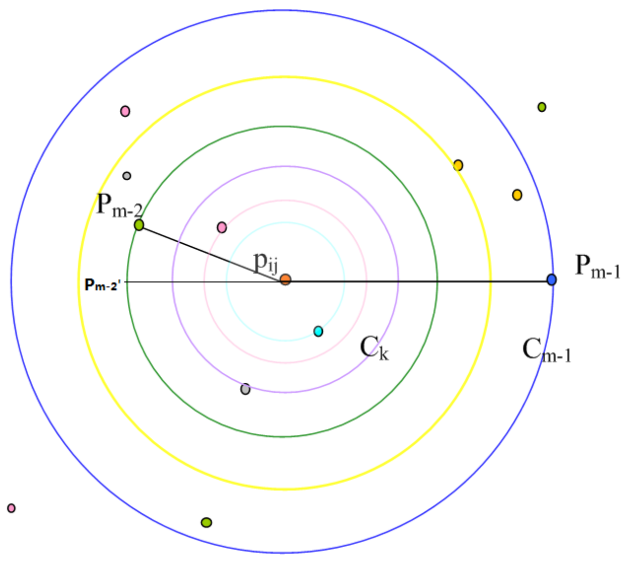


Fig. 1. The rainbow circles for each p_{ij}

Lemma 2. The point p_{m-1} is the only one with the color of circle c_{m-1}.

Proof. Let the color of circle $c_m - 1$ be x. If there is any point with color x inside the rainbow circle c_{m-1}, the point p_{m-1} cannot be the nearest to p_{ij} with color x. □

Theorem 1. The complexity for constituting each color-spanning set V_{ij} is $O(m \log n)$.

Proof. Each point can be the initial point p_{ij} of n possible color-spanning sets and its distances to other points are supposed to be known. Otherwise, we need time $O(\log n_x)$ to find the nearest point to p_{ij} in each color set S_i , where n_x is the number of points in color set S_x. As we know, there are n points in the plane. Then,

$$\sum_{x=1}^{m-1} n_x + n_i = n$$

the n_i is the number of points in S_i which is the color set the p_{ij} belong to. Therefore, it spends $O(\log n)$ to comupte the nearest distance to p_{ij} in each color set at most. For $m-1$ color set except the one p_{ij} belong to, it cost time of $O(m \log n)$ to constitute each V_{ij} at most.

□

4 An Approximation Algorithm for SCSC Problem

4.1 An Approximation Algorithm

For each initial point, according to the Algorithm 1 above, each V_{ij} can be consisted by the nearest neighbor points to p_{ij} with m given different colors. Based on the conclusions above, we come up with an approximation algorithm for the SCSC problem. The first and quite simple idea to do is computing d_{m-1} and d_{m-2} of each V_{ij}. The procedure is shown by Algorithm 2.

Algorithm 2. An Approximation Algorithm

```
    For given $n$ point
 1: for all initial point $p_{ij}$ do
 2:     for all points in $V_{ij}$ do
 3:         save the furthest one to $p_{ij}$ as $p_{m-1}$ and define their interval as $d_{m-1}$ ;
 4:     end for
 5:     for all points in $V_{ij}$ except $p_{m-1}$ do
 6:         save the furthest one to $p_{ij}$ as $p_{m-2}$ and define their interval as $d_{m-2}$ ;
 7:     end for
 8:     for all $p_{m-1}$ and $p_{m-2}$ of $p_{ij}$ do
 9:         compute $d_{ij} = d_{m-1} + d_{m-2}$
10:     end for
11: end for
12: for all $d_{ij}$ do
13:     compute the minimal one and save as $d_s = \min\{d_{m-1} + d_{m-2}\}$
14: end for
```

The d_{ij} is the sum of d_{m-1} and d_{m-2} of each V_{ij} in this Approximation Algorithm. We can draw a circle c_s with diameter of the minimum value of d_{ij}. The diameter of c_s is the approximation solution and denoted as d_s. Therefore the following assertion holds.

Definition 5. A union set contains p_{m-1} and all points in circle c_{m-2}. We define it as A for each V_{ij}, where $A \subseteq V_{ij}$.

We draw a circle denoted as c_{ij} with the diameter of d_{ij} for each V_{ij}. Let the line L_{ij} through the initial point p_{ij} and p_{m-1}. For each V_{ij}, the projection of each point p_k on L_{ij} is noted as p_k'. Find a point On L_{ij} saved as p_{m-2}', which is on the left of p_{ij} and has the same interval with p_{m-2} to p_{ij}. Each candidate c_{ij} of the approximation circle should satisfy properties as follows. Firstly, it can cover all given colors. Secondly, the center of this circle is at the middle of p_{m-2}' and p_{m-1}. Finally, the diameter d_{ij} of c_{ij} is equal to the sum of d_{m-1} and d_{m-2}.

Lemma 3. Each c_{ij} can cover all given m colors.

Proof. As we know, c_{ij} contains all points in the union set A above. Seen in Fig. 2. Furthermore, the points in A contain all points the circle c_{m-2} for each p_{ij}. Therefore, the $m-1$ colors of points in c_{m-2} are also covered by A. Furthermore, the color of p_{m-1} is unique in V_{ij} and it is contain by A too. As the result,A can cover all m given colors, so does c_{ij}. □

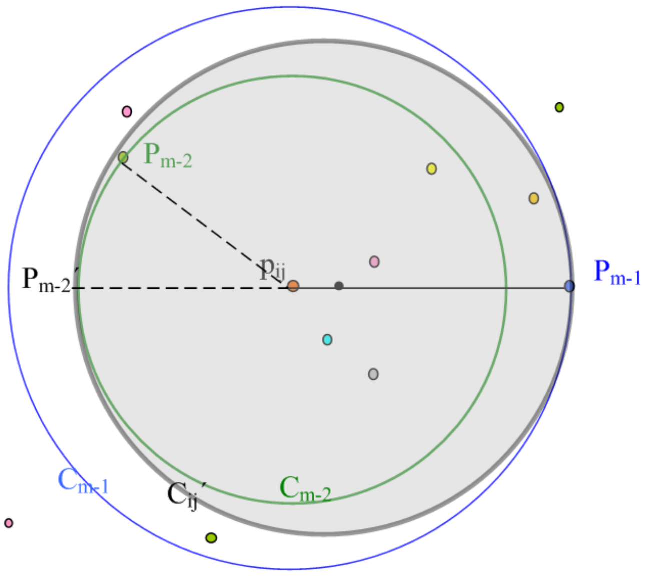


Fig. 2. The color-spanning circle c_{ij} of each p_{ij}

4.2 The Upper Bound and Lower Bound of the SCSC Problem

According to the conclusions above, we know each c_{ij} can contain all given m colors. In each c_{ij}, p_{m-1} and $p_{m-2}{}'$ is always on the boundary. Shown in the Fig.2, we can find the position of each point in A desides the optimal circle. The exact position is uncertainty, while distances between each two points are certain in this paper. Therefore, three cases of the possible position of all point in A are existed. We donot think about the region outside the rainbow circle c_{m-1} for each V_{ij}.

Case 1: The projection on L_{ij} of p_{m-2} in on the left of p_{ij}, just like the Fig 2. Let $p_{m-2}{}''$ be the projection of p_{m-2}. The radius of c_{ij}, which is saved as d_{ij}, should be equal to the distance of $p_{m-2}{}''$ and its center o_{ij}. It can be show as :

$$r_{ij} = d_{ij}/2 = |o_{ij}p_{m-2}|$$

And if the p_{m-2} should not be out of circle c_{ij}, then

$$d_{m-2} = |p_{ij}p_{m-2}| \geq |p_{ij}p_{m-2}{}''|\,.$$

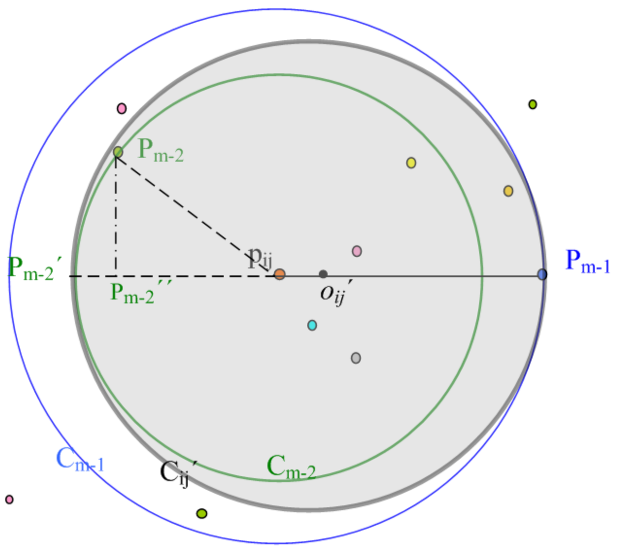


Fig. 3. Case 1

If and only if the equality hold up, we can get the minimum of the radius. That means,

$$r_{ij} = |o_{ij}p_{m-2}{''}| = |o_{ij}p_{m-2}|$$

Therefore, when the position of p_{m-2} is overlap with the point p_{ij}, we can find the worst case of the circle c_{ij}. The diameter of the smallest c_{ij} is the minimal value under the worst case.

Case 2: Denote the p_k as any point in A.c between p_{ij} and p_{m-1}. As $d_{m-1} \geq d_{m-2}$, the minimum of this radius should not be less than d_{m-1}. The optimal diameter is the minimal value of d_{m-1} and save as d_t. Seen in Fig. 4.

Case 3: When not all points in A satify

$${d_{p_{m-1}p_k}}^2 \leq {d_{p_{ij}p_{m-1}}}^2 - {d_{p_{ij}p_k}}^2$$

and their projections is not on the right of p_{ij} but the point p_{m-2}. Seen in Fig. 5.

The argument for case 3 is quite simlar to the proof of case 1. If the diameter of circle is less than d_{m-1}, it is certain that the points whose projection is on the right of p_{ij} cannot be covered by this circle. Therefore, the optimal diameter in cases should be more than d_t in case 2 but less than the d_s.

The maximum value of the four cases is the finally worst case of each color-spanning set of p_{ij}. As the result, we can prove the worst case is the situation where p_{m-2}, p_{ij} and p_{m-1} are on one line L_{ij}.

Lemma 4. The lower bound of the SCSC problem is the circle with the diameter d_t, where $d_t = \min\{d_{m-1}\}$.

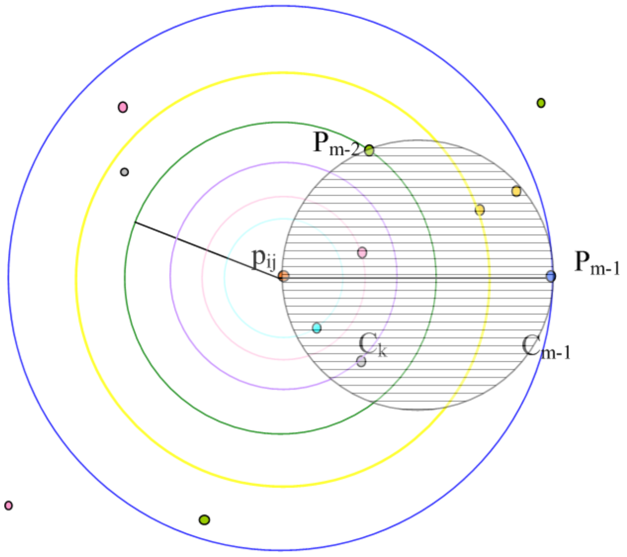


Fig. 4. Case 2

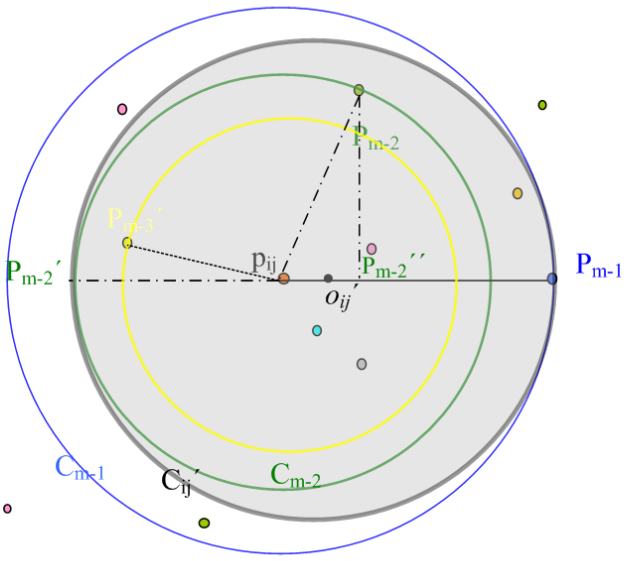


Fig. 5. Case 3

Proof. As the conclusion of the lemma 4, we can find the minimum possible radius in the case 2, which is shown as Fig 4. As the case of $\min\{d_{m-1}\}$, the point p_{ij} and p_{m-1} in c_t cannot be out of this optimal circle. Moreover, $|p_{ij}p_{m-2}|$ is the diameter of the circle. Chords cannot be longer than diameter in one circle. So the diameter d_{opt} of color-spanning circle cannot be less than the interval between p_{ij} and p_{m-1} of c_t. The length of minimal value of d_{m-1} is the lower bound of the diameter of color-spanning circle. Denote it as d_t, then $d_t = \min\{d_{m-1}\}$. □

Lemma 5. The optimal solution of SCSC problem cannot be less than the minimum of the furthest distinct colored distances d_{m-1}, as $d_{opt} \geq \min\{d_{m-1}\}$.

Proof. According to the argument of case 1, the upper bound of the optimal solution depends on the minimal worst case. That solution cannot be less than the approximation solution $r_s = \left\{ \frac{d_{m-1}+d_{m-2}}{2} \middle| \min\{d_{m-1}\} \right\}$. □

From the argument above, Theorem 2 can be concluded.

Theorem 2. The upper bound of the SCSC problem is the smallest c_{ij}. The low bound of the SCSC problem is the circle with the diameter of the minimum d_{m-1}.

4.3 The Approximation Ratio

Theorem 3. The approximation ratio $r(A)$ is the value between 1 and 2, $r(A) \in [1, 2]$.

Proof. According to the analysis of case 1 above, when the p_{m-2} and p_{m-2}' in circle c_s is overlap, $r(A) = 1$. Moreover, in case 2, if a circle with radius of $\frac{d_{m-1}}{2}$ is satisfied center at middle of p_{ij} and p_{m-1}, all points of V_{ij} can be covered. As a result, $r(A) = 2$.Therefore, we can know the approximation ratio is between 1 and 2 and depends on the ratio of d_{m-1} and d_{m-2} of the circle r_s. We compute the approximation ratio $r(A)$ as follows.

$$r(A) = \sup_I \frac{A(I)}{opt(I)}$$

As the argument of case 1 and case 2, we have the conclusion as below.

$$r(A) = \frac{\min\{d_{m-1} + d_{m-2}\}}{\min\{d_{m-1}\}}$$

Because d_s is the minimum value of d_{ij}, the sum of $\min\{d_{m-1}\}$ and $d_{m-2}\,|_{\min\{d_{m-1}\}}$ can not more than d_s. $d_{m-2}\,|_{\min\{d_{m-1}\}}$ means the d_{m-2} in the circle c_t. Then,

$$r(A) \leq \frac{\min\{d_{m-1}\} + d_{m-2}\,|_{\min\{d_{m-1}\}}}{\min\{d_{m-1}\}} = 1 + \frac{d_{m-1}}{d_{m-1}}\,|_{\min\{d_{m-1}\}}$$

As d_{m-2} is not alway more than d_{m-1} in each V_{ij}, we have the conclusion:

$$1 \leq r\,(A) \leq 2.$$

The ratio of our approximation algorithm can be proved.

□

4.4 The Complexity of the Approximation Algorithm

Theorem 4. The time complexity for computing the approximation solution of the minimum circle color-spanning set is $O(nm \log n + n \log m)$.

Proof. According to Algorithm 1, each initial point runs less than $O(m \log n)$ time to find the point set V_{ij}. The step 1 and step 2 cost the same time of $O(\log m)$ to find d_{m-1} and d_{m-2} of each V_{ij}. However, the number of initial point p_{ij} is n, such that we can get n color-spanning sets V_{ij}. As a result, traversal of all initial points still need n times. Furthermore, it spend $O(\log n)$ to find the smallest one. Then, we know

$$O(n) = T(n(m \log n + 2 \log m) + \log n) = O(nm \log n + n \log m).$$

As the result, the time complexity of this approximation algorithm is $O(nm \log n + n \log m)$. □

Conclusion

Given a set of n points, which is a unit set of m colored sets, we study the smallest circle to cover at least one of each colored set. Based on the worst case analysis, we present an algorithm to find a color-spanning set for every point. Based on the theory of geometric properties, the upper bound and lower bound of the SCSC problem are computed. The proposed approximation algorithm has an improved ratio (less than) 2. If the distance of each two points is known, the running time of the approximation algorithm is improved to O(nm log n + n log m).

Acknowledgments. The authors would like to acknowledge the financial support of Grants (No.61221063) from NSF of China and (No.IRT1173) from PCSIRT of China.

References

1. Dobkin, D.P., Drysdale, R.L., Guibas, L.J.: Finding smallest polygons. Computational Geometry **1**, 181–214 (1983)
2. Overmars, M.H., Rote, G., Woeginger, G.: Finding minimum area k-gons. Utrecht University, Department of Computer Science (1989)
3. Aggarwal, A., Imai, H., Katoh, N., et al.: Finding k points with minimum spanning trees and related problems. In: Proceedings of the Fifth Annual Symposium on Computational Geometry, pp. 283–291. ACM (1989)

4. Eppstein, D., Overmas, M.H., Rote, G., Woeginger, G.: Finding minimum area k-gons. Discrete Comput. Geom. **7**, 45–58 (1992)
5. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive computing **2**(1), 46–55 (2003)
6. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1112–1127 (2004)
7. Ju, W., Fan, C., Luo, J., et al.: On some geometric problems of color-spanning sets. Journal of Combinatorial Optimization **26**(2), 266–283 (2013)
8. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: Smallest color-spanning objects. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 278–289. Springer, Heidelberg (2001)
9. Morton, J.D., et al.: Sets with no empty convex 7-gons. Canadian Mathematical Bulletin **26**(4), 482 (1983)
10. Fleischer, R., Xu, X.: Computing minimum diameter color-spanning sets. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) FAW 2010. LNCS, vol. 6213, pp. 285–292. Springer, Heidelberg (2010)
11. Karmakar, A., Roy, S., Das, S.: Fast computation of smallest enclosing circle with center on a query line segment. Information Processing Letters **108**(6), 343–346 (2008)
12. Zhang, D., et al.: Keyword search in spatial databases: towards searching by document. In: IEEE 25th International Conference on Data Engineering, ICDE 2009, pp. 688–699 (2009)
13. Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O., et al.: Euclidean minimum spanning trees and bichromatic closest pairs. Discrete and Computational Geometry **6**(1), 407–422 (1991)
14. Eppstein, D., Erickson, J.: Iterated nearest neighbors and finding minimal polytopes. Discrete and Computational Geometry **11**(1), 321–350 (1994)
15. Graf, T., Hinrichs, K.: Algorithms for proximity problems on colored point sets. Universitt Mnster, Angewandte Mathematik und Informatik (1992)

Approximation Algorithms for the Connected Sensor Cover Problem

Lingxiao Huang, Jian Li, and Qicai Shi(✉)

Institute for Interdisciplinary Information Sciences (IIIS),
Tsinghua University, Beijing 100084, China
{huanglx12,sqc12}@mails.tsinghua.edu.cn, lijian83@mail.tsinghua.edu.cn

Abstract. We study the minimum connected sensor cover problem (MIN-CSC) and the budgeted connected sensor cover (Budgeted-CSC) problem, both motivated by important applications in wireless sensor networks. In both problems, we are given a set of sensors and a set of target points in the Euclidean plane. In MIN-CSC, our goal is to find a set of sensors of minimum cardinality, such that all target points are covered, and all sensors can communicate with each other (i.e., the communication graph is connected). We obtain a constant factor approximation algorithm, assuming that the ratio between the sensor radius and communication radius is bounded. In Budgeted-CSC problem, our goal is to choose a set of B sensors, such that the number of targets covered by the chosen sensors is maximized and the communication graph is connected. We also obtain a constant approximation under the same assumption.

1 Introduction

In many applications, we would like to monitor a region or a collection of targets of interests by deploying a set of wireless sensor nodes. A key challenge in such applications is the limited energy supply for each sensor node. Hence, designing efficient algorithms for minimizing energy consumption and maximizing the lifetime of the network is an important problem in wireless sensor networks and many variations have been studied extensively. We refer interested readers to the book by Du and Wan [11] for many algorithmic problems in this domain.

In this paper, we consider two important sensor coverage problems. Now, we introduce some notations and formally define our problem. We are given a set $\mathcal{S}$ of n sensors in $\mathbb{R}^d$. All sensors in $\mathcal{S}$ have the same communication range R_c and the same sensing range R_s. In other words, two sensors s and s' can communicate with each other if $\mathsf{dist}(s, s') \le R_\mathsf{c}$, and a target point p can be covered by sensor s if $\mathsf{dist}(p, s) \le R_\mathsf{s}$. We use $D(s, R)$ to denote the disk with radius R centered at point s. Let $D_\mathsf{c}(s) = D(s, R_\mathsf{c})$ and $D_\mathsf{s}(s) = D(s, R_\mathsf{s})$.

Research supported in part by the National Basic Research Program of China Grant 2015CB358700, 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61202009, 61033001, 61361136003.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 183–196, 2015.
DOI: 10.1007/978-3-319-21398-9_15

Assumption 1. *In this paper, we assume that $R_\mathsf{s}/R_\mathsf{c}$ can be upper bounded by a constant $C = O(1)$ (i.e., $R_\mathsf{s}/R_\mathsf{c} \leq C$). Note that this assumption holds for most practical applications. Without loss of generality, we can assume that $R_\mathsf{c} = 1$. Hence, $R_\mathsf{s} = O(1)$.*

The first problem we study is the the *minimum Connected sensor covering* (MIN-CSC) problem. This problem considers the problem of selecting the minimum number of sensors that form a connected network and detect all the targets. It is somewhat similar, but different from, the connected dominating set problem. We will discuss the difference shortly. The formal problem definition is as follows:

Definition 1. MIN-CSC*: Given a set $\mathcal{S}$ of sensors and a set $\mathcal{P}$ of target points, find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of minimum cardinality such that all points in $\mathcal{P}$ are covered by the union of sensor areas in $\mathcal{S}'$ and the communication links between sensors in $\mathcal{S}'$ form a connected graph.*

In some applications, instead of monitoring a set of discrete target points, we would like to monitor a continuous range R, such as a rectangular area. Such problems can be easily converted into a MIN-CSC with discrete points, by creating a target point (which we need to cover) in each cell of the arrangement of the sensing disks $\{D_\mathsf{s}(s)\}_{s\in\mathcal{S}}$ restricted in R (see [36] for details).

The second problem studied in this paper is the *Budgeted connected sensor cover* (Budgeted-CSC) problem. The problem setting is the same as MIN-CSC, except that we have an upper bound on the number of sensors we can open, and the goal becomes to maximize the number of covered targets.

Definition 2. Budgeted-CSC*: Given a set $\mathcal{S}$ of sensors , a set $\mathcal{P}$ of target points and a positive integer B, find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq B$ and the number of points in $\mathcal{P}$ covered by the union of sensor areas in $\mathcal{S}'$ is maximum and the communication links between sensors in $\mathcal{S}'$ form a connected graph.*

1.1 Previous Results and Our Contributions

MIN-CSC. The MIN-CSC problem was first proposed by Gupta et al. [19]. They gave an $O(r \ln n)$-approximation (r is an upper bound of the hop-distance between any two sensors having nonempty sensing intersections). Wu et al. [36] give an $O(r)$-approximation algorithm, which is best approximation ratio known so far (in terms of r). If $R_\mathsf{s} \leq R_\mathsf{c}/2$, $r = 1$ and the above result implies a constant approximation. However, even R_s is slightly larger than $R_\mathsf{c}/2$, r may still be arbitrarily large. We also notice that if $r = O(1)$, we must have $R_\mathsf{s}/R_\mathsf{c} = O(1)$. So Assumption 1 is a weaker assumption than the assumption that $r = O(1)$.

MIN-CSC is in fact a special case the *group Steiner tree* problem (as also observed in Wu et al [36]). In fact, this can be seen as follows: consider the communication graph (the edges are the communication links). For each target, we create a group which consists for all sensor nodes that can cover the target. The goal is to find a minimum cost tree spanning all groups.[1] Garg et al [16], combined with the optimal probabilistic tree embedding [13], obtained an $O(\log^3 n)$ factor approximation algorithm the group Steiner tree problem via LP rounding. Chekuri et al. [6] obtained nearly the same approximation ratio using pure combinatorial method.

Our first main contribution is a constant factor approximation algorithm for MIN-CSC under Assumption 1, improving on the aforementioned results. Our improvement heavily rely on the geometry of the problem (which the group Steiner tree approach ignores).

Theorem 1. *There is a polynomial time approximation algorithm which can achieve an approximation factor $O(C^2)$ for* MIN-CSC. *Under Assumption 1, the approximation factor is a constant.*

Budgeted-CSC. Recall in Budgeted-CSC, we have a budget B, which is the upper bound of the number of sensors we can use and our goal is to maximize the number of covered target points. Kuo et al.[25] study this problem under the assumption that the communication and the sensing radius of sensors are the same (i.e., $R_{\mathsf{s}} = R_{\mathsf{c}}$). They obtained an $O(\sqrt{B})$-approximation by transforming the problem to a more general connected submodular function maximization problem.

Recently, Khuller et al. [23] obtained a constant approximation for the *budgeted generalized connected dominating set problem*, defined as follows: Given an undirected graph $G(V, E)$ and budget B, and a monotone *special submodular function* [2] $f : 2^V \to \mathbb{Z}^+$, find a subset $S \subseteq V$ such that $|S| \leq B$, S induces a connected subgraph and $f(S)$ is maximized. If $R_{\mathsf{s}} \leq R_{\mathsf{c}}/2$ in Budgeted-CSC, the coverage function $f(S)$ (the number of targets covered by sensor set S) is a special submodular function. Hence, we have a constant approximation for Budgeted-CSC when $R_{\mathsf{s}} \leq R_{\mathsf{c}}/2$. When $R_{\mathsf{s}} > R_{\mathsf{c}}/2$, $f(S)$ may not be special submodular and the algorithm and analysis in [23] do not provide any approximation guarantee for Budgeted-CSC.

We note that it is also possible to adapt the greed approach developed by group Steiner tree [6] and polymatroid Steiner tree [4] to get polylogarithmic approximation for Budgeted-CSC. However, it is unlike the approach can be made to achieve constant approximation factors, and we omit the details.

In this paper, we improve the above results by presenting the first constant factor approximation algorithm under the more general Assumption 1.

[1] Notice that the group Steiner tree is edge-weighted but MIN-CSC is node-weighted. However, since all nodes have the same (unit) weight, the edge-weight and node-weight of a tree differ by at most 1.

[2] f is a special submodular function if (1) f is submodular: $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B)$ for any $A \subset B \subsetneq V$; (2) $f(A \cup X) - f(A) = f(A \cup B \cup X) - f(A \cup B)$ if $N(X) \cup N(B) = \emptyset$ for any $X, A, B \subseteq V$. Here, $N(X)$ denotes the neighborhood of X (including X).

Theorem 2. *There is a polynomial time approximation algorithm which can achieve approximation factor of* $\frac{1}{102C^2}$ *for* Budgeted-CSC*. Under Assumption 1, the approximation factor is* $O(1)$*.*

Our algorithm is inspired by, but completely different from [23]. In particular, we make crucial use of the geometry of the problem to get around the issue required by [23] (i.e., the coverage function is required to be special submodular in their work).

1.2 Other Related Work

MIN-CSC is closely related to the the *minimum dominating set* (MIN-DS) and the *minimum connected dominating set* (MIN-CDS) problem. In fact, if the communication radius R_{c} is the same as the sensing radius R_{s}, MIN-CSC reduces to MIN-CDS. In general graphs, MIN-CDS inherits the inapproximability of set cover, so it is NP-hard to approximation MIN-CDS within a factor of $\rho \ln n$ for any $\rho < 1$ [10,14]. Improving upon Klein et al. [24], Guha et al.[18] obtained a $1.35 \ln n$-approximation, which is the best result known for general graphs.

Lichtenstein et al. [27] proved that MIN-CDS in unit disk graphs (UDG) is NP-hard (which also implies that MIN-CSC is NP-hard). The first constant approximation algorithm for the unweighted MIN-CDS problem in UDG was obtained by Wan et al.[33]. This was later improved by Cheng et al.[7], who gave the first PTAS. For the weighted (connected) dominating set problem , Ambühl et al. [1] obtained the first constant ratio approximation algorithms for both problems (the constants are 72 and 94 for MIN-DS and MIN-CDSrespectively). The constants were improved in a series of subsequent papers [9,21,35,38]. Very recently, Li and Jin [26] obtained the first PTAS for weighted MIN-DS and an improved constant approximation for weighted MIN-CDS in UDG. Many variants of MIN-DS and MIN-CDS, motivated by various applications in wireless sensor network, have been studied extensively. See [11] for a comprehensive treatment.

Budgeted-CSC is a special case of the submodular function maximization problem subject to a cardinality constraint and a connectivity constraint. Submodular maximization under cardinality constraint, which generalizes the maximum coverage problem, is a classical combinatorial optimization problem and it is known the optimal approximation is $1-1/e$ [14,29]. Submodular maximization under various more general combinatorial constraints (in particular, downward monotone set systems) is a vibrant research area in theoretical computer science and there have been a number of exciting new developments in the past few years (see e.g., [3,32] and the references therein). The connectivity constraint has also been considered in some previous work [23,25,37], some of which we mentioned before.

2 Preliminaries

We need the following *maximum coverage* (MaxCov) in our algorithms.

Definition 3. MaxCov*: Given a universe U of elements and a family $\mathcal{S}$ of subsets of U, and a positive integer B, find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq B$ and the number of elements covered by $\cup_{S \in \mathcal{S}'} S$ is maximized.*

We need to following well known result, by [20,29].

Lemma 1 (Corollary 1.1 of Hochbaum et al. [20]). *The greedy algorithm is a $(1-\frac{1}{e})$-approximation for* MaxCov.

A closely related problem is the *hitting set* problem.

Definition 4. HitSet*: Given a universe U of weighted elements (with weight function $c : U \to \mathbb{R}^+$) and a family $\mathcal{S}$ of subsets of U find a subset $H \subseteq U$ such that $H \cap S \neq \emptyset$ for all $S \in \mathcal{S}$ (i.e., H hits every subset in $\mathcal{S}$) and $\sum_{u \in H} c_u$ is minimized.*

The HitSet problem is equivalent to the set cover problem (where the elements and subsets switch roles). It is well known that a simple greedy algorithm can achieve an approximation factor of $\ln n$ for HitSet and the factor is essentially optimal [10,14]. In this paper, we use a geometric version of HitSet in which the set of given elements are points in $\mathbb{R}^2$ and the subsets are induced by given disks (i.e., each $S \in \mathcal{S}$ is the subset of points that can be covered by a given disk). Geometric hitting set admits constant factor approximation algorithms (even PTAS) for many geometric objects (including disks) [2,5,8,28,31]. As mentioned in the introduction, MIN-CSC is a special case of the following *group Steiner tree* (GST) problem.

Definition 5. GST*: We are given an undirected graph $G = (V, E, c, \mathcal{F})$ where $c : E \to \mathbb{Z}^+$ is the edge cost function, and $\mathcal{F}$ is a collection of subsets of V. Each subset in $\mathcal{F}$ is called a group. The goal is to find a subtree T, such that $T \cap S \neq \emptyset$ for all $S \in \mathcal{F}$ (i.e., T spans all groups) and the cost of the tree $\sum_{e \in T} c_e$ is minimized.*

Our algorithm for Budgeted-CSC also needs the following *quota Steiner tree* (QST) problem.

Definition 6. QST*: Given an undirected graph $G = (V, E, c, p)$ ($c : E \to \mathbb{Z}^+$ is the edge cost function, $p : V \to \mathbb{Z}^+$ is the vertex profit function) and an integer q, find a subtree $T = \arg\max_{T \subset E, \sum_{e \in T} c(e) \leq q} \sum_{v_i \in T} p(v_i)$ of the graph G (T tries to collect as much profit as possible subject to the quota constraint).*

Johnson et al. [22] proposed the QST problem and proved that any α-approximation for the k-MST problem yields an α-approximation for the QST problem. Combining with the 2-approximation for k-MST developed by Garg [15], we can get a 2-approximation for the QST problem.

Lemma 2. *These is an approximation algorithm with approximation factor* 2 *for* QST.

3 Minimum Connected Sensor Cover

We first construct an edge-weighted graph $\mathcal{G}_{\mathsf{c}}$ as follows: If $\mathsf{dist}(s, s') \leq R_{\mathsf{c}}$, we add an edge between s and s' (It is easy to see that $\mathcal{G}_{\mathsf{c}}$ is in fact a unit disk graph). $\mathcal{G}_{\mathsf{c}}$ is called *the communication graph.* Recall that MIN-CSC requires us to find a set of vertices that induces a connected subgraph in the communication graph $\mathcal{G}_{\mathsf{c}}$.

First, we note that $\mathcal{G}_{\mathsf{c}}$ may have several connected components. We can see any feasible solution must be contained in a single connected component (otherwise, the solution can not induce a connected graph). Our algorithm tries to find a solution in every connected component. Our final solution will be the one with the minimum cost among all connected component. Note that for some connected component, there may not be a feasible solution in that component (some target point can not be covered by any point in that component), and our algorithm ignores such component.

From now on, we fix a connected component $\mathcal{C}$ in $\mathcal{G}_{\mathsf{c}}$. Similar with Wu et al. [36], we formulate the MIN-CSC problem as a group Steiner tree (GST) problem. Each edge $e \in \mathcal{G}[\mathcal{C}]$ is associated with a cost $c_e = 1$. For each target $p \in \mathcal{P}$, we create a group

$$\mathsf{gp}(p) = \mathcal{C} \cap D(p, R_{\mathsf{s}}) = \{s \mid s \in \mathcal{C}, \mathsf{dist}(p, s) \leq R_{\mathsf{s}}\}.$$

The goal is to find a tree $\mathcal{T}$ (in $\mathcal{G}[\mathcal{C}]$) such that $\mathcal{T} \cap \mathsf{gp}(p) \neq \emptyset$ for all $p \in \mathcal{P}$ and the cost is minimized. We can easily see the GST instance constructed above is equivalent to the original MIN-CSC problem (the cost of the tree $\mathcal{T}$ is the number of nodes in $\mathcal{T}$ minus 1). The GST problem can be formulated as the following linear integral program: We pick a root $r \in \mathcal{C}$ for the tree $\mathcal{T}$ (we need to enumerate all possible roots). For each edge $e \in \mathcal{G}[\mathcal{C}]$, we use Boolean variable x_e to denote whether we choose edge e.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in \mathcal{G}[\mathcal{C}]} x_e \\ \text{subject to} \quad & \sum_{e \in \partial(S)} x_e \geq 1, \quad \text{for all } S \subset \mathcal{C} \text{ such that } r \in S \text{ and } \exists p, S \cap G_p = \emptyset; \\ & x_e \in \{0,1\}, \quad \forall e \in \mathcal{G}[\mathcal{C}]. \end{aligned} \tag{1}$$

The second constraint says that for any cut $\partial(S)$ that separates the root r from any group, there must be at least one chosen edge. By replacing $x_e \in \{0, 1\}$ with $x \in [0, 1]$, we obtain the linear programming relaxation of (1) (denoted as Lp-GST). By the duality between flow and cut, we can see that the second constraint is equivalent to dictating that we can send at least 1 unit of flow from the root r to nodes in $\mathsf{gp}(p)$, for each p. This flow viewpoint (also observed in the original GST paper [16]) will be particularly useful to us later. So we write down the flow LP explicitly as follows. We first replace every undirected edge $e = (u, v)$ by two directed arcs (u, v) and (v, u). For each $p \in \mathcal{P}$ and each directed arc (u, v), we have a variable x^p_{uv} indicating the flow of commodity p on arc (u, v). We use $y^p_v = \sum_u x^p_{uv} - \sum_w x^p_{vw}$ to denote the net flow of commodity p into node v. Then $\mathsf{Lp\text{-}GST}$ can be equivalently rewritten as the following linear program (denoted as $\mathsf{Lp\text{-}flow}$):

$$\begin{aligned}
\text{minimize} \quad & \sum_{(u,v)\in\mathcal{G}[\mathcal{C}]} x_{uv} \qquad (2)\\
\text{subject to} \quad & y^p_v = \sum_u x^p_{uv} - \sum_w x^p_{vw} \quad \text{for all } v \in \mathcal{C}\\
& y^p_r = -1 \quad \text{for all } p \in \mathcal{P},\\
& \sum_{v\in\mathsf{gp}(p)} y^p_v \geq 1 \quad \text{for all } p \in \mathcal{P},\\
& y^p_u = 0 \quad \text{for all } u \notin \mathsf{gp}(p), u \neq r,\\
& x^p_{uv} \leq x_{uv} \quad \text{for all } p \in \mathcal{P}, u, v \in \mathcal{C},\\
& x^p_{uv}, y^p_v \in [0, 1], \quad \text{for all } u, v \in \mathcal{G}[\mathcal{C}].
\end{aligned}$$

Now, we describe our algorithm. Our algorithm mainly consists of two steps. In the first step, we extract a *geometric hitting set* instance from the optimal fractional solution of $\mathsf{Lp\text{-}flow}$. We can find an integral solution H for the hitting set problem and we can show its cost is at most $O(C^2\mathsf{OPT})$. Moreover all sensors in H can cover all target points $p \in \mathcal{P}$. In the second step, we extract a Steiner tree instance, again from the optimal fractional solution of $\mathsf{Lp\text{-}flow}$. We show it is possible to round the Steiner tree LP to get a constant approximation integral Steiner tree, which can connect all points in H.

Step 1: Constructing the Hitting Set Problem :

We first solve the linear program $\mathsf{Lp\text{-}flow}$ and obtain the fractional optimal solution (x_{uv}, y_v). Let $\mathsf{Opt}(\mathsf{Lp\text{-}flow})$ to denote the optimal value of $\mathsf{Lp\text{-}flow}$. We place a grid with grid size $l = \frac{\sqrt{2}}{2}$ in the plane (i.e., each cell is a $\frac{\sqrt{2}}{2} \times \frac{\sqrt{2}}{2}$ square). For each $p \in \mathcal{P}$, consider the set of sensors $\mathsf{gp}(p)$, that is the set of sensors which can cover p. Since $\mathsf{gp}(p)$ is contained in a disk of radius $R_\mathsf{s} \leq C$, there are at most $\frac{\sqrt{2}}{2}O(C^2) = O(1)$ grid cells that may contain some points in $\mathsf{gp}(p)$. Since $\sum_{v\in\mathsf{gp}(p)} y^p_v \geq 1$, there must be a cell (say $\mathsf{cl}(p)$) such that

$$\sum_{v\in\mathsf{gp}(p)\cap\mathsf{cl}(p)} y^p_v \geq \Omega(1/C^2) = \Omega(1). \qquad (3)$$

Now, we construct a geometric hitting set (HitSet) instance $(\mathcal{U}, \mathcal{F})$ as follows: Let the set of points be $\mathcal{U} = \cup_{p\in\mathcal{P}}(\mathsf{gp}(p) \cap \mathsf{cl}(p))$ and the family of subsets be $\mathcal{F} = \{\mathsf{gp}(p)\}_{p\in\mathcal{P}}$. The goal is to choose a subset H of $\mathcal{U}$ such that $\mathsf{gp}(p) \cap H \neq \emptyset$ for all $p \in \mathcal{P}$ (i.e., we want to hit every set in $\mathcal{F}$). Write the linear program relaxation for the HitSet problem (denoted as $\mathsf{Lp\text{-}HS}$):

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in \mathcal{U}} z_u & (4) \\ \text{subject to} \quad & \sum_{u \in \mathsf{gp}(p)} z_u \geq 1 \quad \text{for all } p \in \mathcal{P}, \\ & z_u \in [0,1], \quad \text{for all } u \in \mathcal{U}. \end{aligned}$$

Let $\mathsf{Opt}(\mathsf{Lp\text{-}HS})$ to denote the optimal value of $\mathsf{Lp\text{-}HS}$. We need the following simple lemma. Due to space constraints, the proof can be found in the full version of this paper.[3]

Lemma 3. $\mathsf{Opt}(\mathsf{Lp\text{-}HS}) \leq O(C^2 \mathsf{Opt}(\mathsf{Lp\text{-}flow}))$.

Bronnimann et al. [2], combined with the existence of ϵ-net of size $O(1/\epsilon)$ for disks (see e.g., [30]), showed that we can round the above linear program $\mathsf{Lp\text{-}HS}$ to obtain an integral solution (i.e., an actual hitting set) $H \subset \mathcal{U}$ such that $|H| \leq O(\mathsf{Opt}(\mathsf{Lp\text{-}HS}))$ (the connection to ϵ-net was made simpler and more explicit in Even et al. [12]). Hence, $|H| \leq O(C^2 \mathsf{OPT})$.

Step 2: Constructing the Steiner Tree Problem : Recall that for each $p \in \mathcal{P}$, there is a cell $\mathsf{cl}(p)$ such that $\sum_{v \in \mathsf{gp}(p) \cap \mathsf{cl}(p)} y_v^p \geq \Omega(1/C^2)$. Consider the collection $\Delta = \{\mathsf{cl}(p) \mid p \in \mathcal{P}\}$ of all such cells (if there is a cell which contains the root r, we exclude it from Δ), from each cell $\mathsf{cl} \in \Delta$, we pick an arbitrary point, called the *representative node* $v(\mathsf{cl})$ of cl. From Equation 3 (i.e., $\sum_{v \in \mathsf{gp}(p) \cap \mathsf{cl}(p)} y_v^p \geq \Omega(1/C^2)$), we can see at least $\Omega(1/C^2)$ flow of commodity p that enters $\mathsf{cl}(p)$. Now, we *reroute* such flow to the representative $v(\mathsf{cl}(p))$, in order to create a Steiner tree LP. Consider the optimal fractional solution (x_{uv}, y_v) of $\mathsf{Lp\text{-}flow}$. We would like to create another feasible fractional solution $(\widehat{x}_{uv}, \widehat{y}_v)$ for $\mathsf{Lp\text{-}flow}$.

- (Flow Rerouting) For each node $u \in \mathsf{gp}(p) \cap \mathsf{cl}(p)$, let $\widetilde{x}^p_{uv(\mathsf{cl}(p))} \leftarrow x^p_{uv(\mathsf{cl}(p))} + y_u$. In other words, we route the flow excess at node u to node $v(\mathsf{cl}(p))$. After such updates, for each $u \in \mathsf{gp}(p) \cap \mathsf{cl}(p), u \neq v(\mathsf{cl}(p))$ we can see the flow excess is zero, or equivalently $\widetilde{y}^p_u = 0$. The flow excess at node $v(\mathsf{cl}(p))$ is
$$\widetilde{y}^p_{v(\mathsf{cl}(p))} = \sum_{v \in \mathsf{gp}(p) \cap \mathsf{cl}(p)} y_v^p \geq \Omega(1/C^2).$$
We repeat the above process for all $\mathsf{cl} \in \Delta$.
- By uniformly increasing all variables, we obtain another feasible solution $(\widehat{x}_{uv}, \widehat{y}_v)$:
$$\widehat{x}^p_{uv} = \min\{C^2 \widetilde{x}^p_{uv}, 1\} \quad \text{and} \quad \widehat{y}^p_v = \min\{C^2 \widetilde{y}^p_v, 1\}.$$
Then, it is easy to see that $\widehat{y}^p_{v(\mathsf{cl})} \geq 1$ for all $\mathsf{cl} \in \Delta$. In equivalent words, at least 1 unit flow (thinking $\widehat{x}^p_{uv}$ as flow value on (u, v)) that enters $v(\mathsf{cl}(p))$.

[3] A full version is available from the CS arXiv.

Now, consider the Steiner tree problem in $\mathcal{G}(\mathcal{C})$ in which the set of terminals is defined to be $\mathsf{Ter} = \{r\} \cup \{v(\mathsf{cl}) \mid \mathsf{cl} \in \Delta\}$. Let $\check{x}_e = \max_{p\in\mathcal{P}} \widehat{x}^p_{uv} + \max_{p\in\mathcal{P}} \widehat{x}^p_{vu}$(Notice that $\mathsf{Lp\text{-}flow}$ is formulated on directed graphs and Steiner tree is formulated on undirected graphs. Here e is the undirected edge corresponding to directed edges uv and vu). It is easy to see that $\check{x}_e$ is a feasible solution for the following linear program relaxation for the Steiner tree problem (denoted as $\mathsf{Lp\text{-}ST}$):

$$\text{minimize} \quad \sum_{e\in\mathcal{G}[\mathcal{C}]} x_e \tag{5}$$

$$\text{subject to} \quad \sum_{e\in\partial(S)} x_e \geq 1, \quad \text{for all } S \subset \mathcal{C} \text{ such that } r \in S \text{ and } \exists \mathsf{cl} \in \Delta, v(\mathsf{cl}) \notin S$$

$$x_e \in \{0,1\}, \quad \forall e \in \mathcal{G}[\mathcal{C}].$$

Lemma 4. $\mathsf{Opt}(\mathsf{Lp\text{-}ST}) \leq O(C^2\mathsf{Opt}(\mathsf{Lp\text{-}flow}))$.

It is well known that the integrality gap of the Steiner tree problem is a constant [34]. In particular, it is known that using the primal-dual method (based on $\mathsf{Lp\text{-}ST}$) in [17] (see also [34, Chapter7.2]), we can obtain an integral solution $\overline{x}_e$ such that

$$\sum_{e\in\mathcal{G}[\mathcal{C}]} x_e \leq 2\mathsf{Opt}(\mathsf{Lp\text{-}ST}) \leq O(C^2\mathsf{Opt}(\mathsf{Lp\text{-}flow})) \leq O(C^2\mathsf{OPT}).$$

Let J be the set of vertices spanned by the integral Steiner tree $\{\overline{x}_e\}$. The above discussion shows that $|J| \leq O(C^2\mathsf{OPT})$. Our final solution (the set of sensors we choose) is $\mathsf{Sol} = H \cup J$. The feasibility of Sol is proved in the following simple lemma.

Lemma 5. Sol *is a feasible solution.*

Proof. We only need to show that Sol induces a connected graph and covers all the target points. Obviously, H covers all target points, so does Sol. Since J is a Steiner tree, thus connected. Moreover, J connects all representatives $v(\mathsf{cl})$ for all $\mathsf{cl} \in \Delta$. H consists of only sensors in $\mathsf{cl} \in \Delta$. So every sensor in $v \in H$ (say $v \in \mathsf{cl}$) is connected to the representative $v(\mathsf{cl})$. So $H \cup J$ induces a connected subgraph. □

Lastly, we need to show the performance guarantee. This is easy since we have shown that both $|H| \leq O(C^2\mathsf{OPT})$ and $|J| \leq O(C^2\mathsf{OPT})$. So $|\mathsf{Sol}| = O(C^2\mathsf{OPT}) = O(\mathsf{OPT})$ since C is assumed to be a constant.

4 Budgeted Connected Sensor Cover

Again we assume that $R_{\mathsf{c}} = 1$ and $R_{\mathsf{s}} = C$. Recall that our goal is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of sensors with cardinality B which induces a connected subgraph and covers as many targets as possible. We first construct the communication graph $\mathcal{G}_{\mathsf{c}}$ as in Section 3. Again, we only need to focus on a connected component of $\mathcal{G}_{\mathsf{c}}$. Then we find a square Q in the Euclidean plane large enough such that all of the n sensors are inside Q. We partition Q into small square cells of equal size. Let the side length of each cell be $l = \frac{\sqrt{2}}{2}$. Denote the cell in the ith row and jth column of the partition as $\mathsf{cl}_{i,j}$. Let $V_{i,j} = \{v \in \mathcal{S} \mid v \in \mathsf{cl}_{i,j}\}$ be the collection of sensors in $\mathsf{cl}_{i,j}$. We then partition these cells into k^2 different cell groups $\mathsf{CG}_{a,b}$, where $k = \lceil 2C/l + 1 \rceil$. In particular, we let

$$\mathsf{CG}_{a,b} = \{\mathsf{cl}_{i,j} \mid i \equiv a (\bmod k), j \equiv b (\bmod k)\} \text{ for } a \in [k], b \in [k].$$

and $\mathcal{V}_{a,b} = \mathcal{S} \cap \mathsf{CG}_{a,b}$ be the collection of sensors in $\mathsf{CG}_{a,b}$.

With the above value k, we make a simple but useful observation as follows.

Observation 1. *There is no target covered by two different sensors contained in two different cells of* $\mathsf{CG}_{a,b}$.

Denote the optimal solution of $\mathsf{Budgeted\text{-}CSC}$ problem as OPT. In this section, we present an $O\big(\frac{1}{C^2}\big)$ factor approximation algorithm for the $\mathsf{Budgeted\text{-}CSC}$ problem.

4.1 The Algorithm

For $0 \le a, b < k$, we repeat the following two steps, and output a tree T with $O\big(B\big)$ vertices (sensors) which covers the maximum number of targets. Then based on T, we find a subtree $\tilde{T}$ with exactly B vertices as our final output.

Step 1: Reassign profit : The profit $p(S)$ of a subset $S \subseteq \mathcal{S}$ is the number of targets covered by S. $p(S)$ is a submodular function. In this step, we design a new profit function (called *modified profit function*) $\widehat{p} : \mathcal{S} \to \mathbb{Z}^+$ for the set of sensors. To some extent, $\widehat{p}$ is a linearized version of p (module a constant approximation factor).

Now, we explain in details how $\widehat{p}$ is defined. Fix a cell group $\mathsf{CG}_{a,b}$. [4] For the vertices in $\mathsf{CG}_{a,b}$, we use the greedy algorithm to reassign profits of the vertices in $\mathcal{V}_{a,b}$. Among all vertices in $\mathcal{V}_{a,b}$, we pick a vertex v_1 which can cover the most number of targets, and use this number as its modified profit $\widehat{p}(v_1)$. Remove the chosen vertex and targets covered by it. We continue to pick the vertex v_2 in $\mathcal{V}_{a,b}$ which can cover the most number of uncovered targets. Set the modified profit $\widehat{p}(v_2)$ to be the number of newly covered targets. Repeat the above steps until all the sensors in $\mathcal{V}_{a,b}$ have been picked out. For other vertices v which are not in $\mathcal{V}_{a,b}$, we simply set their modified profit $\widehat{p}(v)$ as 0.

Let us first make some simple observations about p and $\widehat{p}$. We use $\widehat{p}(S)$ to denote $\sum_{v \in S} \widehat{p}(v)$. First, it is not difficult to see that $\widehat{p}(S) \le p(S)$ for any subset $S \subseteq \mathcal{S}$. Second, we can see that it is equivalent to run the greedy algorithm for each cell in $\mathsf{CG}_{a,b}$ separately (due to Observation 1). Suppose $S_1 \subseteq \mathsf{cl}_{c,d}$,

[4] For each $\mathsf{CG}_{a,b}$, we define a modified profit function $\widehat{p}_{a,b}$. For ease of notation, we omit the subscripts.

$S_2 \subseteq \mathsf{cl}_{c',d'}$ where $\mathsf{cl}_{c,d}$ and $\mathsf{cl}_{c',d'}$ are two different cells in $\mathsf{CG}_{a,b}$, then $p(S_1 \cup S_2) = p(S_1) + p(S_2)$ due to Observation 1.

Consider a cell $\mathsf{cl}_{c,d} \in \mathsf{CG}_{a,b}$. Let $D_{c,d} = \{v_1, v_2, ..., v_n\} \subseteq \mathsf{cl}_{c,d} \cap \mathcal{S}$, where the vertices are indexed by the order in which they were selected by the greedy algorithm. Let $D^i_{c,d} = \{v_1, v_2, ..., v_i\}$ be the first i vertices in $D_{c,d}$. By the following lemma, we can see that the modified profit function $\widehat{p}$ is a constant approximation to true profit function p over any vertex subset $V \subseteq \mathcal{V}_{a,b}$.

Lemma 6. *For a set of vertices V in the same cell $\mathsf{cl}_{c,d} \in \mathsf{CG}_{a,b}$, such that $|V| \leq i$, we have that $p(D^i_{c,d}) = \widehat{p}(D^i_{c,d}) \geq (1 - 1/e)p(V)$.*

Proof. By the greedy rule, we can see $p(D^i_{c,d}) = \widehat{p}(D^i_{c,d})$. By Lemma 1, we know that $\widehat{p}(D^i_{c,d}) \geq (1 - 1/e) \max_{|V| \leq i} p(V)$. $\square$

Step 2: Guess the optimal profit and calculate a tree T : Although the actual profit of OPT is unknown, we can guess the profit of OPT (by enumerating all possibilities). For each $0 \leq a, b < k$, we calculate in this step a tree T of size at most $4B$, using the QST algorithm (see Lemma 2). We can show that among these trees (for different a, b values), there must be one tree of profit no less than $\frac{1}{k^2}\left(1 - \frac{1}{e}\right)\mathsf{OPT}$.

After choosing the best tree T with the highest profit, we construct a subtree $\tilde{T}$ of size B based on T as our final solution of $\mathsf{Budgeted\text{-}CSC}$.

We first show that there exists $0 \leq a, b < k$, such that based on the modified profit $\widehat{p}$ on $\mathsf{CG}_{a,b}$, there exists a tree with at most $2B$ vertices of total modified profit at least $\frac{1}{k^2}\left(1 - \frac{1}{e}\right)\mathsf{OPT}$. We use T_{OPT} to denote the set of vertices of the optimal solution.

Lemma 7. *There exists a tree T_0 in $\mathcal{G}_{\mathsf{c}}$, $|T_0| \leq 2B$ such that $\widehat{p}(T_0) \geq \dfrac{1}{k^2}\left(1 - \dfrac{1}{e}\right)\mathsf{OPT}$.*

Then, by the Lemma 2 and Lemma 7, if we run the QST algorithm (with $\widehat{p}$ as the profit function), we can obtain the suitable tree T with at most $4B$ vertices of profit at least $\frac{1}{k^2}\left(1 - \frac{1}{e}\right)p(\mathsf{OPT})$. The pseudocode of the algorithm can be found in the full version of the paper.

Lemma 8. *Let T be the tree obtained by the above step, then $p(T) \geq \frac{1}{k^2}\left(1 - \frac{1}{e}\right)\mathsf{OPT}$*

Proof. By Lemma 7, we can obtain a tree T with at most $4B$. We also have $\widehat{p}(T) \geq \frac{1}{k^2}\left(1 - \frac{1}{e}\right)\mathsf{OPT}$. Since $p(S) \geq \widehat{p}(S)$ for any S, we have proved the lemma. $\square$

Finally, we construct a subtree $\tilde{T}$ of B vertices based on tree T. This can be done using a simple dynamic program, in the same way as [23]. Denote the subtree with highest total profit as $\tilde{T}$. We can show the following lemma.[5]

[5] The proof is similar to that in [23] and can be find in the full version of the paper.

Lemma 9. $p(\tilde{T}) \geq \frac{1}{8}p(T)$.

Use the same dynamic programming algorithm in Khuller et al. [23], we can find $\tilde{T}$ from tree T. Combining Lemma 8 and Lemma 9, $p(\tilde{T}) \geq \frac{1}{8}\left(1 - \frac{1}{e}\right)\frac{1}{(2\sqrt{2}C+1)^2}\mathsf{OPT} = \frac{1}{12.66(8C^2+4\sqrt{2}C+1)}\mathsf{OPT} \geq \frac{1}{102C^2}\mathsf{OPT}$ (When C is large). Thus, we have obtained Theorem 2.

5 Conclusion and Future Work

There are several interesting future directions. The first obvious open question is that whether we can get constant approximations for MIN-CSC and Budgeted-CSC without Assumption 1 (it would be also interesting to obtain approximation ratios that have better dependency on C). Generalizing the problem further, an interesting future direction is the case where different sensors have different transmission ranges and sensing ranges. Whether the problems admit better approximation ratios than the (more general) graph theoretic counterparts is still wide open.

References

1. Ambühl, C., Erlebach, T., Mihalák, M., Nunkesser, M.: Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 3–14. Springer, Heidelberg (2006)
2. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite vc-dimension. DCG **14**(1), 463–479 (1995)
3. Calinescu, G., Chekuri, C., Pál, M., Vondrák, J.: Maximizing a monotone submodular function subject to a matroid constraint. SICOMP **40**(6), 1740–1766 (2011)
4. Calinescu, G., Zelikovsky, A.: The polymatroid steiner problems. JCO **9**(3), 281–294 (2005)
5. Chan, T.M., Grant, E., Könemann, J., Sharpe, M.: Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In: SODA, pp. 1576–1585. SIAM (2012)
6. Chekuri, C., Even, G., Kortsarz, G.: A greedy approximation algorithm for the group steiner problem. Discrete Applied Mathematics **154**(1), 15–34 (2006)
7. Cheng, X., Huang, X., Li, D., Wu, W., Du, D.Z.: A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. Networks **42**(4), 202–208 (2003)
8. Clarkson, K.L., Varadarajan, K.: Improved approximation algorithms for geometric set cover. DCG **37**(1), 43–58 (2007)
9. Dai, D., Yu, C.: A 5+ ϵ-approximation algorithm for minimum weighted dominating set in unit disk graph. TCS **410**(8), 756–765 (2009)
10. Dinur, I., Steurer, D.: Analytical approach to parallel repetition. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, pp. 624–633. ACM (2014)
11. Du, D.Z., Wan, P.J.: Connected Dominating Set: Theory and Applications, vol. 77. Springer Science & Business Media (2012)

12. Even, G., Rawitz, D., Shahar, S.M.: Hitting sets when the vc-dimension is small. IPL **95**(2), 358–362 (2005)
13. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: STOC, pp. 448–455. ACM (2003)
14. Feige, U.: A threshold of ln n for approximating set cover. JACM **45**(4), 634–652 (1998)
15. Garg, N.: Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In: STOC, pp. 396–402. ACM (2005)
16. Garg, N., Konjevod, G., Ravi, R.: A polylogarithmic approximation algorithm for the group steiner tree problem. In: SODA, pp. 253–259. SIAM (1998)
17. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SICOMP **24**(2), 296–317 (1995)
18. Guha, S., Khuller, S.: Improved methods for approximating node weighted steiner trees and connected dominating sets. Information and computation **150**(1), 57–74 (1999)
19. Gupta, H., Zhou, Z., Das, S.R., Gu, Q.: Connected sensor cover: self-organization of sensor networks for efficient query execution. IEEE/ACM Transactions on Networking **14**(1), 55–67 (2006)
20. Hochbaum, D.S., Pathria, A.: Analysis of the greedy approach in problems of maximum k-coverage. Naval Research Logistics **45**(6), 615–627 (1998)
21. Huang, Y., Gao, X., Zhang, Z., Wu, W.: A better constant-factor approximation for weighted dominating set in unit disk graph. JCO **18**(2), 179–194 (2009)
22. Johnson, D.S., Minkoff, M., Phillips, S.: The prize collecting steiner tree problem: theory and practice. In: SODA, vol. 1, p. 4. Citeseer (2000)
23. Khuller, S., Purohit, M., Sarpatwar, K.K.: Analyzing the optimal neighborhood: algorithms for budgeted and partial connected dominating set problems. In: SODA, pp. 1702–1713. SIAM (2014)
24. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted steiner trees. Journal of Algorithms **19**(1), 104–115 (1995)
25. Kuo, T.W., Lin, K.J., Tsai, M.J.: Maximizing submodular set function with connectivity constraint: theory and application to networks. In: INFOCOM, pp. 1977–1985. IEEE (2013)
26. Li, J., Jin, Y.: A PTAS for the weighted unit disk cover problem. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9134, pp. 898–909. Springer, Heidelberg (2015)
27. Lichtenstein, D.: Planar formulae and their uses. SICOMP **11**(2), 329–343 (1982)
28. Mustafa, N.H., Ray, S.: PTAS for geometric hitting set problems via local search. In: SCG, pp. 17–22. ACM (2009)
29. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions. Mathematical Programming **14**(1), 265–294 (1978)
30. Pyrga, E., Ray, S.: New existence proofs ε-nets. In: SCG, pp. 199–207. ACM (2008)
31. Varadarajan, K.: Weighted geometric set cover via quasi-uniform sampling. In: STOC, pp. 641–648. ACM (2010)
32. Vondrák, J., Chekuri, C., Zenklusen, R.: Submodular function maximization via the multilinear relaxation and contention resolution schemes. In: STOC, pp. 783–792. ACM (2011)
33. Wan, P.J., Alzoubi, K.M., Frieder, O.: Distributed construction of connected dominating set in wireless ad hoc networks. In: INFOCOM, vol. 3, pp. 1597–1604. IEEE (2002)

34. Williamson, D.P., Shmoys, D.B.: The design of approximation algorithms. Cambridge University Press (2011)
35. Willson, J., Ding, L., Wu, W., Wu, L., Lu, Z., Lee, W.: A better constant-approximation for coverage problem in wireless sensor networks (preprint)
36. Wu, L., Du, H., Wu, W., Li, D., Lv, J., Lee, W.: Approximations for minimum connected sensor covereq:cellsum. In: INFOCOM, pp. 1187–1194. IEEE (2013)
37. Zhang, W., Wu, W., Lee, W., Du, D.Z.: Complexity and approximation of the connected set-cover problem. Journal of Global Optimization **53**(3), 563–572 (2012)
38. Zou, F., Wang, Y., Xu, X.H., Li, X., Du, H., Wan, P., Wu, W.: New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs. TCS **412**(3), 198–208 (2011)

Circuits Algorithms

Skew Circuits of Small Width

Nikhil Balaji[1](✉), Andreas Krebs[2], and Nutan Limaye[3]

[1] Chennai Mathematical Institute, Chennai, India
nikhil@cmi.ac.in
[2] University of Tübingen, Tübingen, Germany
mail@krebs-net.de
[3] Indian Institute of Technology, Bombay, India
nutan@cse.iitb.ac.in

Abstract. In this work, we study the power of bounded width branching programs by comparing them with bounded width skew circuits.

It is well known that branching programs of bounded width have the same power as skew circuit of bounded width. The naive approach converts a BP of width w to a skew circuit of width w^2. We improve this bound and show that BP of width $w \geq 5$ can be converted to a skew circuit of width 7. This also implies that skew circuits of bounded width are equal in power to skew circuits of width 7. For the other way, we prove that for any $w \geq 2$, a skew circuit of width w can be converted into an equivalent branching program of width w. We prove that width-2 skew circuits are not universal while width-3 skew circuits are universal and that any polynomial sized CNF or DNF is computable by width 3 skew circuits of polynomial size.

We prove that a width-3 skew circuit computing Parity requires exponential size. This gives an exponential separation between the power of width-3 skew circuits and width-4 skew circuits.

1 Introduction

The Boolean circuit complexity class NC^1 consists of Boolean functions computable by polynomial sized logarithmic depth circuits. Basic arithmetic operations like addition, multiplication and division are known to be in NC^1. All regular languages have uniform NC^1 families deciding them and there is a regular language which is NC^1-hard. Over the years, several useful characterizations of NC^1 have emerged: NC^1 contains exactly those regular languages that are characterized by having a monoid containing a non-solvable group. They are also equally expressive as Branching Programs of constant width. Our interest in NC^1 is motivated by the celebrated result of Barrington [Bar89], that Branching Programs of width 5 are sufficient to capture NC^1 in its entirety.

Branching programs have been pivotal to our understanding of computation with limited resources. They were first defined in [Lee59] and formally studied by Masek in his thesis [Mas76]. Borodin et al.[BDFP86] proved that AC^0 is contained in the class of functions computed by bounded width branching

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 199–210, 2015.
DOI: 10.1007/978-3-319-21398-9_16

programs and conjectured that Majority cannot be computed by them. In a surprising result, Barrington showed that in fact, width 5 branching programs can compute all of NC^1 and hence the Majority function.

After the strong lower bound results of [Raz87][Smo87] for AC^0, the question of proving lower bounds for NC^1 gained a lot of attention. However this has turned out to be a notorious open problem. The branching program characterization of NC^1 has provided an avenue to understand the power of classes that reside inside NC^1. Though proving lower bounds for width 5 branching programs is equivalent to proving lower bounds for NC^1, it is conceivable that proving lower bounds for width 4 branching programs is easier. In this regard, it is known [Bar85] that width 3 branching programs of a restricted type (permutation branching programs) require exponential size to compute the AND function. It is worthwhile to contrast this against the situation at width 5, where permutation branching programs are known to be as powerful as general branching programs of width 5 and hence NC^1 itself.

It is known that bounded width branching programs can be equivalently thought of as bounded width skew circuits (see for example [RR10]). Here, we take a closer look at this relationship. The folklore construction[1] converts a polynomial size branching program of width w into a polynomial size skew circuit of width w^2. We improve this construction and show that any bounded width branching program of width greater than or equal to 5 can be converted into an equivalent skew circuit of width 7. We also study the conversion of skew circuits into branching programs. Here, the known construction converts a skew circuit of width w into a branching program of width $w + 1$ [RR10]. We improve this construction and prove that a polynomial size skew circuit of width w can be converted into a polynomial size branching program of width w. These results prove that width 7 skew circuits of polynomial size characterize NC^1.

These structural results allow us to examine the set of languages in NC^1 by varying the width of skew circuits between 1 and 7. Like for permutation branching programs, some natural questions arise for bounded width skew circuits. We start by examining the power of width 2 skew circuits. We observe that they are not universal as they cannot compute parity of two bits.

We then study the power of width 3 skew circuits. Recall that a CNF (DNF) is an AND (OR) of ORs (ANDs) of variables, i.e. in a CNF the AND gate is (possibly) non-skew. We implement a CNF by a width 3 skew circuit. Formally, we prove that any k-CNF or any k-DNF of size s has width 3 skew circuits of length $O(sk)$. Given that any Boolean function on n variables has a CNF of exponential (in n) size, this also proves that width 3 skew circuits are universal.

We consider the problem of proving lower bound for width 3 skew circuits. A natural candidate is a function which has no polynomial sized CNF or DNF. It is known that Parity is one such function. We prove that Parity requires width 3 skew circuits of exponential size. We observe that Parity and Approximate Majority have respectively, linear and polynomial size width 4 skew circuits. This separates width 3 skew circuits from width 4 skew circuits.

[1] Replace each wire by an AND gate and each node by an OR gate.

2 Preliminaries

A directed acyclic graph $G = (V, E)$ is called layered if the vertex set of the graph can be partitioned, $V = V_1 \cup \ldots \cup V_\ell$ in such a way that for each edge $e = (u, v)$ there exists $1 \leq i < \ell$ such that $u \in V_i$ and $v \in V_{i+1}$. Given a layered graph G the *length* of the graph is the number of layers in it and the *width* of the graph is the maximum over $i \in [\ell]$, $|V_i|$.

Definition 1. *(Branching Programs) A Deterministic Branching Program (BP) is a layered directed acyclic graph G with the following properties:*

- *There is a designated source vertex s in the first layer (of in-degree 0) and a sink vertex t (of out-degree 0) in the last layer.*
- *The edges are labelled by an element of $X \cup \{0, 1\}$, where X is the set of input variables to the branching program.*

The branching program naturally computes a boolean function $f(X)$, where $f(X) = 1$ if and only if there is path from s to t in which each edge is labelled by a true literal or a constant 1 on input X. The length *(*width*) of the BP is the length (respectively, width) of the underlying layered DAG.*

We will denote the class of languages accepted by width-w BP by BP^w. Barrington [Bar85][Bar89] defined a restricted notion of branching programs called the Permutation Branching Program (PBP):

Definition 2. *(Permutation Branching Programs as a graph) A width-w PBP is a layered width w BP in which the following conditions hold:*

- *There are designated source vertices $s_1, s_2, \ldots, s_w$ in the first layer, say layer 1 (of in-degree 0) and sink vertices $t_1, t_2, \ldots, t_w$ (of out-degree 0) in the last layer, layer ℓ.*
- *Each layer has exactly w vertices.*
- *In each layer $1 \leq i < \ell$, all the edges are labelled by a unique variable, say x_{j_i}.*
- *In each layer $1 \leq i \leq \ell$ and $b \in \{0, 1\}$, the edges activated when $x_{j_i} = b$ forms a permutation/matching, say $\theta_{i,b}$.*

The permutation branching program naturally computes a boolean function $f(X)$, where $f(X) = 1$ if and only if there is path from s_1 to t_1, s_2 to t_2, and so on till s_w to t_w, where in each path each edge is labelled by a true literal or a constant 1 on input X. We will refer to the class of languages accepted by polynomial sized width-w PBP by PBP^w.

The above definition of PBP can be rephrased as follows:

Definition 3. *(Permutation Branching Programs as a set of instructions) A width-w length-ℓ PBP is a program given by a set of ℓ instructions in which for any $1 \leq i \leq \ell$, the ith instruction is a three tuple $\langle j_i, \theta^i, \sigma^i \rangle$, where j_i is an index from $\{1, 2, \ldots, |X|\}$, θ^i, σ^i are permutations of $\{1, 2, \ldots, w\}$. The output of the instruction is θ^i if $x_{j_i} = 1$ and it is σ^i if $x_{j_i} = 0$. The output of the program on input x is the product of the output of each instruction of the program on x.*

We say that a permutation branching program computes a function f if there exists a fixed permutation $\pi \neq id$ such that for every x such that $f(x) = 1$ the program outputs π and for every x such that $f(x) = 0$ the program outputs id[2].

It is easy to see that the above two definitions of PBP are equivalent.

Definition 4. *(Skew Circuits) An AND gate is called* skew *if all but one of its children are input variables. A Boolean circuit in which all the AND gates are skew is called a* skew circuit*.*

We assume that the skew circuits are layered. The width of the circuit is the maximum number of gates in any layer. The layer may have AND, OR or input gates. Each type of gate contributes towards the width. We assume that the fan-in of the AND gates is bounded by 2 and there are no NOT gates (negations appear only for the input variables)[3]. We denote the class of languages decided by width-w skew circuits by SK^w. The following lemma summarises some well known connections between BP^w, PBP^w, SK^w.

Lemma 1. *Let $w \in \mathbb{N}$. Then For any w, $\mathsf{PBP}^w \subseteq \mathsf{BP}^w$, for any $w \geq 5$, $\mathsf{BP}^w \subseteq \mathsf{PBP}^w$ [Bar89] and for any w, BP^w is contained in SK^{w^2} (see e.g. [RR10]).*

Definition 5. *(Approximate Majority) Approximate Majority* $\mathsf{ApproxMaj}_{a,n}$: $\{0,1\}^n \to \{0,1\}$ *is the promise problem defined as: $maths f ApproxMaj_{a,n}(x) :=$ 0 if x has at most a no. of 1s and is 1 if x has at least $n - a$ no. of 1s.*

3 Branching Programs and Skew Circuits

Here we analyze the conversion from branching programs to skew circuits and vice versa. First we recall the following folklore lemma.

Lemma 2. *(Folklore) Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function computed by a width-w length-ℓ branching program with at most k edges between any two consecutive layers and with the additional property that each layer reads at most one variable or its negation. Then there is a skew circuit of width max$\{w+2, k\}$ and size $O((k+w)\ell)$ computing f.*

3.1 Permutation Branching Programs to Skew Circuits

A permutation θ is called a *transposition* if either it is the identity permutation or there exists $i \neq j$ such that $\theta(i) = j$, $\theta(j) = i$ and for all $k \neq i \neq j$, $\theta(k) = k$. We call a transposition non-trivial if it is not the identity permutation, trivial otherwise.

[2] This is often called the strong acceptance condiction. Other notions of acceptance have been studied in the literature. See for example [Bro05].

[3] This assumption is not without loss of generality. However, we will see that when a branching program is converted into a skew circuits, exactly this type of skew circuits arise.

Definition 6. *(Transposition Branching Programs, TBP) A width-w length-ℓ TBP is a program given by a set of ℓ instructions in which for any $1 \leq i \leq \ell$, the ith instruction is a three tuple $\langle j_i, \theta^i, \sigma^i \rangle$, where j_i is an index from $\{1, 2, \ldots, |X|\}$, θ_i, σ_i are transpositions of $\{1, 2, \ldots, w\}$. The output of the instruction is θ_i if $x_{j_i} = 1$ and it is σ_i if $x_{j_i} = 0$. The output of the program on input x is the product of the output of each instruction of the program on x.*

Lemma 3. *Given a width-w PBP of length ℓ there is an equivalent width-w TBP of length $O(w\ell)$.*

Proof. It is known (see e.g. [Her06]) that any permutation of $\{1, 2, \ldots, w\}$ can be written as a product of W transpositions of $\{1, 2, \ldots, w\}$, where $W = O(w)$. Let P be a width-w PBP of length ℓ. Consider the ith instruction in the program, say $\langle j_i, \theta_i, \sigma_i \rangle$. We know that we can write θ_i as a product of W transpositions, i.e. $\theta_i = t_{i,1} \cdot t_{i,2} \ldots t_{i,W}$, where for $1 \leq j \leq W$ $t_{i,j}$ is a transposition. Similarly, we have $\sigma_i = s_{i,1} \cdot s_{i,2} \ldots s_{i,W}$, where $s_{i,j}$ is a transposition for $1 \leq j \leq W$.

To give a TBP equivalent to P, we replace every instruction $\langle j_i, \theta_i, \sigma_i \rangle$ in P by the following: $\langle j_i, t_{i,1}, id \rangle \cdot \langle j_i, t_{i,2}, id \rangle \ldots \langle j_i, t_{i,W}, id \rangle \cdot \langle j_i, id, s_{i,1} \rangle \cdot \langle j_i, id, s_{i,2} \rangle \ldots \langle j_i, id, s_{i,W} \rangle$. By a simple inductive argument we can prove that the the transposition branching program thus obtained is equivalent to P. As $W = O(w)$, the upper bound on the length of the resulting branching program follows.

We defined TBPs as a set of instructions. Like in the case of PBPs, the definition of TBPs can be rephrased in terms of the underlying DAG. We observe the following about the DAG resulting from TBPs.

A width-w TBP is a layered width w PBP in which the following conditions hold: (1) There are designated source vertices $s_1, s_2, \ldots, s_w$ in the first layer, say layer 1 (of in-degree 0) and sink vertices $t_1, t_2, \ldots, t_w$ (of out-degree 0) in the last layer, layer ℓ, (2) Each layer has exactly w vertices, (3) In each layer all the edges are labelled by a unique variable, (4) In each layer $1 \leq i \leq \ell$, one of the following holds: (4a) either the edges corresponding to $x_{j_i} = 1$ form a non-trivial transposition and the edges cooresponding to $x_{j_i} = 0$ form the identity permutation or (4b) the edges corresponding to $x_{j_i} = 0$ form a non-trivial transposition and the edges cooresponding to $x_{j_i} = 1$ form the identity permutation.

Remark 1. *As a result of the above properties of the TBP the total number of distinct edges between any two layers in a width-w TBP is at most $w + 2$: there are w edges corresponding to the identity permutation, 2 edges corresponding to the transposition of two elements, and $w - 2$ edges corresponding to the identity maps for all but the two transposed elements. The $w - 2$ last edges overlap with the w edges corresponding to the identity permutation.*

Lemma 4. $\mathsf{PBP}^w \subseteq \mathsf{SK}^{w+2}$.

Proof. Given a PBP^w for L of size s, by Lemma 3 we know that L also has a width-w TBP. By Remark 1 the underlying DAG for the TBP has at most $w+2$ edges between any two consecutive layers. Using Lemma 2 we get a skew circuit of width $w + 2$ for L. Note that the size of such a circuit is $O(ws)$.

Using Barrington's characterization of NC^1 and Lemma 4 we get the following: $\mathsf{NC}^1 = \mathsf{BP}^5 = \mathsf{PBP}^5 \subseteq \mathsf{SK}^7$.

3.2 Skew Circuits to Branching Programs

In this section we start from a skew circuit of bounded width and convert it into a branching program of bounded width. Formally, we prove the following:

Theorem 2. *If C is a skew circuit of width w and length ℓ then there is an equivalent branching program P of width w and size $O(w\ell)$.*

Proof. Recall that in a skew circuit C, AND gates have fan-in 2 and at least one child is an input variable whereas OR gates have arbitrary fan-in and arbitrary predecessors. Given a skew circuit C of width w and length ℓ_0 we will construct a branching program P of width w that will recognize the same language. Let $G_{i1}, \ldots, G_{iw}$ be the gates of C on layer i for $i = 1, \ldots, \ell_0$.

Let $X = \{\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_L}\}$ ($|X| = L$) be the set of layers on which there is at least one input gate. Without loss of generality we assume that in each $j \in [L]$ the gate $G_{\ell_{i_j} w}$ is an input gate. (There may be other input gates as well.)

We will construct a branching program of length $s = L + 2$ and width w. The nodes in the branching program in layer $\ell_{i_j} \in X$ will be called $N_{j0}, \ldots, N_{j(w-1)}$. The nodes N_{00} and $N_{(s-1)(w-1)}$ are respectively, initial and target nodes.

The nodes $N_{11}, \ldots, N_{(s-1)(w-1)}$ will, by our construction, compute the value of the nodes in a layer in X. More formally, for every input x, the gate $G_{\ell_j c}$ in layer ℓ_j of the circuit (and layer j in X) evaluates to 1 iff the node N_{jc} can be reached from the initial node. Since the gate G_{iw} in X is an input gate we will not add corresponding gate in the branching program. We have completely specified the vertex set of the branching program P.

We now describe the edge set of P. We add an edge from $N_{(j-1)0}$ to N_{j0} labeled by 1 for every $1 \leq j \leq s - 1$. This ensures that all nodes N_{j0} are always reachable from the initial node.

Suppose that the layer ℓ_j and $\ell_j + 1$ are both in X, i.e. $\ell_{j+1} = \ell_j + 1$, then the edges between the nodes in the layer ℓ_j and ℓ_{j+1} in the branching program are easy to state. A node N_{j+1c} is connected to N_{jd} if there is an edge between the corresponding gates $G_{\ell_{j+1} c}$ and $G_{\ell_j d}$. Also the edge in the branching program is labeled by 1 if the gate $G_{\ell_j d}$ is an OR gate, and labeled by the variable x_i (or its negtion $\neg x_i$) if $G_{\ell_j d}$ is an AND gate querying x_i, resp. $\neg x_i$. If an OR gate in ℓ_j is connected to an input gate, we generate an edge to N_{j0} labeled by the literal queried by the input gate.

Now assume that the layer ℓ_j is in X and ℓ_{j+1} is the next layer in X and $\ell_{j+1} > \ell_j + 1$. Then in the skew circuit, no input gates occur strictly between the layers ℓ_j and ℓ_{j+1}. This implies that there are no AND gates in the layers $\ell_j + 2, \ldots, \ell_{j+1}$. Hence the functions computed by the gates in layer ℓ_{j+1} are ORs of some gates in layer $\ell_j + 1$. In layer gates in layer $\ell_j + 1$ are ORs of either ANDs of gates in layer $\ell_j + 1$ and an input variable or ORs of directly gates in layer $\ell_j + 1$. Therefore, we add the following edges in the branching program:

a node $N_{(j+1)c}$ is connected to N_{jd} if the OR function computed by $G_{\ell_{j+1}c}$ has $G_{\ell_j d}$ as one of the inputs. This edge in the branching program is labeled by 1 if this was a direct OR, it is labeled by the variable x_i (or its negtion $\neg x_i$) it it was an 'or' of an 'and' querying x_i resp. $\neg x_i$. e.

It is easy to verify by induction on the layers that N_{jc} is reachable from the inital gate if the corresponding gate evaluates true. Finally we add an edge from the node corresponding to the output gate to $N_{(s-1)(w-1)}$.

Putting together Lemma 4 and Theorem 2 we get the following corollary:

Corollary 1. $\mathsf{NC}^1 = \mathsf{BP}^5 = \mathsf{PBP}^5 = \mathsf{SK}^7$

4 Width ≤ 7 Skew Circuits

Here we study the structure of the languages in NC^1 by investigating properties of skew circuits of width 7 or less. By definition $\mathsf{SK}^i \subseteq \mathsf{SK}^{i+1}$ for $1 \leq i \leq 6$. We start by proving that width 2 circuits are not universal (We defer the proofs to the full version of the paper due to lack of space).

Lemma 5. *A width 2 skew circuit of any size cannot compute Parity of 2 bits.*

Recall that a *k-DNF of size s on n variables* is an OR of s terms, where each term is an AND of at most k literals from $\{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\}$. Similarly, *k-CNF of size s on n variables* is an AND of s clauses, where each clause is an OR of at most k literals from $\{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\}$.

Lemma 6. *Let f be a k-DNF of size s on n variables. Then f has a width-3 skew circuit of length $O(sk)$.*

We defer the proof of the above lemma to the full version of the paper due to lack of space. As any Boolean function on n variables has an n-DNF of size at most 2^n, we get the following corollary.

Corollary 2. *Let $f : \{0,1\}^n \to \{0,1\}$. Then f can be computed by width-3 skew circuit of length $O(n2^n)$, i.e. width-3 skew circuits are universal.*

Lemma 7. *Let f be a k-CNF of size s on n variables. Then f has a width-3 skew circuit of length $O(sk)$.*

Proof. Note that in a CNF, the top AND gate gets clauses as inputs. That is, the AND gate is not skew. However, it is still possible to get a skew circuit for CNFs. We prove this by induction on the number of clauses, i.e. s. The base case is $s = 1$. This is just an OR of literals, which is computable by width-2 skew circuit of length $O(k)$. Let $f_i(x_1, \ldots, x_n) = C_1 \wedge \ldots \wedge C_i$ be computable by width-3 skew circuit of length $O(ik)$. Now $f_{i+1} = f_i \wedge C_{i+1}$ (where $C_{i+1} = x_{j_1} \vee x_{j_2} \ldots \vee x_{j_k}$) is computed as follows: $f_{i+1} = (\ldots((f_i \wedge x_{j_1}) \vee (f_i \wedge x_{j_2}) \ldots (f_i \wedge x_{j_k}))\ldots)$. Note that we need width 1 each for the f_i, the AND gate and the input variable. (Even though we require width 3 to compute f_i, after the computation, it just requires width 1 to carry around the value of the function to the next stage). (See Figure 1.) $\square$

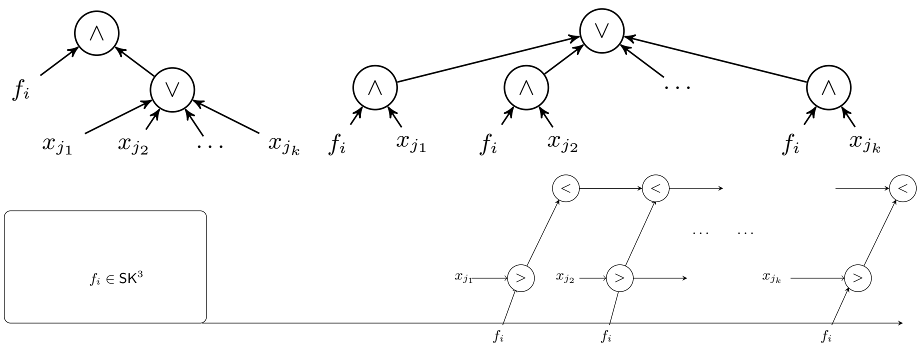


Fig. 1. Width 3 skew circuits for CNFs

By a result of Viola [Vio09], it is known that Approximate Majority is computable by P-uniform depth-3 circuits of polynomial size with alternating AND/OR layers with the output gate being an OR gate. This along with Lemma 7 above yields:

Corollary 3. $\mathsf{ApproxMaj}_{a,n}$ *has skew circuits of width* 4 *and polynomial length.*

Razborov[Raz87] and Smolensky[Smo87] show that Parity does not have constant depth circuit of polynomial size. It also implies that Parity does not have polynomial size CNFs or DNFs . However, we now show that Parity has skew circuits of width 4 and polynomial size.

Lemma 8. *Parity on n variables has a skew circuit of width* 4 *and length $O(n)$.*

Proof. This is an easy observation which comes from the fact that Parity has a branching program of width 2 and length $O(n)$. This fact along with part 3 of Lemma 1 proves the result. □

5 Parity and SK^3

In this section we prove that Parity does not have polynomial length width-3 skew circuits. As Parity has width 4 skew circuits of linear length (Lemma 8), this separates SK^3 from SK^4.

Theorem 3. $\mathsf{SK}^3 \subsetneq \mathsf{SK}^4$.

In order to prove this we first show that any width three skew circuit computing Parity can be converted into a normal form. We then show that any polynomial sized circuit of that normal form cannot compute Parity.

Lemma 9. *Let C be a Boolean width* 3 *skew circuit with size $s(n)$ computing* PARITY_n. *The circuit C can be converted into another circuit D such that D computes Parity of at least $n-2$ bits and has the following structure:*

1. *The top gate of D is an OR of at most $3s(n)^2$ disjoint skew circuits, say $C_1, C_2, \ldots, C_m$, where $m \leq 3s(n)^2$.*
2. *The sum of sizes of all C_is is at most $O(s(n)^3)$*
3. *At most two of these circuits have width 3 and all the other have width at most 2.*

Lemma 10. *Let D be a circuit satisfying properties 1,2,3 from Lemma 9 and computing* $\textsc{Parity}_n$. *Then there exists a constant c such that the size of D is at least $2^n/n^c$.*

Using Lemma 9 and Lemma 10 it is easy to see that the lower bound for Parity follows.

5.1 Proof of Lemma 9

Let C be a width three circuit computing Parity of n bits of size $s(n)$. The top gate of C cannot be AND as, by fixing the input wires of the AND gate, we can fix the output of the circuit, however, Parity of n bits cannot be fixed by fixing $< n$ bits. Therefore, we can assume that the top gate is an OR gate. We prove the following structural statement about the skew circuits (We defer the proof of the proposition to the full version of the paper due to lack of space):

Proposition 1. *Let C be any width three skew circuit computing Parity of n bits. Let k be the highest layer in C consisting of only AND gates, say g_k^1, g_k^2, g_k^3. We can convert this into another width 3 skew circuit D computing Parity of at least $n-2$ bits such that no layer of D contains three AND gates.*

Therefore, we will now assume that we have a circuit D which computes parity of at least $n-2$ bits in which no layer consists of only AND gates.

Let $G = (V, E)$ denote the DAG underlying the circuit D. Let $X \subseteq V$ denote the subset of gates which have a path to the top gate via only OR gates. Note that all the vertices in this set are themselves OR gates. We refer to the set of vertices X as ORSET.

Let $X_{in} \subseteq V$ denote the set of vertices in $V \setminus X$ which have an edge incident from it to some vertex in X. Similarly, let $X_{out} \subseteq V$ denote the set of vertices in $V \setminus X$ which have an edge incident to them from some vertex in X.

(a) Disconnect all edges incident to the set X_{out} from X. Let the new dangling wires be labelled with the constant 0 input.
(b) If after step (a) any AND gates receives constant input 0 then delete the gate and if any OR gate receives constant input 0 then delete this input of the OR gate.
(c) In the graph obtained after steps (a) and (b), consider $V \setminus X$. This disconnects the DAG G and gives rise to some connected components, say $X_1, X_2, \ldots, X_\ell$. For each $i \in [\ell]$ and edge (u, v) such that $u \in X_i$ and $v \in X$ let $C_{i,u}$ be the subgraph (DAG) of X_i with u as its sink.

Proposition 2. *The step (a) does not change the output function of the circuit.*

The proof of the above is deferred to the full version of the paper.Note that $V \backslash X$ is partitioned by X_is, $\therefore \ell \leq s(n)$. Also, $\cup_{i=1}^{m} X_i \subseteq V \backslash X$, hence $\sum_{i=1}^{m} |X_i| \leq s(n)$. The number of edges in G is $\leq 3s(n)$. Therefore, the total number of edges between any X_i and X is at most $3s(n)$ and the number of circuits $C_{u,i}$ is at most $s(n)^2$ and each such circuit is of size at most $s(n)$. This proves parts $1, 2$ of Lemma 9.

We will now prove part 3 of Lemma 9. The top gate of D is the same as the top gate of C, and it is an OR gate. Therefore, this gate is in the ORSET. Now as long as there are OR gates on the layers, we have at least one gate per layer in the ORSET. Finally, if there is a layer with no OR gates, then this layer must have at least one input variable (as D does not have any layer with three AND gates). The other two gates at this layer be g, h. Let X_g, X_h be two DAGs rooted at g and h, respectively, and C_g and C_h be the two corresponding circuits. These are possibly width three circuits. However, all other connected components of $V \setminus X$ are of width at most 2 due to step (b) above. This gives Part 3 of Lemma 9.

5.2 Proof of Lemma 10

Let D be the circuit given by Lemma 9. Let n_0 denote the number of unfixed variables. Let $C_1, C_2, \ldots, C_m$ be circuits given by Lemma 9. We know that at most two circuits among these have width 3. Let us assume without loss of generality that the two circuits are C_1, C_2. The output gates of these circuits are AND gates, say G_1 and G_2, respectively. These being skew circuits, all but one of the inputs of the AND gates are input gates. We will first prove the following proposition.

Proposition 3. *By fixing at most 2 variables both G_1 and G_2 can be set to 0.*

The proof of this proposition is deferred to the full version of the paper.

Let N denote the number of variables which were not set. Here, $N \geq n_0 - 2$. The new circuit, say D', is now an OR of $C_3, C_4, \ldots, C_m$ and by our assumption, it computes Parity of N variables. We will show that OR of polynomially many polynomial size width-2 skew circuits cannot compute Parity.

Let us fix some notation. Let $L_{\oplus}$ denote the 1 set of parity, i.e. $L_{\oplus} = \{x \in \{0,1\}^N \mid Parity(x) = 1\}$. We know that $|L_{\oplus}| = 2^{N-1}$. For any Boolean circuit C, let L_C denote $\{x \in \{0,1\}^N \mid C(x) = 1\}$. Note that as D' above is an OR of $C_3, C_4 \ldots, C_m$, we have $L_{D'} = \cup_{i=3}^{m} L_{C_i}$.

Definition 7. *We say that a Boolean circuit C α-*approximates *a function $f : \{0,1\}^n \to \{0,1\}$ if the following conditions hold:*

- *$\forall x \in \{0,1\}^n$, if $f(x) = 0$ then $C(x) = 0$, i.e. C has no false positives.*
- *The ratio of $|\{x \mid C(x) = 1\}|$ to $|\{x \mid f(x) = 1\}|$ is at least α*

For the sake of contradiction we have assumed that D' computes parity of N bits. Assuming this and from the fact that $L_{D'} = \cup_{i=3}^{m} L_{C_i}$, we get that there exists an $i \in \{3, 4, \ldots, m\}$ such that C_i $1/m$-approximates parity of N bits. We will now prove that no such C_i exists, which will give us the contradiction. Formally, we prove the following:

Claim. Let D' and $C_3, C_4, \ldots, C_m$ be defined as above. There does not exists $i \in \{3, 4, \ldots, m\}$ such that C_i $1/m$-approximates Parity of N bits.

Proof. Suppose there exists a C_i which $1/m$-approximates parity of N bits. Recall that C_i is a width 2 skew circuit. Let the last layer be L and the first layer be 1. Let $\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_t}$ be the layers in which there is one input gate, with ℓ_{i_1} being closest to layer 1 and ℓ_{i_t} being closest to layer L. (Note that, we can assume without loss of generality that layer 1 is the only layer which has two input gates.) Let the variables queried by these gates be $x_{i_1}, x_{i_2}, \ldots, x_{i_t}$, respectively. Let $h_{i_{t+1}}$ denote the output gate in layer L. Similarly, let h_{i_1} be the gate in layer ℓ_{i_1} (other than the input gate), h_{i_2} be the gate in layer ℓ_{i_2} (other than the input gate) and so on till h_{i_t} be the gate in layer ℓ_{i_t} (other than the input gate).

As there are no NOT gates in the circuit, $h_{i_{j+1}}$ is a monotone function of x_{i_j}, h_{i_j} for every $1 \leq j \leq t$. There is a unique value of x_{i_j}, say $b_{i_j} \in \{0, 1\}$, such that by setting $x_{i_j} = b_{i_j}$, $h_{i_{j+1}}$ becomes a non-trivial function of h_{i_j}. (This is because, there are at most 6 different monotone functions on two bits, two of which cannot occur in a minimal circuit. And the other four (AND, OR, NAND, NOR) have this property.)

Note that, the setting of $x_{i_t} = b_{i_t}$ will not fix the value of h_{i_t}. Suppose h_{i_t} gets fixed due to this setting. In that case, value of $h_{i_{t+1}}$ will also get fixed. Suppose the value of $h_{i_{t+1}}$ becomes 1, then for all settings of $x \neq x_{i_t}$, $h_{i_{t+1}}$ will continue to have value 1. But we have assumed that C_i has no false positives. Therefore, this is not possible. On the other hand, if the value of $h_{i_{t+1}}$ gets fixed to 0, then for all settings of variables $x \neq x_{i_t}$ the circuit will output 0. That is, for 2^{N-1} different inputs the circuit will output 0. However, we have assumed that the circuit outputs 1 for at least $2^{N-1}/m$ many inputs.

Assuming $x_{i_t} = b_{i_t}$ and $x_{i_t} \neq x_{i_{t-1}}$, we will repeat this argument for $x_{i_{t-1}}$. Let $x_{i_{t-1}} = b_{i_{t-1}}$ be the setting of $x_{i_{t-1}}$ which makes h_{i_t} a function of $h_{i_{t-1}}$. Suppose this setting of $x_{i_{t-1}}$ fixes h_{i_t} then that will inturn fix $h_{i_{t+1}}$. As before, to avoid false positives, the value of $h_{i_{t+1}}$ cannot be fixed to 1. And to ensure that the circuit evaluated to 1 on at least $2^{N-1}/m$ inputs, it cannot be fixed to 0.

In this way, we can repeat the argument for k distinct variables as long as $k < (N-1) - \lceil \log m \rceil$. Let k_0 be such that $k_0 = \omega(\log m)$ and $k_0 < (N-1) - \lceil \log m \rceil$. We fix k_0 distinct variables as above. But now note that any other setting of these k_0 variables fixes the value of $h_{i_{t+1}}$ to 0. Therefore, the circuit can be 1 on at most $O(2^{N-k_0})$ inputs. But this contradicts our assumption that $h_{i_{t+1}}$ evaluated to 1 on at least $2^N/m$ inputs from $L_\oplus$. □

6 Discussion

The above study provides a wide range of interesting questions, answers to which may improve our understanding of functions in NC^1. Namely, the questions regarding lower bounds for width k skew circuits for $4 \leq k \leq 6$. Some of these questions could be more tractable than the daunting question of proving lower bounds for NC^1 circuits. We conjecture that Majority is a function with respect to which width 4 circuits will have exponential lower bound. Towards proving such a result, it may be interesting to obtain a normal form for width 4 skew circuits. It may also be possible that any function in NC^1 is computable by width k circuits for $4 \leq k \leq 6$. Such a result will tighten the connection between branching programs and skew circuits.

References

[Bar85] Barrington, D.A.: Width-3 permutation branching programs. Technical Memo MIT/LCS/TM-293, Massachusetts Institute of Technology, Laboratory for Computer Science (1985)

[Bar89] Barrington, D.A.: Bounded-width polynomial-size branching programs can recognize exactly those languages in NC^1. Journal of Computer and System Sciences **38**, 150–164 (1989)

[BDFP86] Borodin, A., Dolev, D., Fich, F.E., Paul, W.: Bounds for width two branching programs. SIAM Journal on Computing **15**(2), 549–560 (1986)

[Bro05] Brodsky, A.: An impossibility gap between width-4 and width-5 permutation branching programs. Information Processing Letters **94**(4), 159–164 (2005)

[Her06] Herstein, I.N.: Topics in algebra. John Wiley & Sons (2006)

[Lee59] Lee, C.-Y.: Representation of switching circuits by binary-decision programs. Bell System Technical Journal **38**(4), 985–999 (1959)

[Mas76] Masek, W.J.: A fast algorithm for the string editing problem and decision graph complexity. PhD thesis, Massachusetts Institute of Technology (1976)

[Raz87] Razborov, A.A.: Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. Mathematical Notes **41**(4), 333–338 (1987)

[RR10] Rao, B.V.R.: A study of width bounded arithmetic circuits and the complexity of matroid isomorphism. [HBNI TH 17] (2010)

[Smo87] Smolensky, R.: Algebraic methods in the theory of lower bounds for boolean circuit complexity. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing, pp. 77–82. ACM (1987)

[Vio09] Viola, E.: On approximate majority and probabilistic time. Computational Complexity **18**(3), 337–375 (2009)

Correlation Bounds and #SAT Algorithms for Small Linear-Size Circuits

Ruiwen Chen[1](✉) and Valentine Kabanets[2]

[1] School of Informatics, University of Edinburgh, Edinburgh, UK
rchen2@inf.ed.ac.uk

[2] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
kabanets@cs.sfu.ca

Abstract. We revisit the gate elimination method, generalize it to prove correlation bounds of boolean circuits with Parity, and also derive deterministic #SAT algorithms for small linear-size circuits. In particular, we prove that, for boolean circuits of size $3n - n^{0.51}$, the correlation with Parity is at most $2^{-n^{\Omega(1)}}$, and there is a #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$; for circuit size $2.99n$, the correlation with Parity is at most $2^{-\Omega(n)}$, and there is a #SAT algorithm running in time $2^{n-\Omega(n)}$. Similar correlation bounds and algorithms are also proved for circuits of size almost $2.5n$ over the full binary basis B_2.

Keywords: Boolean circuit · Random restriction · Correlation bound · Satisfiability algorithm

1 Introduction

Connections between circuit lower bounds and efficient algorithms have been explicitly exploited in several recent breakthroughs. In particular, the "random restriction" technique, which was used to prove circuit lower bounds, was extended to get both satisfiability algorithms and average-case lower bounds for boolean formulas [6,16,17,22] and AC^0 circuits [2,13].

For de Morgan formulas, Santhanam [22] gave a #SAT algorithm running in time $2^{n-\Omega(n)}$ for formulas of linear size; the algorithm is based on a generalization of the "shrinkage under random restrictions" property, which was used to prove formula lower bounds [11,25]. Santhanam [22] observed that, one can define a random process of restrictions such that the formula size shrinks with high probability. This *concentrated shrinkage* implies not only #SAT algorithms but also correlation bounds. As shown in [22], a linear-size de Morgan formula has correlation at most $2^{-\Omega(n)}$ with Parity; the correlation of two n-input functions f and g is $|\mathbf{Pr}[f(x) = g(x)] - \mathbf{Pr}[f(x) \neq g(x)]|$, where x is chosen uniformly at random from $\{0,1\}^n$. Santhanam's algorithm was extended to $2^{n-n^{\Omega(1)}}$-time #SAT algorithms for de Morgan formulas of size $n^{2.49}$ in [6] and size $n^{2.63}$ in [7]. For formulas over the full binary basis B_2, Seto and Tamaki [24] extended [22]

DOI: 10.1007/978-3-319-21398-9_17

Table 1. Worst-case and average-case lower bounds for computing Parity

	Worst-Case Lower Bounds	Average-Case Upper / Lower Bounds	
AC^0	$s = \exp(n^{\theta(\frac{1}{d-1})})$ [10,26]	$\epsilon = 2^{-\Omega(n/(\log s)^{d-1})}$ [12]	
De Morgan formulas	$s = n^{2-\theta(1)}$ [25]	$\epsilon \geqslant 2^{-\Omega(n^2/s)}$	$\epsilon \leqslant 2^{-\Omega(n/\sqrt{s})}$ [1,21]
			$\epsilon \leqslant 2^{-\Omega(n/c^2)}$ for $s = cn$ [22]
U_2-circuits	$s = 3n - \theta(1)$ [23]	$\epsilon \geqslant 2^{-\Omega(3n-s)}$	$\epsilon \leqslant 2^{-\Omega((3n-s)^2/n)}$ [This work]

to give a $2^{n-\Omega(n)}$-time #SAT algorithm for B_2-formulas of linear size, and also showed that such formulas cannot approximately compute affine extractors.

On the other hand, Komargodski, Raz, and Tal [16,17] also used the concentrated shrinkage property to generalize the worst-case formula lower bounds to the average case. They gave an explicit function (computable in polynomial time) such that de Morgan formulas of size $n^{2.99}$ can compute correctly on at most $1/2 + 2^{-n^{\Omega(1)}}$ fraction of inputs. Combining the techniques in [6,17], one can get a randomized $2^{n-n^{\Omega(1)}}$-time #SAT algorithm for de Morgan formulas of size $n^{2.99}$.

1.1 Our Results and Techniques

In this work, we get correlation bounds and #SAT algorithms for general boolean circuits. We consider circuits over the full binary basis B_2 and circuits over the basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$.

We prove that, for U_2-circuits of size $3n - n^{\epsilon}$ for $\epsilon > 0.5$, the correlation with Parity is at most $2^{-n^{\Omega(1)}}$, and there is a #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$; for U_2-circuits of size $3n - \epsilon n$ for $\epsilon > 0$, the correlation is at most $2^{-\Omega(n)}$, and there is a #SAT algorithm running in time $2^{n-\Omega(n)}$. For B_2-circuits, we give a similar #SAT algorithm for circuits of size almost $2.5n$, and show the average-case hardness of computing affine extractors using such circuits.

Our correlation bounds of U_2-circuits with Parity are almost optimal, up to constant factors in the exponents. In fact, one can construct a U_2-circuit of size $3n-l$ which computes Parity on at least $1/2 + 2^{-\Omega(l)}$ fraction of inputs. Table 1 summarizes the known worst-case and average-case lower bounds against Parity for several restricted circuit models. Note that, for the average-case bounds, we express the correlation ϵ as a function of the circuit size s.

However, there is still a gap between our average-case lower bounds and the worst-case lower bounds. The best known worst-case explicit lower bound is $5n - o(n)$ for U_2-circuits [15,18], and $3n - o(n)$ for B_2-circuits [3].

For #SAT algorithms, there is a known algorithm for B_2-circuits by Nurk [20] which runs in time $O(2^{0.4058s})$ for circuits of size s. The running time of our algorithm for B_2-circuits is almost the same as Nurk's [20]. We are not aware of any #SAT algorithm for U_2-circuits.

Our Techniques. We extend the gate elimination method which was previously used to prove worst-case circuit lower bounds [3,9,15,18,23,28]. We define a random process of restrictions such that the circuit size shrinks with high probability. This is similar to the concentrated shrinkage approach for boolean formulas [6,16,17,22,24]. We analyze this random process using the concentration bound given by a variant of Azuma's inequality as in [6]. This analysis is then used to get both correlation bounds and #SAT algorithms. The same approach works for both U_2-circuits and B_2-circuits, although we need different rules on defining restrictions.

As a byproduct of our algorithms, we show that small linear-size circuits have decision trees of non-trivial size. In particular, U_2-circuits of size s have equivalent decision trees of size $2^{n-\Omega((3n-s)^2/n)}$, and B_2-circuits of size s have parity decision trees of size $2^{n-\Omega((2.5n-s)^2/n)}$. Our correlation bounds follow directly from such non-trivial decision-tree representations.

Related Work. For U_2-circuits, the best known worst-case lower bound is $5n - o(n)$ by Iwama and Morizumi [15], improving upon a $4.5n - o(n)$ lower bound by Lachish and Raz [18], a $4n - c$ lower bound against symmetric functions by Zwick [28], and a $3n - c$ lower bound against Parity by Schnorr [23]. For B_2-circuits, the best known worst-case lower bound is $3n - o(n)$ by Blum [3]; Demenkov and Kulikov [9] gives an alternative proof of this lower bound against affine dispersers. Nurk [20] gave a satisfiability algorithm in time $O(2^{0.4058s})$ for B_2-circuits of size s. Nurk's algorithm [20] is also based on gate elimination and the running time is similar to ours, although we use a slightly different case analysis for gate elimination. We are not aware of any previous average-case lower bounds (correlation bounds) for general circuits.

2 Preliminaries

2.1 Circuits

Let B be a binary basis, i.e., a set of boolean functions on two variables. A *B-circuit* on n input variables is a directed acyclic graph with (1) nodes of in-degree 0 labeled by variables or constants, which we call *inputs*, and (2) nodes of in-degree 2 labeled by functions from B, which we call *gates*. There is a single node of out-degree 0, designated as the *output*. Without loss of generality, we assume, for each variable x_i, there is at most one input labeled by x_i. A circuit on n variables computes a boolean function $f\colon \{0,1\}^n \to \{0,1\}$. For two nodes u and v, we will write $u \to v$ if u feeds into v.

We consider two binary bases: the full basis B_2, which contains all boolean functions on two variables, and the basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$. Specifically, the basis B_2 contains the following 16 functions $f(x, y)$: (1) six degenerate functions: 0, 1, x, $\neg x$, y, $\neg y$; (2) eight $\wedge$-type functions: $x \wedge y$, $x \vee y$, and the variations by negating one or both inputs; (3) two $\oplus$-type functions: $x \oplus y$, $x \equiv y$.

The *size* of a circuit C, denoted by $s(C)$, is the number of gates in C. The *circuit size* of a function $f\colon \{0,1\}^n \to \{0,1\}$ is the minimal size of a boolean circuit computing f. For convenience, we define $\mu(C) = s(C) + N(C)$, where

$N(C)$ is the number of inputs that C depends on. We let $\mu(C) = 0$ if C is constant, and $\mu(C) = 1$ if C is a literal.

A *restriction* ρ is a mapping from the input variables to $\{0, 1, *\}$. For a circuit C, the restricted circuit $C|_\rho$ is obtained by fixing $x_i = b$ for all x_i such that $\rho(x_i) = b \in \{0, 1\}$.

It is convenient to work with circuits without redundant nodes or wires. We will call a non-constant circuit (over U_2 or B_2) *simplified* if it does not have the following: (1) nodes labeled by constants, (2) gates labeled by degenerate functions, (3) non-output gates with out-degree 0, or (4) any input x and two gates u, v with three wires $x \to u$, $x \to v$, $u \to v$.

Lemma 1. *For any circuit C, there is a polynomial-time algorithm transforming C into an equivalent simplified circuit C' such that $s(C') \leqslant s(C)$ and $\mu(C') \leqslant \mu(C)$.*

Proof (Sketch). Cases (1)-(3) are trivial. For case (4), suppose w is the other node feeding into u. If C is over B_2, then v computes a binary function of x and w; if C is over U_2, then v computes an $\wedge$-type function of x and w (because a $\oplus$-type function requires at least 3 gates). In either case, we can connect w directly to v, remove the wire $u \to v$, and change the gate label of v. By checking through each input and gate, the transformation can be done in polynomial time. □

2.2 Correlation

Definition 1. *Let f and g be two boolean functions on n input variables. The* correlation *of f and g is defined as*

$$Corr(f, g) = |\mathbf{Pr}[f(x) = g(x)] - \mathbf{Pr}[f(x) \neq g(x)]| = |2\mathbf{Pr}[f(x) = g(x)] - 1|,$$

where x is chosen uniformly at random from $\{0, 1\}^n$.

The *correlation* of f with a circuit class $\mathcal{C}$ is the maximum of $\mathrm{Corr}(f, C)$ for any $C \in \mathcal{C}$. Note that, a circuit C has correlation c with f if and only if C computes f or its negation correctly on a fraction $(1 + c)/2$ of all inputs. The correlation bound is also referred to as the *average-case lower bound* in the literature.

2.3 Decision Tree

A *decision tree* is a tree where (1) each internal node is labeled by a variable x, and has two outgoing edges labeled by $x = 0$ and $x = 1$, and (2) each leaf is labeled by a constant 0 or 1. A decision tree computes a boolean function by tracking the paths from the root to leaves. The *size* of a decision tree is the number of leaves of the tree.

A *parity decision tree* extends a decision tree such that each internal node is labeled by the parity of a subset of variables (including one single variable as a special case). We insist that, for each path from the root to a leaf, the parities appearing in the internal nodes are linearly independent.

2.4 Concentration Bounds

A sequence of random variables $X_0, X_1, \ldots, X_n$ is called a *supermartingale* with respect to a sequence of random variables $R_1, \ldots, R_n$ if $\mathbf{E}[X_i \mid R_{i-1}, \ldots, R_1] \leqslant X_{i-1}$, for $1 \leqslant i \leqslant n$. The following is a variant of Azuma's inequality which holds for supermartingales with one-side bounded differences.

Lemma 2 ([6]). *Let $\{X_i\}_{i=0}^n$ be a supermartingale with respect to $\{R_i\}_{i=1}^n$. Let $Y_i = X_i - X_{i-1}$. If, for every $1 \leqslant i \leqslant n$, the random variable Y_i (conditioned on $R_{i-1}, \ldots, R_1$) assumes two values with equal probability, and there exists $c_i \geqslant 0$ such that $Y_i \leqslant c_i$, then, for any $\lambda \geqslant 0$, we have $\mathbf{Pr}[X_n - X_0 \geqslant \lambda] \leqslant \exp\left(-\frac{\lambda^2}{2\sum_{i=1}^n c_i^2}\right)$.*

3 U_2-circuits

All known lower bounds for U_2-circuits [15,18,23,28] were proved using the gate elimination method. We will generalize this method by defining a random process of restrictions under which the circuit size reduces with high probability. This allows us to get a #SAT algorithm for U_2-circuits of size almost $3n$, and also prove a correlation bound against Parity.

3.1 Concentrated Shrinkage Under Restrictions

We call an $\wedge$-type function of two variables a *twig*. We now define a random process of restrictions where, at each step, we pick a variable or a twig and randomly assign it a value 0 or 1; we also simplify the circuit by eliminating unnecessary gates. The choice of variables or twigs at each step is determined by the following cases: (1) If the circuit is a literal, choose the variable in the literal. (2) If there is an input x with out-degree at least two, choose x. (3) Otherwise, there must be a gate u fed by two variables having out-degree 1; we choose u (which is a twig).

Let C be a simplified U_2-circuit on inputs $x_1, \ldots, x_n$. Let C' be the simplified circuit obtained after one step of restriction. Then we have the following lemma on the reduction of $\mu(C)$.

Lemma 3. *Suppose $\mu(C) \geqslant 4$. Let $\sigma = \mu(C) - \mu(C')$. Then we have $\sigma \geqslant 3$, and $\mathbf{E}[\sigma] \geqslant 4$.*

Proof. Consider the following cases (see also Figure 3.1):

(1) Suppose there is an input x_i feeding into two gates u and v. By Lemma 1, there is no edge between u and v. We randomly assign 0 or 1 to x_i, and consider the following sub-cases on the successors of u and v.

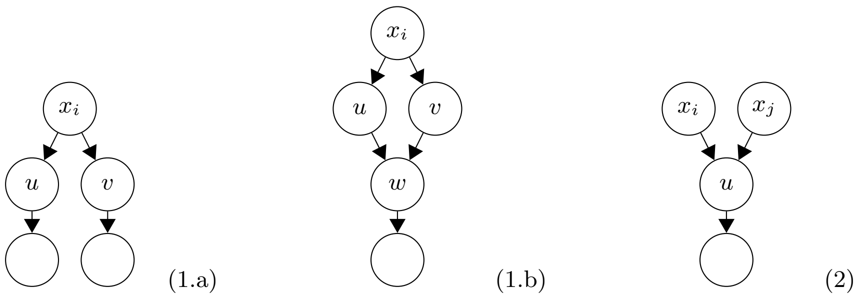


Fig. 1. Cases in Lemma 3

(a) If u and v feed into two different successors, we have the following possibilities. If under one assignment to x_i, none of u, v become constants, then we can eliminate x_i, u, v; and under the other assignment to x_i, since both of u, v will be constants, we can eliminate two more gates (successors of u, v); thus we have $\mathbf{Pr}[\sigma \geqslant 5] \geqslant 1/2$, and $\sigma \geqslant 3$. If under each assignment to x_i, only one of u, v becomes a constant, then we can eliminate x_i, u, v and one successor; thus $\sigma \geqslant 4$.

(b) If u and v feed into one single common successor w, we have similar situations as above. If under one assignment to x_i, both u and v become constants, then we can eliminate x_i, u, v, w and a successor of w; and under the other assignment to x_i, we can eliminate x_i, u, v. If under each assignment to x_i, only one of u, v becomes a constant, then we can eliminate x_i, u, v, w.

(2) If all inputs have out-degree 1, find a gate u fed by two inputs, say x_i and x_j. We randomly assign 0 and 1 to u; for each assignment, eliminate x_i, x_j, u and at least one successor of u. Then we have $\sigma \geqslant 4$.

In all cases, we have $\sigma \geqslant 3$, and $\mathbf{E}[\sigma] \geqslant 4$. □

Next consider the reduction of $\mu(C)$ under a sequence of restrictions. Let $C_0 := C$, and, for $i = 1, \ldots, d$, let C_i be the circuit obtained after the i-th step. For convenience, we let $\mu_i := \mu(C_i)$. Let R_i be the random value assigned to the variable or twig at each step. We define a sequence of random variables $\{Z_i\}$ as follows:

$$Z_i = \begin{cases} \mu_i - (\mu_{i-1} - 4), & \mu_{i-1} \geqslant 4, \\ 0, & \mu_{i-1} < 4. \end{cases}$$

Note that $0 < \mu_{i-1} < 4$ holds only when C_{i-1} itself is a literal or a twig, which means C_i will be a constant.

Lemma 4. *Let $X_0 = 0$ and $X_i = \sum_{j=1}^{i} Z_i$. Then we have $Z_i \leqslant 1$, and $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$.*

Proof. By Lemma 3, conditioning on $R_1, \ldots, R_{i-1}$, when $\mu_{i-1} \geqslant 4$, we have $\mu_i \leqslant \mu_{i-1} - 3$ and $\mathbf{E}[\mu_i] \leqslant \mu_{i-1} - 4$. Therefore, we get $Z_i \leqslant 1$, $\mathbf{E}[Z_i \mid R_{i-1}, \ldots, R_1] \leqslant 0$, and $\mathbf{E}[X_i \mid R_{i-1}, \ldots, R_1] \leqslant X_{i-1}$. Thus $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$. □

Lemma 5. *For $\lambda \geqslant 0$, $\mathbf{Pr}\left[\mu_d \geqslant \max\{\mu_0 - 4d + \lambda, 1\}\right] \leqslant \exp(-\lambda^2/2d)$.*

Proof. Conditioning on $R_1, \ldots, R_{i-1}$, the variable Z_i assumes two values with equal probability. By Lemma 4, we have $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$, and $Z_i \leqslant c_i \equiv 1$. Applying the bound in Lemma 2, we have

$$\mathbf{Pr}\left[\sum_{i=1}^{d} Z_i \geqslant \lambda\right] \leqslant \exp\left(-\frac{\lambda^2}{2d}\right).$$

When $\mu_d > 0$, we have $\sum_{i=1}^{d} Z_i = \mu_d - \mu_0 + 4d$. Let E_1 be the event that $\mu_d > 0$; let E_2 be the event that $\sum_{i=1}^{d} Z_i \geqslant \lambda$. Then the final probability is $\mathbf{Pr}[E_1 \wedge E_2] \leqslant \mathbf{Pr}[E_2] \leqslant \exp(-\lambda^2/2d)$. □

3.2 #SAT Algorithms

We now give a #SAT algorithm for circuits of size almost $3n$ based on the concentrated reduction of circuit size.

Theorem 1. *For U_2-circuits of size $s < 3n$, there is a deterministic #SAT algorithm running in time $2^{n - \Omega((3n-s)^2/n)}$.*

Proof. Let C be a circuit on n inputs $x_1, \ldots, x_n$ with size $s < 3n$. Let $\mu_0 := \mu(C) \leqslant s + n$. We use the following procedure to construct a generalized decision tree, where each internal node is labeled by a variable or a twig. We start with the root node and C.

- If C is a constant, label the current node by this constant and return.
- Use the cases in Lemma 3 to find either a variable or a twig; denote it by u. Label the current node by u.
- Build two outgoing edges labeled by $u = 0$ and $u = 1$. For each child node, simplify the circuit, and recurse.

We say a complete assignment to $x_1, \ldots, x_n$ is *consistent* with a path (from the root to a leaf) if it satisfies the restrictions along the path. Since each assignment $a \in \{0,1\}^n$ is consistent with only one path, the paths give a disjoint partitioning of the boolean cube $\{0,1\}^n$. To count the number of satisfying assignments for C, one can count for each path with leaf labeled by 1, and return the summation. Restrictions along each path is essentially a read-once 2-CNF, for which counting is easy. We next only need to bound the size of the tree.

We wish to bound the probability that a random path has length larger than $n - k$, for k to be chosen later. Let $\lambda = 4(n-k) - \mu_0 + 1$. Then by Lemma 5, at depth $n - k$, the restricted circuit becomes a constant with probability at least

$1 - \exp(-\lambda^2/2(n-k)) \geqslant 1 - 2^{-c\lambda^2/n}$ for a constant $c > 0$. The total number of paths with length larger than $n-k$ is at most $2^{n-k} \cdot 2^{-c\lambda^2/n} \cdot 2^k \leqslant 2^{n-c\lambda^2/n}$. Therefore, the size of the tree is at most $2^{n-k} + 2^{n-c\lambda^2/n}$. Choosing $k = (3n-s)/8$, both the tree size and the running time of the counting algorithm are bounded by $2^{n-\Omega((3n-s)^2/n)}$. □

The following corollary is immediate.

Corollary 1. *(1) For U_2-circuits of size $3n - \epsilon n$ with $\epsilon > 0$, there is a deterministic #SAT algorithm running in time $2^{n-\Omega(n)}$. (2) For U_2-circuits of size $3n - n^\epsilon$ with $\epsilon > 0.5$, there is a deterministic #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$.*

3.3 Correlation with Parity

Schnorr [23] proved a $3n-c$ lower bound for computing Parity using the following fact: a simplified U_2-circuit computing Parity cannot have any input variable with out-degree exactly 1. Indeed, if such an input x exists, one can fix all other variables such that the gate fed by x becomes a constant, but this makes the function independent of x, which is impossible for Parity.

We next generalize this lower bound to the average case by showing that a U_2-circuit of size $s < 3n$ cannot approximate well with Parity. The proof is by converting the generalized decision tree constructed in the proof of Theorem 1 into a normal decision tree without twigs, and argue that the tree size will not increase too much, as stated in the next lemma (the proof is left in the full version [5] of the paper).

Lemma 6. *Any function computed by a U_2-circuit of size $s < 3n$ has a decision tree of size $2^{n-\Omega((3n-s)^2/n)}$.*

The following lemma gives a simple relationship between the size of a decision tree and its correlation with Parity. It was previously used to derive correlation bounds for de Morgan formulas [22] and AC^0 circuits [13].

Lemma 7 ([22]). *A decision tree of size 2^{n-k} has correlation at most 2^{-k} with Parity.*

Theorem 2. *Let C be a U_2-circuit of size $s < 3n$. Then its correlation with Parity is at most $2^{-\Omega((3n-s)^2/n)}$. In particular, for $s = 3n - \epsilon n$ with $\epsilon > 0$, the correlation is at most $2^{-\Omega(n)}$; for $s = 3n - n^\epsilon$ with $\epsilon > 0.5$, the correlation is at most $2^{-n^{\Omega(1)}}$.*

Proof. The proof is immediate by Lemmas 6 and 7. □

The above correlation bounds with Parity almost match with the upper bounds. To see this, we can construction an approximate circuit for Parity in the following way. Divide n inputs into l groups each of size n/l, use circuits of size $3(n/l-1)$ to compute Parity exactly for each group, and then take the disjunction of the outputs from all groups. This circuit outputs 0 with probability 2^{-l}, but whenever it outputs 0, it agrees with Parity. Thus its correlation with Parity is at least 2^{-l}. The circuit size is $3(n/l-1) \cdot l + l = 3n - 2l$.

4 B_2-circuits

In this section, we give #SAT algorithms and correlation bounds for B_2-circuits of size almost $2.5n$.

4.1 Concentrated Shrinkage and #SAT Algorithms

Given a simplified B_2-circuit C, we will construct a generalized parity decision tree, where each internal node is labeled by either a twig or a parity of a subset of variables. Starting from the root with the given circuit C, we use the following case analysis to identify labels and build branches recursively.

If the circuit becomes a constant, we label the current node by the constant; then this node is a leaf. If the circuit is a literal or a gate fed by two variables, then we choose the variable of the literal or the circuit itself as the label, and build two branches. Otherwise, consider a topological order on the gates of the circuit, and let u be the first gate which is either $\oplus$-type of out-degree at least 2 or $\wedge$-type. Consider the following cases (see also Figure 4.1):

(1) If u is a $\oplus$-type gate of out-degree at least 2, then it computes $\oplus_{i\in I}x_i$ (or its negation) for some subset $I \subseteq [n]$. We choose $\oplus_{i\in I}x_i$ as the label, and build two branches; for the branch $\oplus_{i\in I}x_i = b \in \{0,1\}$, we replace u by a constant, and substitute an arbitrary variable x_j for $j \in I$ by a sub-circuit $\oplus_{i\in I\setminus\{j\}}x_i \oplus b$. In both branches, we can eliminate one variable x_j, and at least 3 gates (u and its two successors).

(2) If u is an $\wedge$-type gate fed by some $\oplus$-type gate v, suppose w is the other node feeding into u.
   - If w has out-degree 1, then we choose the parity function computed at v as the label, and build two branches similar to Case (1). In one branch, we can eliminate some input x_j and two gates v, u; in the other branch, we can eliminate two more nodes: w and a successor of u.
   - If w has out-degree at least 2, then it must be a variable. We choose w as the label, and build two branches. In one branch, we can eliminate w and its two successors; in the other branch, we can eliminate two more gates: v and a successor of u.

(3) If u is an $\wedge$-type gate fed by two inputs x_i and x_j where at least one of them, say x_i, has out-degree at least 2, then we choose x_i as the label and build two branches. In one branch, we can eliminate x_i and its two successors; in the other branch, we can eliminate one more gate: a successor of u.

(4) If u is an $\wedge$-type gate fed by two inputs each of out-degree 1, then choose the twig computed at u as the label. In both branches, we can eliminate x_i, x_j, u and a successor of u.

Consider a random path from the root of the decision tree to its leaves. Let $C_0 := C$, and let C_i be the restricted circuit obtained at depth i. Let $\mu_i := \mu(C_i)$. The next lemma follows directly from the above case analysis.

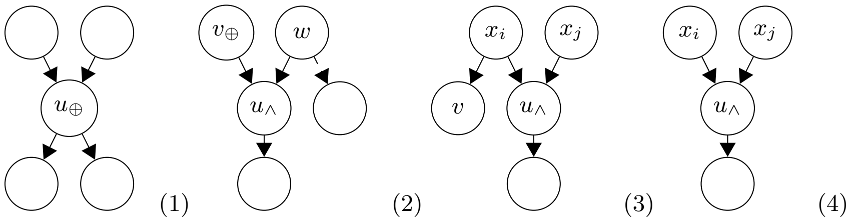


Fig. 2. Cases for eliminating gates in B_2-circuits

Lemma 8. *If $\mu_i > 4$, then $\mu_i - \mu_{i+1} \geqslant 3$, and $\mathbf{E}[\mu_i - \mu_{i+1}] \geqslant 3.5$. If $\mu_i \leqslant 4$, then $\mu_{i+1} = 0$.*

Then we have the following concentrated shrinkage.

Lemma 9. *For $\lambda \geqslant 0$, $\mathbf{Pr}\left[\mu_d \geqslant \max\{\mu_0 - 3.5d + \lambda, 1\}\right] \leqslant \exp(-\lambda^2/2d)$.*

Theorem 3. *For B_2-circuits of size $s < 2.5n$, there is a deterministic #SAT algorithm running in time $2^{n-\Omega((2.5n-s)^2/n)}$. In particular, for $s = 2.5n - \epsilon n$ with $\epsilon > 0$, the algorithm runs in time $2^{n-\Omega(n)}$; for $s = 2.5n - n^\epsilon$ with $\epsilon > 0.5$, the algorithm runs in time $2^{n-n^{\Omega(1)}}$.*

We omit the proofs of Lemma 9 and Theorem 3 since they are similar to the proofs of Lemma 5 and Theorem 1.

4.2 Correlation Bounds

Demenkov and Kulikov [9] proved that affine dispersers for sources of dimension d requires B_2-circuits of size $3n - \Omega(d)$. We next extend this result to the average case by showing that affine extractors have small correlations with B_2-circuits of size less than $2.5n$.

Definition 2. *Let F_2 be the finite field with elements $\{0,1\}$. A function $\mathsf{AE}\colon F_2^n \to F_2$ is a (k,ϵ)-*affine extractor *if for any uniform distribution X over some k-dimensional affine subspace of F_2^n, $|\mathbf{Pr}[\mathsf{AE}(X) = 1] - 1/2| \leqslant \epsilon$.*

We will need the following constructions of affine extractors.

Theorem 4 ([4,19,27]). *(1) For any $\delta > 0$ there exists a polynomial-time computable (k,ϵ)-affine extractor $\mathsf{AE}_1\colon \{0,1\}^n \to \{0,1\}$ with $k = \delta n$ and $\epsilon = 2^{-\Omega(n)}$. (2) There exists a constant $c > 0$ and a polynomial-time computable (k,ϵ)-affine extractor $\mathsf{AE}_2\colon \{0,1\}^n \to \{0,1\}$ with $k = cn/\sqrt{\log\log n}$ and $\epsilon = 2^{-n^{\Omega(1)}}$.*

We will prove our correlation bounds using the following representation of B_2-circuits by parity decision trees.

Lemma 10. *Any function computed by a B_2-circuit of size $s < 2.5n$ is computable by a parity decision tree of size $2^{n-\Omega((2.5n-s)^2/n)}$.*

The proof, which we omit here, is almost the same as the proof of Lemma 6. That is, using the algorithm in Theorem 3, one can construct a generalized parity decision tree which may have twigs, and then expand the twigs and argue that the tree size does not increase much. Note that, when we restrict a twig, the two variables in the twig are completely eliminated; when we restrict a parity, since one variable is substituted, all parity restrictions are linearly independent.

The following lemma gives the correlation of (relatively small) parity decision trees with affine extractors given in Theorem 4. It was implicit in [24] and was also given in [8].

Lemma 11 ([8,24]). *(1) For any $\delta > 0$, a parity decision tree of size 2^{n-k} for $k = \delta n$ has correlation at most $2^{-\Omega(n)}$ with AE_1. (2) There is a constant $c > 0$ such that a parity decision tree of size 2^{n-k} for $k = cn/\sqrt{\log\log n}$ has correlation at most $2^{-n^{\Omega(1)}}$ with AE_2.*

The next theorem follows by Lemma 10 and Lemma 11.

Theorem 5. *(1) For any $\delta > 0$ and B_2-circuit of size $2.5n - \delta n$, its correlation with AE_1 is at most $2^{-\Omega(n)}$. (2) There exists a constant $c > 0$ such that, for any B_2-circuit of size $2.5n - cn/\sqrt[4]{\log\log n}$, its correlation with AE_2 is at most $2^{-n^{\Omega(1)}}$.*

5 Open Questions

It is open whether our correlation bounds (for the size almost $3n$ for U_2-circuits, and almost $2.5n$ for B_2-circuits) can be improved to match with the best known worst-case lower bounds (for the size almost $5n$ for U_2-circuits, and almost $3n$ for B_2-circuits). Pseudorandom generators for boolean formulas were constructed in [14] based on concentrated shrinkage and decomposition of the formula tree. It would be interesting to get pseudorandom generators for general boolean circuits.

References

1. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. JACM **48**(4), 778–797 (2001)
2. Beame, P., Impagliazzo, R., Srinivasan, S.: Approximating ac^0 by small height decision trees and a deterministic algorithm for $\#ac^0$ sat. In: Proceedings of the 2012 IEEE Conference on Computational Complexity, CCC 2012 (2012)
3. Blum, N.: A Boolean function requiring $3n$ network size. Theoretical Computer Science **28**, 337–345 (1984)
4. Bourgain, J.: On the construction of affine-source extractors. Geometric and Functional Analysis **17**(1), 33–57 (2007)
5. Chen, R., Kabanets, V.: Correlation bounds and #sat algorithms for small linear-size circuits. ECCC **21**, 184 (2014)

6. Chen, R., Kabanets, V., Kolokolova, A., Shaltiel, R., Zuckerman, D.: Mining circuit lower bound proofs for meta-algorithms. In: CCC 2014 (2014)
7. Chen, R., Kabanets, V., Saurabh, N.: An improved deterministic #SAT algorithm for small de Morgan formulas. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 165–176. Springer, Heidelberg (2014)
8. Cohen, G., Shinkar, I.: The complexity of DNF of parities. ECCC **21**, 99 (2014)
9. Demenkov, E., Kulikov, A.S.: An elementary proof of a $3n-o(n)$ lower bound on the circuit complexity of affine dispersers. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 256–265. Springer, Heidelberg (2011)
10. Håstad, J.: Almost optimal lower bounds for small depth circuits. In: STOC 1986, pp. 6–20 (1986)
11. Håstad, J.: The shrinkage exponent of de Morgan formulae is 2. SIAM Journal on Computing **27**, 48–64 (1998)
12. Håstad, J.: On the correlation of parity and small-depth circuits. ECCC **19**, 137 (2012)
13. Impagliazzo, R., Matthews, W., Paturi, R.: A satisfiability algorithm for AC^0. In: SODA 2012, pp. 961–972 (2012)
14. Impagliazzo, R., Meka, R., Zuckerman, D.: Pseudorandomness from shrinkage. In: FOCS 2012, pp. 111–119 (2012)
15. Iwama, K., Morizumi, H.: An explicit lower bound of $5n-o(n)$ for boolean circuits. In: MFCS 2002, pp. 353–364 (2002)
16. Komargodski, I., Raz, R.: Average-case lower bounds for formula size. In: STOC 2013, pp. 171–180 (2013)
17. Komargodski, I., Raz, R., Tal, A.: Improved average-case lower bounds for demorgan formula size. In: FOCS 2013, pp. 588–597 (2013)
18. Lachish, O., Raz, R.: Explicit lower bound of $4.5n$–o(n) for boolena circuits. In: STOC 2001, pp. 399–408. ACM, New York (2001)
19. Li, X.: A new approach to affine extractors and dispersers. In: CCC 2011, pp. 137–147 (2011)
20. Nurk, S.: An $o(2^{0.4058m})$ upper bound for circuit sat. PDMI Preprint (2009)
21. Reichardt, B.: Reflections for quantum query algorithms. In: SODA 2011, pp. 560–569 (2011)
22. Santhanam, R.: Fighting perebor: new and improved algorithms for formula and qbf satisfiability. In: FOCS 2010, pp. 183–192 (2010)
23. Schnorr, C.: Zwei lineare untere schranken für die komplexität boolescher funktionen. Computing **13**(2), 155–171 (1974)
24. Seto, K., Tamaki, S.: A satisfiability algorithm and average-case hardness for formulas over the full binary basis. In: CCC 2012, pp. 107–116 (2012)
25. Subbotovskaya, B.A.: Realizations of linear functions by formulas using and or, not. Soviet Math. Doklady **2**, 110–112 (1961)
26. Yao, A.C.: Separating the polynomial-time hierarchy by oracles. In: FOCS 1985, pp. 1–10 (1985)
27. Yehudayoff, A.: Affine extractors over prime fields. Combinatorica **31**(2), 245–256 (2011)
28. Zwick, U.: A 4n lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic boolean functions. SIAM J. Comput. **20**(3), 499–505 (1991)

Commuting Quantum Circuits with Few Outputs are Unlikely to be Classically Simulatable

Yasuhiro Takahashi[1(✉)], Seiichiro Tani[1], Takeshi Yamazaki[2], and Kazuyuki Tanaka[2]

[1] NTT Communication Science Laboratories, NTT Corporation, Atsugi 243-0198, Japan
{takahashi.yasuhiro,tani.seiichiro}@lab.ntt.co.jp
[2] Mathematical Institute, Tohoku University, Sendai 980-8578, Japan
{yamazaki,tanaka}@math.tohoku.ac.jp

Abstract. We study the classical simulatability of commuting quantum circuits with n input qubits and $O(\log n)$ output qubits, where a quantum circuit is classically simulatable if its output probability distribution can be sampled up to an exponentially small additive error in classical polynomial time. Our main result is that there exists a commuting quantum circuit that is not classically simulatable unless the polynomial hierarchy collapses to the third level. This is the first formal evidence that a commuting quantum circuit is not classically simulatable even when the number of output qubits is exponentially small. Then, we consider a generalized version of the circuit and clarify the condition under which it is classically simulatable. We apply these results to examining the ability of IQP circuits to implement fundamental operations, and to examining the classical simulatability of slightly extended Clifford circuits.

1 Introduction and Summary of Results

One of the most important challenges in quantum information processing is to understand the difference between quantum and classical computation. An approach to meeting this challenge is to study the classical simulatability of quantum computation. Previous studies have shown that restricted models of quantum computation, such as commuting quantum circuits, contribute to this purpose [1,2,4,6,7,9,10]. Because of the simplicity of such restricted models, they also contribute to identifying the source of the computational power of quantum computers. It is thus of interest to study their classical simulatability.

We study the classical simulatability of commuting quantum circuits with n input qubits and $O(\mathrm{poly}(n))$ ancillary qubits initialized to $|0\rangle$, where a commuting quantum circuit is a quantum circuit consisting of pairwise commuting gates, each of which acts on a constant number of qubits. When every gate in a commuting quantum circuit acts on at most c qubits, the circuit is said to be c-local. A commuting quantum circuit is a restricted model of quantum computation in the

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 223–234, 2015.
DOI: 10.1007/978-3-319-21398-9_18

sense that all the gates in the circuit can be applied in an arbitrary order. Moreover, there exists a basis in which all the gates are diagonal and thus, under some conditions, they can be implemented simultaneously [5]. In spite of these severe restrictions, as mentioned below, there are evidences that commuting quantum circuits are not classically simulatable in various settings [2,7]. This remarkable feature makes such circuits particularly interesting for study.

For considering the classical simulatability, we adopt strong and weak simulations. The strong simulation of a quantum circuit is to compute its output probability in classical polynomial time and the weak one is to sample its output probability distribution likewise. Any strongly simulatable quantum circuit is easily shown to be weakly simulatable. Our main focus is on the hardness of classically simulating quantum circuits and thus we mainly consider weak simulatability, which yields a stronger result. Previous hardness results on the weak simulatability are usually obtained with respect to multiplicative errors [2,6,10]. They can usually be turned into hardness results with respect to exponentially small additive errors, although in general it is difficult to exactly determine the relative strength of these error settings. In this paper, we deal with exponentially small additive errors. We note that our hardness results in this paper can be turned into hardness results with respect to multiplicative errors.

Bremner et al. [2] showed that there exists a 2-local IQP circuit with $O(\text{poly}(n))$ output qubits such that it is not weakly simulatable (under a plausible assumption), where an IQP circuit is a commuting quantum circuit such that each commuting gate is diagonal in the X-basis $\{(|0\rangle \pm |1\rangle)/\sqrt{2}\}$. This means that, when the number of output qubits is large, even a simple commuting quantum circuit is powerful. On the other hand, Ni et al. [7] showed that any 2-local commuting quantum circuit with $O(\log n)$ output qubits is strongly simulatable, whereas there exists a 3-local commuting quantum circuit with only one output qubit such that it is not strongly simulatable (under a plausible assumption). Thus, when the number of output qubits is $O(\log n)$, the classical simulatability of commuting quantum circuits depends on the number of qubits affected by each gate. A natural question is whether there exists a commuting quantum circuit with $O(\log n)$ output qubits such that it is not weakly simulatable.

We provide the first formal evidence for answering the question affirmatively:

Theorem 1. *There exists a 5-local commuting quantum circuit with $O(\log n)$ output qubits such that it is not weakly simulatable unless the polynomial hierarchy* PH *collapses to the third level, i.e., unless* $\mathsf{PH} = \Delta_3^p$.

It is believed that PH does not collapse to any level. Thus, the circuit in Theorem 1 is a desired evidence. To prove Theorem 1, we first show the existence of a depth-3 quantum circuit A_n with $O(\text{poly}(n))$ output qubits such that it is not weakly simulatable unless $\mathsf{PH} = \Delta_3^p$. Our idea for constructing the circuit in Theorem 1 is to decrease the number of the output qubits by combining A_n with the OR reduction quantum circuit [5], which reduces the computation of the OR function on k bits to that on $O(\log k)$ bits. The resulting circuit has $O(\log n)$ output qubits and is not weakly simulatable unless $\mathsf{PH} = \Delta_3^p$, but it

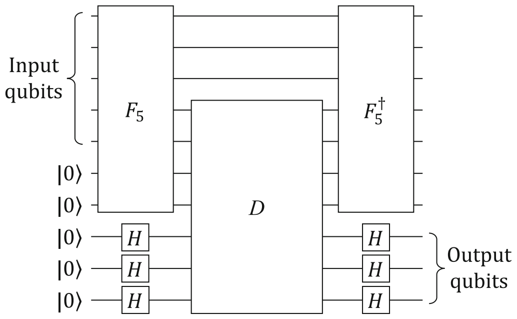


Fig. 1. Circuit $(F_n^\dagger \otimes H^{\otimes l})D(F_n \otimes H^{\otimes l})$, where $n = 5$, $s = 2$, $t = 4$, and $l = 3$

is not a commuting quantum circuit. An important observation is that the OR reduction circuit can be transformed into a 2-local commuting quantum circuit. We regard a quantum circuit consisting of A_n, g, and $A_n^\dagger$ as a single gate $A_n^\dagger g A_n$ for any gate g that is either a ΛX gate or a commuting gate in the commuting OR reduction circuit, where $A_n^\dagger$ is the circuit obtained from A_n by reversing the order of the gates and replacing each gate with its inverse. A rigorous analysis of a quantum circuit consisting of the gates $A_n^\dagger g A_n$ implies Theorem 1.

Then, we study the weak simulatability of a generalized version of the circuit in Theorem 1. We assume that we are given two quantum circuits F_n and D: F_n has n input qubits, $s = O(\text{poly}(n))$ ancillary qubits, and t output qubits, and D is a commuting quantum circuit on $t + l$ qubits such that each commuting gate is diagonal in the Z-basis $\{|0\rangle, |1\rangle\}$, where $l = O(\log n)$. We consider the circuit of the form depicted in Fig. 1, which we denote as $(F_n^\dagger \otimes H^{\otimes l})D(F_n \otimes H^{\otimes l})$, although its precise definition is provided in Section 3.2. The input qubits and output qubits of the circuit are the input qubits of F_n and the l qubits on which H gates are applied, respectively. In particular, when F_n is A_n and D consists only of controlled phase-shift gates, a commuting version of the whole circuit is the circuit in Theorem 1. We show the following relationship:

Theorem 2. *If F_n is weakly simulatable, then $(F_n^\dagger \otimes H^{\otimes l})D(F_n \otimes H^{\otimes l})$ with $l = O(\log n)$ output qubits is also weakly simulatable.*

This is a generalization of the previous result that any IQP circuit with $O(\log n)$ output qubits is weakly simulatable [2], which corresponds to the case when F_n is a layer of H gates. We show Theorem 2 by generalizing the proof of the previous result. Theorem 2 implies a suggestion on how to improve Theorem 1 in terms of locality as follows. Choosing a depth-3 quantum circuit as F_n yields the 5-local commuting quantum circuit in Theorem 1 and a possible way to construct a 3- or 4-local one that is not weakly simulatable would be to somehow choose a depth-2 quantum circuit as F_n. By Theorem 2, such a construction is impossible. This is because, since any depth-2 quantum circuit is weakly simulatable [10], choosing a depth-2 quantum circuit as F_n yields only a weakly simulatable quantum circuit.

We consider two applications of our results. The first one is to examine the ability of IQP circuits to implement a fundamental operation, the OR reduction operation defined as follows: a quantum operation on $n+m$ qubits is called an OR reduction operation if it maps $|x\rangle|0^m\rangle$ to $|x\rangle|\eta\rangle$ for any $x \in \{0,1\}^n$, where $m = O(\log n)$, $|\eta\rangle = |0^m\rangle$ if $x = 0^n$, and $\langle 0^m|\eta\rangle = 0$ otherwise [5]. This operation is shown to be quite useful for significantly reducing the depth of quantum circuits with a certain gate set [5,8]. A 2-local commuting quantum circuit can implement an OR reduction operation as will be shown in the proof of Theorem 1, but we provide an evidence that IQP circuits cannot:

Theorem 3. *For any OR reduction operation, there does not exist an IQP circuit for implementing it unless* $\mathsf{PH} = \Delta_3^p$.

Theorem 3 provides the reason for the difference, which is shown by Theorem 1, between IQP circuits and 5-local commuting quantum circuits in terms of the weak simulatability. It also shows the limitation of the computational power of IQP circuits. The proof of Theorem 3 is a direct application of Theorems 1 and 2.

The second application is to examine the weak simulatability of slightly extended Clifford circuits. For comparison, let us consider Clifford circuits with n input qubits, $O(\mathrm{poly}(n))$ ancillary qubits in a product state, and $O(\log n)$ output qubits. A simple extension of the proof in Refs. [3,6] implies that any Clifford circuit in this setting is strongly simulatable. We show the following theorem:

Theorem 4. *There exists a Clifford circuit augmented by a depth-1 non-Clifford layer with* $O(\mathrm{poly}(n))$ *ancillary qubits in a particular product state and with* $O(\log n)$ *output qubits such that it is not weakly simulatable unless* $\mathsf{PH} = \Delta_3^p$.

Just like Theorems 1 and 2, Theorem 4 contributes to understanding a subtle difference between quantum and classical computation. The proof of Theorem 4 is a simple modification of the proof of Theorem 1.

2 Preliminaries

2.1 Quantum Circuits

The elementary gates in this paper are a Hadamard gate H, a phase-shift gate $R(\theta)$ with angle $\theta = \pm 2\pi/2^k$ for any $k \in \mathbb{N}$, and a controlled-Z gate ΛZ, where

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},\ R(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix},\ \Lambda Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

We denote $R(\pi)$, $R(\pi/2)$, and $HR(\pi)H$ as Z, P, and X, respectively, where Z and X (with $Y = iXZ$ and identity I) are called Pauli gates. We also denote $H\Lambda Z H$ as ΛX, which is a CNOT gate, where H acts on the target qubit. A quantum circuit consists of the elementary gates. A Clifford circuit is a quantum circuit consisting only of H, P, and ΛZ. A commuting quantum circuit is a quantum circuit consisting of pairwise commuting gates, where we do not require that each commuting gate be one of the elementary gates. In other words, when we think of a quantum circuit as a commuting quantum circuit, we are allowed to regard a group of elementary gates in the circuit as a single gate and we require that such gates be pairwise commuting. An IQP circuit is a commuting quantum circuit such that each commuting gate is diagonal in the X-basis $\{(|0\rangle \pm |1\rangle)/\sqrt{2}\}$ [2]. Any IQP circuit on s qubits can be represented as follows: the first part consists of H gates on s qubits, the middle part a commuting quantum circuit D, and the last part H gates on s qubits, where each commuting gate in D is diagonal in the Z-basis $\{|0\rangle, |1\rangle\}$.

The complexity measures of a quantum circuit are its size and depth. The size of a quantum circuit is the number of elementary gates in the circuit. To define the depth, we consider the circuit as a set of layers $1, \ldots, d$ consisting of one-qubit or two-qubit gates, where gates in the same layer act on pairwise disjoint sets of qubits and any gate in layer j is applied before any gate in layer $j+1$. The depth of the circuit is the smallest possible value of d [4]. It might be natural to require that each gate in a layer be one of the elementary gates, but for simplicity, when we count the depth, we allow any one-qubit or two-qubit gates that can be obtained as a sequence of elementary gates in the circuit. This does not essentially affect our results, since, regardless of whether we adopt the requirement or not, the depth of the circuit we are interested in is a constant. A quantum circuit can use ancillary qubits initialized to $|0\rangle$.

We deal with a uniform family of polynomial-size quantum circuits $\{C_n\}_{n\geq 1}$, where each C_n has n input qubits and $O(\mathrm{poly}(n))$ ancillary qubits, and angles θ of phase-shift gates in C_n are restricted to $\pm 2\pi/2^k$ with $k = O(\mathrm{poly}(n))$. Some of the input and ancillary qubits are called output qubits. At the end of the computation, Z-measurements, i.e., measurements in the Z-basis, are performed on the output qubits. The uniformity means that there exists a polynomial-time deterministic classical algorithm for computing the function $1^n \mapsto \overline{C_n}$, where $\overline{C_n}$ is the classical description of C_n. A symbol denoting a quantum circuit, such as C_n, also denotes its matrix representation in some fixed basis. Any quantum circuit in this paper is understood to be an element of a uniform family of quantum circuits and thus, for simplicity, we deal with a quantum circuit C_n in place of a family $\{C_n\}_{n\geq 1}$. We require that each commuting gate in a commuting quantum circuit act on a constant number of qubits. When every commuting gate acts on at most c qubits, the circuit is said to be c-local [7].

2.2 Classical Simulatability and Complexity Classes

We deal with polynomial-size classical circuits and randomized classical circuits [2] to model polynomial-time deterministic classical algorithms and their probabilistic versions, respectively. Let C_n be a quantum circuit with n input qubits, $O(\mathrm{poly}(n))$ ancillary qubits, and m output qubits. For any $x \in \{0,1\}^n$, there exists an output probability distribution $\{(y, \Pr[C_n(x) = y])\}_{y\in\{0,1\}^m}$, where $\Pr[C_n(x) = y]$ is the probability of obtaining $y \in \{0,1\}^m$ by

Z-measurements on the output qubits of C_n with the input state $|x\rangle$. The classical simulatability is defined as follows [2,6,7,9,10]:

Definition 1. – *C_n is strongly simulatable if $\Pr[C_n(x) = y]$ and its marginal output probability can be computed up to an exponentially small additive error in classical $O(\text{poly}(n))$ time. More precisely, for any $0 < m' \leq m$, m' output qubits chosen from the m output qubits of C_n, and polynomial p, there exists a polynomial-size classical circuit D_n such that, for any $x \in \{0,1\}^n$ and $y' \in \{0,1\}^{m'}$, $|D_n(x, y') - \Pr[C_n(x) = y']| \leq 1/2^{p(n)}$.*

– *C_n is weakly simulatable if $\{(y, \Pr[C_n(x) = y])\}_{y \in \{0,1\}^m}$ can be sampled up to an exponentially small additive error in classical $O(\text{poly}(n))$ time. More precisely, for any polynomial p, there exists a polynomial-size randomized classical circuit R_n such that, for any $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$, $|\Pr[R_n(x) = y] - \Pr[C_n(x) = y]| \leq 1/2^{p(n)}$.*

Any strongly simulatable quantum circuit is weakly simulatable [2,10].

The following two complexity classes are important for our discussion [2]:

Definition 2. *Let L be a language, i.e., $L \subseteq \{0,1\}^*$.*

– *$L \in \mathsf{PostBQP}$ if there exists a polynomial-size quantum circuit C_n with n input qubits, $O(\text{poly}(n))$ ancillary qubits, one output qubit, and one particular qubit (other than the output qubit) called the postselection qubit such that, for any $x \in \{0,1\}^n$, $\Pr[\text{post}_n(x) = 0] > 0$, $\Pr[C_n(x) = 1|\text{post}_n(x) = 0] \geq 2/3$ if $x \in L$, and $\Pr[C_n(x) = 1|\text{post}_n(x) = 0] \leq 1/3$ if $x \notin L$, where the event "$\text{post}_n(x) = 0$" means that the classical outcome of the Z-measurement on the postselection qubit is 0.*

– *$L \in \mathsf{PostBPP}$ if there exists a polynomial-size randomized classical circuit R_n with n input bits that, for any $x \in \{0,1\}^n$, outputs $R_n(x), \text{post}_n(x) \in \{0,1\}$ such that $\Pr[\text{post}_n(x) = 0] > 0$, $\Pr[R_n(x) = 1|\text{post}_n(x) = 0] \geq 2/3$ if $x \in L$, and $\Pr[R_n(x) = 1|\text{post}_n(x) = 0] \leq 1/3$ if $x \notin L$.*

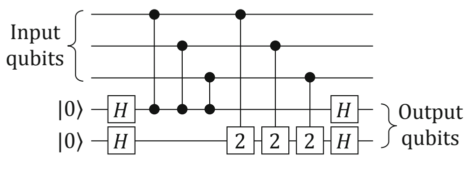
We use the notation $\text{post}_n(x) = 0$ both in the quantum and classical settings, but the meaning will be clear from the context.

Another important class is the polynomial hierarchy $\mathsf{PH} = \bigcup_{j \geq 1} \Delta_j^p$. Here, $\Delta_1^p = \mathsf{P}$ and $\Delta_{j+1}^p = \mathsf{P}^{\mathsf{N}\Delta_j^p}$ for any $j \geq 1$, where P is the class of languages decided by polynomial-size classical circuits and $\mathsf{N}\Delta_j^p$ is the non-deterministic class associated to Δ_j^p [2]. It is believed that $\mathsf{PH} \neq \Delta_j^p$ for any $j \geq 1$. As shown in Ref. [2], if $\mathsf{PostBQP} \subseteq \mathsf{PostBPP}$, then $\mathsf{PH} = \Delta_3^p$. It can be shown that, in our setting of elementary gates and quantum circuits, this relationship also holds when the condition $\Pr[\text{post}_n(x) = 0] > 0$ in the definition of $\mathsf{PostBQP}$ is replaced with the condition that, for some polynomial q (depending only on C_n), $\Pr[\text{post}_n(x) = 0] \geq 1/2^{q(n)}$. In the following, we adopt the latter condition.

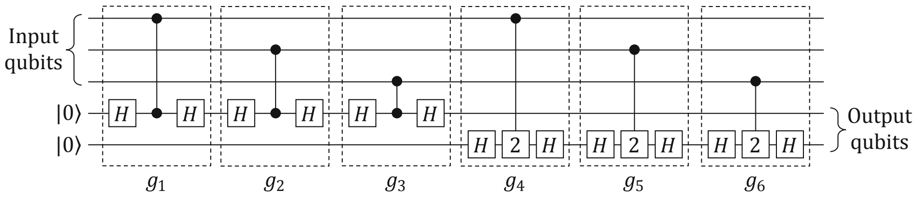
3 Commuting Quantum Circuits

3.1 Hardness of the Weak Simulation

A key component of the circuit in Theorem 1 is a depth-3 quantum circuit with $O(\text{poly}(n))$ output qubits such that it is not weakly simulatable unless



(a)



(b)

Fig. 2. (a): The non-commuting OR reduction circuit, where the gate represented by two black circles connected by a vertical line is a ΛZ gate, i.e., a controlled-$R(2\pi/2^1)$ gate, and the gate "2" is an $R(2\pi/2^2)$ gate. (b): The commuting OR reduction circuit.

$\mathsf{PH} = \Delta_3^p$. Such a circuit exists when weak simulatability is defined with respect to a multiplicative error [2]. We analyze its weak simulatability with an exponentially small additive error (Definition 1) and show the following lemma:

Lemma 1. *There exists a depth-3 polynomial-size quantum circuit with* $O(\text{poly}(n))$ *output qubits such that it is not weakly simulatable unless* $\mathsf{PH} = \Delta_3^p$.

The proof is omitted in this paper. We decrease the number of the output qubits of the circuit in Lemma 1 using the OR reduction circuit [5]. The OR reduction circuit with b input qubits has $m = \lceil \log(b+1) \rceil$ ancillary qubits, which are also output qubits. For any input state $|x\rangle|0^m\rangle$ with $x \in \{0,1\}^b$, the circuit outputs $|x\rangle|\eta\rangle$, where $|\eta\rangle = |0^m\rangle$ if $x = 0^b$ and $\langle 0^m|\eta\rangle = 0$ otherwise. The first part consists of H gates on the ancillary qubits. The middle part consists of b controlled-$R(2\pi/2^k)$ gates over all $1 \leq k \leq m$, where each gate uses an input qubit as the control qubit and an ancillary qubit as the target qubit. The last part is the same as the first one. We call the circuit the *non-commuting* OR reduction circuit. It is depicted in Fig. 2(a), where $b = 3$.

An important observation is that the non-commuting OR reduction circuit can be transformed into a 2-local commuting quantum circuit. This is shown by considering a quantum circuit consisting of gates g_j on two qubits, where g_j is a controlled-$R(2\pi/2^k)$ gate sandwiched between H gates on the target qubit. Since $H^2 = I$ and controlled-$R(2\pi/2^k)$ gates are pairwise commuting gates on two qubits, the operation implemented by the circuit is the same as that implemented by the non-commuting OR reduction circuit and the gates g_j are pairwise commuting gates on two qubits. We call the circuit the *commuting* OR reduction circuit. It is depicted in Fig. 2(b), where $b = 3$. Combining this circuit with the circuit in Lemma 1 implies the following lemma:

Lemma 2. *There exists a commuting quantum circuit with $O(\log n)$ output qubits such that it is not weakly simulatable unless* $\mathsf{PH} = \Delta_3^p$.

Proof. We assume that $\mathsf{PH} \neq \Delta_3^p$. Lemma 1 and its proof imply that there exists a depth-3 polynomial-size quantum circuit A_n with n input qubits, $a+b$ ancillary qubits, and $b+2$ output qubits such that it is not weakly simulatable, where $a = O(\mathrm{poly}(n))$ and $b = O(\mathrm{poly}(n))$. Moreover, the proof that A_n is not weakly simulatable depends only on $\Pr[A_n(x) = 0^{b+1}1]$ and $\Pr[A_n(x) = 0^{b+1}0]$ for any $x \in \{0,1\}^n$. We decrease the number of the first $b+1$ output qubits, which are the postselection qubits, using the commuting OR reduction circuit. To do so, we construct a quantum circuit E_n with n input qubits, $a+b+m+1$ ancillary qubits, and $m+1$ output qubits as follows, where $m = \lceil \log(b+2) \rceil$. As an example, E_n is depicted in Fig. 3(a), where $n=5$, $a=0$, and $b=2$.

1. Apply A_n on n input qubits and $a+b$ ancillary qubits, where the input qubits of E_n are those of A_n.
2. Apply a ΛX gate on the last output qubit of A_n and on an ancillary qubit (other than the ancillary qubits in Step 1), where the output qubit is the control qubit.
3. Apply a commuting OR reduction circuit on the $b+1$ postselection qubits and m ancillary qubits (other than the ancillary qubits in Steps 1 and 2), where the $b+1$ qubits are the input qubits of the OR reduction circuit.
4. Apply $A_n^\dagger$ as in Step 1, where $A_n^\dagger$ is the circuit obtained from A_n by reversing the order of the gates and replacing each gate with its inverse.

The output qubits of E_n are the $m+1$ ancillary qubits used in Steps 2 and 3. Step 4 does not affect the output probability distribution of E_n, but it allows us to construct the commuting quantum circuit described below. By the construction of E_n, for any $x \in \{0,1\}^n$, $\Pr[A_n(x) = 0^{b+1}1] = \Pr[E_n(x) = 0^m 1]$ and $\Pr[A_n(x) = 0^{b+1}0] = \Pr[E_n(x) = 0^m 0]$. This implies that E_n is not weakly simulatable. The proof is the same as that of Lemma 1 except that the number of output qubits we need to consider is only $m+1 = O(\log n)$.

We regard a quantum circuit consisting of A_n, g, and $A_n^\dagger$ (in this order) as a single gate $A_n^\dagger g A_n$ for any gate g that is either a ΛX gate in Step 2 of E_n or g_j in the commuting OR reduction circuit. We consider a quantum circuit consisting of the gates $A_n^\dagger g A_n$. The input qubits and output qubits of E_n are naturally considered as the input qubits and output qubits of the new circuit, respectively. The new circuit based on E_n in Fig. 3(a) is depicted in Fig. 3(b). Since these gates g in E_n are pairwise commuting, so are the gates $A_n^\dagger g A_n$. Moreover, since the depth of A_n is three and g acts on two qubits, $A_n^\dagger g A_n$ acts on a constant number of qubits. By the construction of the new circuit, its output probability distribution is the same as that of E_n. Thus, the new circuit is not weakly simulatable. □

We analyze $A_n^\dagger g A_n$ in the above proof, which implies the following lemma:

Lemma 3. *For any gate $A_n^\dagger g A_n$ in the proof of Lemma 2, there exists a quantum circuit on at most five qubits that implements the gate.*

The proof is omitted in this paper. The above lemmas imply Theorem 1.

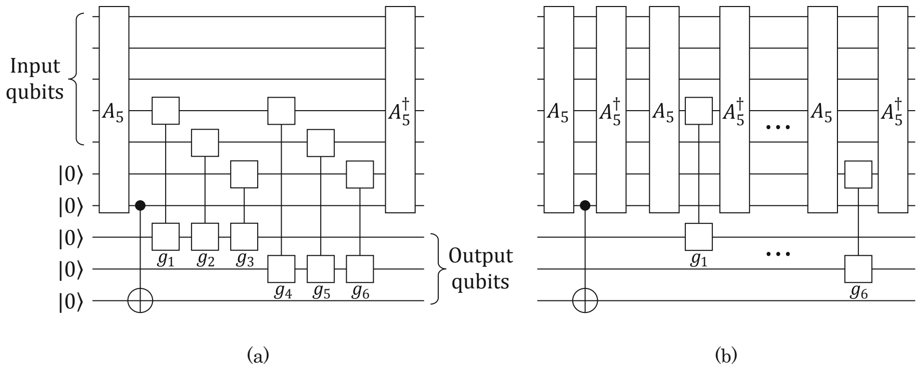


Fig. 3. (a): Circuit E_n, where $n = 5$, $a = 0$, and $b = 2$. The gate represented by a black circle and $\oplus$ connected by a vertical line is a ΛX gate. The gates g_j are the ones in Fig. 2(b). (b): The commuting quantum circuit based on E_n in (a).

3.2 Weak Simulatability of a Generalized Version

As in Fig. 2(a), a non-commuting OR reduction circuit with $b+1$ input qubits can be represented as three parts: the first part consists of H gates on m qubits, the middle part a quantum circuit D', and the last part H gates on m qubits, where $m = \lceil \log(b+2) \rceil$ and D' consists only of controlled-$R(2\pi/2^k)$ gates. Since ΛX (in Step 2 of E_n in the proof of Lemma 2) is $H\Lambda Z H$, the circuit in Theorem 1 can be represented similarly: the first part consists of A_n and H gates on $m+1$ qubits, the middle part D'', and the last part $A_n^\dagger$ and H gates on $m+1$ qubits, where D'' consists of D' and ΛZ, and A_n is the depth-3 quantum circuit (with $a+b$ ancillary qubits and $b+2$ output qubits) described above. The output qubits of the whole circuit are the $m+1$ qubits on which H gates are applied.

We consider a generalized version of the circuit in Theorem 1. We assume that we are given two quantum circuits F_n and D: F_n has n input qubits, $s = O(\text{poly}(n))$ ancillary qubits, and t output qubits, and D is a commuting quantum circuit on $t+l$ qubits such that each commuting gate is diagonal in the Z-basis, where $l = O(\log n)$. We construct a quantum circuit with n input qubits, $s+l$ ancillary qubits, and l output qubits as follows, where we denote this circuit as $(F_n^\dagger \otimes H^{\otimes l})D(F_n \otimes H^{\otimes l})$. As an example, the circuit is depicted in Fig. 1, where $n = 5$, $s = 2$, $t = 4$, and $l = 3$.

1. Apply F_n on n input qubits and s ancillary qubits, where the input qubits of the whole circuit are those of F_n.
2. Apply H gates on l ancillary qubits (other than the ancillary qubits in Step 1).
3. Apply D on $t+l$ qubits, which are the output qubits of F_n and the ancillary qubits in Step 2.
4. Apply H gates as in Step 2 and then $F_n^\dagger$ as in Step 1.

The output qubits of the whole circuit are the l qubits on which H gates are applied. The circuit in Theorem 1 corresponds to the case when $F_n = A_n$,

$D = D''$, $s = a + b$, $t = b + 2$, and $l = m + 1$. When F_n is a layer of H gates with arbitrary s and t, the circuit $(F_n^\dagger \otimes H^{\otimes l})D(F_n \otimes H^{\otimes l})$, which becomes an IQP circuit, is weakly simulatable [2]. A simple generalization of the proof of the previous result implies Theorem 2. The detailed proof is omitted in this paper.

4 Applications

4.1 IQP Circuits and 2-local Commuting Quantum Circuits

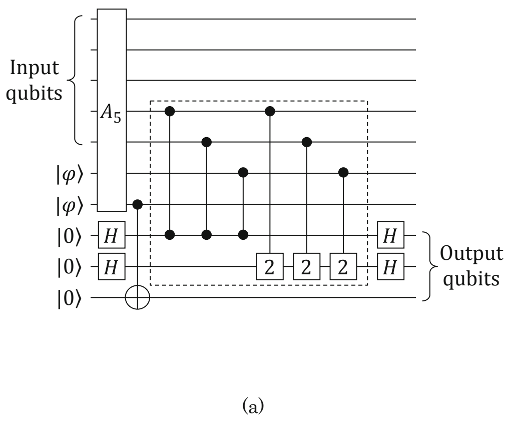
To exhibit the difference between IQP circuits and 2-local commuting quantum circuits, we consider OR reduction operations (defined in Section 1). For example, the operation implemented by the commuting OR reduction circuit in Section 3 is an OR reduction operation. This immediately shows that there exists a 2-local commuting quantum circuit for implementing an OR reduction operation. In contrast, Theorem 3 shows that IQP circuits cannot implement any OR reduction operation unless $\mathsf{PH} = \Delta_3^p$. We now prove the theorem:

Proof. We assume that $\mathsf{PH} \neq \Delta_3^p$ and there exists an IQP circuit (on s qubits) for implementing an OR reduction operation. We use this IQP circuit in place of the commuting OR reduction circuit in the proof of Lemma 2. The resulting circuit B_n is the same as the one depicted in Fig. 3(a) except that the middle part (excluding a ΛX gate) becomes the IQP circuit. The proof of the lemma implies that B_n is not weakly simulatable.

We simplify B_n. Recall that, by the definition, the IQP circuit on s qubits can be represented as follows: the first part consists of H gates on the s qubits, the middle part a commuting quantum circuit D, and the last part H gates on the s qubits, where each commuting gate in D is diagonal in the Z-basis. The third layer of A_n included in B_n can be decomposed into two sublayers: the first sublayer consists of ΛZ gates and the second sublayer H gates. All the H gates are cancelled out by the corresponding H gates in the first part of the IQP circuit. Similarly, all the H gates in the first layer of $A_n^\dagger$ are cancelled out by the corresponding H gates in the last part of the IQP circuit. When an H gate in the IQP circuit is applied on a qubit on which no gate is applied in the third layer of A_n, it can be regarded as a gate in the second layer of A_n or $A_n^\dagger$. Thus, by adding the ΛZ gates in the third layer of A_n and in the first layer of $A_n^\dagger$ to D, we can regard A_n as a depth-2 quantum circuit and thus B_n is a quantum circuit of the form in Theorem 2 with $F_n = A_n$ of depth 2. Since any depth-2 quantum circuit is weakly simulatable [10], B_n is weakly simulatable. This contradicts the above observation. Thus, if $\mathsf{PH} \neq \Delta_3^p$, there does not exist an IQP circuit for implementing an OR reduction operation. □

4.2 Clifford Circuits

We consider Clifford circuits with n input qubits, $O(\text{poly}(n))$ ancillary qubits, and $O(\log n)$ output qubits, where the ancillary qubits are allowed to be in a general product state. Such a Clifford circuit with only one output qubit is strongly simulatable [3,6]. We can simply extend this property as follows:



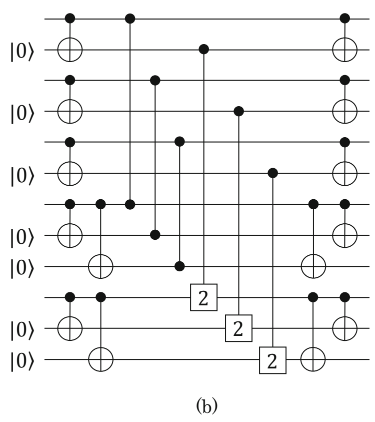


Fig. 4. (a): Circuit E'_n, where $n = 5$, $a = 0$, and $b = 2$. The dashed box represents the middle part of the non-commuting OR reduction circuit. (b): The circuit obtained from the middle part in (a). The qubits in state $|0\rangle$ are new ancillary qubits.

Lemma 4. *Any Clifford circuit with $O(\text{poly}(n))$ ancillary qubits in a general product state and with $O(\log n)$ output qubits is strongly simulatable.*

The proof is omitted in this paper.

Just like Lemma 1, we can show the following lemma:

Lemma 5. *There exists a Clifford circuit with $O(\text{poly}(n))$ ancillary qubits in a particular product state and with $O(\text{poly}(n))$ output qubits such that it is not weakly simulatable unless $\mathsf{PH} = \Delta_3^p$.*

The proof is also omitted. The proof of Lemma 2 leads to the following lemma:

Lemma 6. *There exists a Clifford circuit combined with an OR reduction circuit with $O(\text{poly}(n))$ ancillary qubits in a particular product state and with $O(\log n)$ output qubits such that it is not weakly simulatable unless $\mathsf{PH} = \Delta_3^p$.*

Proof. We assume that $\mathsf{PH} \neq \Delta_3^p$. Lemma 5 and its proof imply that there exists a Clifford circuit A_n with n input qubits, $a = O(\text{poly}(n))$ ancillary qubits initialized to $|0\rangle$, $b = O(\text{poly}(n))$ ancillary qubits initialized to $|\varphi\rangle = (|0\rangle + e^{i\pi/4}|1\rangle)/\sqrt{2}$, and $b+2$ output qubits such that it is not weakly simulatable. Moreover, the proof that A_n is not weakly simulatable depends only on $\Pr[A_n(x) = 0^{b+1}1]$ and $\Pr[A_n(x) = 0^{b+1}0]$ for any $x \in \{0,1\}^n$. Just like E_n in the proof of Lemma 2, we construct a quantum circuit E'_n with n input qubits and $a + b + m + 1$ ancillary qubits as follows, where $m = \lceil \log(b+2) \rceil$. As an example, E'_n is depicted in Fig. 4(a), where $n = 5$, $a = 0$, and $b = 2$.

1. Apply A_n on n input qubits, a ancillary qubits initialized to $|0\rangle$, and b ancillary qubits initialized to $|\varphi\rangle$, where the input qubits of E'_n are those of A_n.
2. Apply a ΛX gate on the last output qubit of A_n and on an ancillary qubit.
3. Apply a *non-commuting* OR reduction circuit on the $b+1$ postselection qubits and m ancillary qubits.

The output qubits of E'_n are the $m+1$ ancillary qubits used in Steps 2 and 3. We can show that E'_n is not weakly simulatable as in the proof of Lemma 2. □

We replace the non-commuting OR reduction circuit with a constant-depth OR reduction circuit with unbounded fan-out gates [5], where an unbounded fan-out gate can be considered as a sequence of ΛX gates with the same control qubit. Decomposing the unbounded fan-out gates into ΛX gates in the constant-depth OR reduction circuit yields a Clifford circuit augmented by a depth-1 non-Clifford layer, which consists only of controlled phase-shift gates. This procedure transforms the middle part of the non-commuting OR reduction circuit in Step 3, which is the only part in E'_n that includes non-Clifford gates, into a quantum circuit that has ΛX gates and a depth-1 layer consisting of all the gates in the middle part. The circuit obtained in this way from the middle part in Fig. 4(a) is depicted in Fig. 4(b). This transformation with Lemma 6 implies Theorem 4.

Acknowledgments. We thank the anonymous referees for valuable comments. S.T. is deeply grateful to the ELC project (Grant-in-Aid for Scientific Research on Innovative Areas MEXT Japan) for encouraging the research presented in this paper.

References

1. Aaronson, S., Arkhipov, A.: The computational complexity of linear optics. In: Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC), pp. 333–342 (2011)
2. Bremner, M.J., Jozsa, R., Shepherd, D.J.: Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. Proceedings of the Royal Society A **467**, 459–472 (2011)
3. Clark, S., Jozsa, R., Linden, N.: Generalized Clifford groups and simulation of associated quantum circuits. Quantum Information and Computation **8**(1&2), 106–126 (2008)
4. Fenner, S.A., Green, F., Homer, S., Zhang, Y.: Bounds on the power of constant-depth quantum circuits. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 44–55. Springer, Heidelberg (2005)
5. Høyer, P., Špalek, R.: Quantum fan-out is powerful. Theory of Computing **1**(5), 81–103 (2005)
6. Jozsa, R., van den Nest, M.: Classical simulation complexity of extended Clifford circuits. Quantum Information and Computation **14**(7&8), 633–648 (2014)
7. Ni, X., van den Nest, M.: Commuting quantum circuits: efficient classical simulations versus hardness results. Quantum Information and Computation **13**(1&2), 54–72 (2013)
8. Takahashi, Y., Tani, S.: Collapse of the hierarchy of constant-depth exact quantum circuits. In: Proceedings of the 28th IEEE Conference on Computational Complexity (CCC), pp. 168–178 (2013)
9. Takahashi, Y., Yamazaki, T., Tanaka, K.: Hardness of classically simulating quantum circuits with unbounded Toffoli and fan-out gates. Quantum Information and Computation **14**(13&14), 1149–1164 (2014)
10. Terhal, B.M., DiVincenzo, D.P.: Adaptive quantum computation, constant-depth quantum circuits and Arthur-Merlin games. Quantum Information and Computation **4**(2), 134–145 (2004)

Evaluating Matrix Circuits

Daniel König and Markus Lohrey(✉)

Universität Siegen, Siegen, Germany
lohrey@eti.uni-siegen.de

Abstract. The circuit evaluation problem (also known as the compressed word problem) for finitely generated linear groups is studied. The best upper bound for this problem is coRP, which is shown by a reduction to polynomial identity testing (PIT). Conversely, the compressed word problem for the linear group $\mathsf{SL}_3(\mathbb{Z})$ is equivalent to PIT. In the paper, it is shown that the compressed word problem for every finitely generated nilpotent group is in $\mathsf{DET} \subseteq \mathsf{NC}^2$. Within the larger class of polycyclic groups we find examples where the compressed word problem is at least as hard as PIT for skew arithmetical circuits.

1 Introduction

The study of circuit evaluation problems has a long tradition in theoretical computer science and is tightly connected to many aspects in computational complexity theory. One of the most important circuit evaluation problems is *polynomial identity testing (PIT)*: The input is an arithmetic circuit, whose internal gates are labelled with either addition or multiplication and its input gates are labelled with variables $(x_1, x_2, \ldots)$ or constants $(-1, 0, 1)$, and it is asked whether the output gate evaluates to the zero polynomial (in this paper, we always work in the polynomial ring over the coefficient ring $\mathbb{Z}$ or $\mathbb{Z}_p$ for a prime p). Based on the Schwartz-Zippel-DeMillo-Lipton Lemma, Ibarra and Moran [10] proved that PIT over $\mathbb{Z}$ or $\mathbb{Z}_p$ belongs to the class coRP (co-randomized polynomial time). Whether PIT $\in \mathsf{P}$ is an important problem. In [11] it was shown that if there is a language in $\mathsf{DTIME}(2^{\mathcal{O}(n)})$ with circuit complexity $2^{\Omega(n)}$, then $\mathsf{P} = \mathsf{BPP}$ (and hence $\mathsf{P} = \mathsf{RP} = \mathsf{coRP}$). On the other hand, Kabanets and Impagliazzo [12] proved that if PIT belongs to P, then (i) there is a language in $\mathsf{NEXPTIME}$ that does not have polynomial size circuits, or (ii) the permanent is not computable by polynomial size arithmetic circuits. Both conclusions are major open problem in complexity theory. Hence, although it is quite plausible that PIT $\in \mathsf{P}$, it is difficult to prove.

Circuit evaluation problems can be also studied for other structures than polynomial rings, in particular non-commutative structures. For finite monoids, the circuit evaluation was studied in [6], where it was shown that for every non-solvable finite monoid the circuit evaluation problem is P-complete, whereas for every solvable finite monoid, the circuit evaluation problem belongs to the parallel complexity class $\mathsf{DET} \subseteq \mathsf{NC}^2$. Starting with [17] the circuit evaluation problem has been also studied for infinite finitely generated (f.g) monoids, in

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 235–248, 2015.
DOI: 10.1007/978-3-319-21398-9_19

particular infinite f.g. groups. In this context, the input gates of the circuit are labelled with generators of the monoid, the internal gates compute the product of the two input gates, and it is asked whether the circuit evaluates to the identity element. In [17] and subsequent work, the circuit evaluation problem is also called the *compressed word problem (CWP)*. This is due to the fact that if one forgets the underlying monoid structure of a multiplicative circuit, the circuit simply evaluates to a word over the monoid generators labelling the input gates. This word can be of length exponential in the number of circuit gates, and the circuit can be seen as a compressed representation of this word. In this context, circuits are also known as *straight-line programs* and are intensively studied in the area of string compression [18].

Concerning the CWP, polynomial time algorithms have been developed for many important classes of groups, e.g., finite groups, f.g. nilpotent groups, f.g. free groups, graph groups (also known as right-angled Artin groups), and virtually special groups. The latter contain all Coxeter groups, one-relator groups with torsion, fully residually free groups, and fundamental groups of hyperbolic 3-manifolds; see [19]. For the important class of f.g. linear groups, i.e., f.g. groups of matrices over a field, the CWP reduces to PIT (over $\mathbb{Z}$ or $\mathbb{Z}_p$, depending on the characteristic of the field) and hence belongs to coRP [19]. Vice versa, in [19] it was shown that PIT over $\mathbb{Z}$ reduces to the CWP for the linear group $\mathsf{SL}_3(\mathbb{Z})$. This result indicates that derandomizing the CWP for a f.g. linear group will be in general very difficult.

In this paper, we further investigate the tight correspondence between commutative circuits over rings and non-commutative circuits over linear groups. In Sec. 6 we study the complexity of the CWP for f.g. nilpotent groups. It is known to be in P [19]. Here, we show that for every f.g. nilpotent group the CWP belongs to the parallel complexity class $\mathsf{DET} \subseteq \mathsf{NC}^2$, which is the class of all problems that are NC^1-reducible to the computation of the determinant of an integer matrix, see [8]. To the knowledge of the authors, f.g. nilpotent groups are the only examples of infinite groups for which the CWP belongs to NC. Even for free groups, the CWP is P-complete [17]. The main step of our proof for f.g. nilpotent groups is to show that for a torsion-free f.g. nilpotent group G the CWP belongs to the logspace counting class $\mathsf{C}_=\mathsf{L}$ (and is in fact $\mathsf{C}_=\mathsf{L}$-complete if $G \neq 1$). To show this, we use the fact that a f.g. torsion-free nilpotent group embeds into the group $\mathsf{UT}_d(\mathbb{Z})$ of d-dimensional unitriangular matrices over $\mathbb{Z}$ for some d. Then, we reduce the CWP for $\mathsf{UT}_d(\mathbb{Z})$ to the $\mathsf{C}_=\mathsf{L}$-complete problem, whether two additive circuits over the naturals evaluate to the same number. Let us mention that there are several $\mathsf{C}_=\mathsf{L}$-complete problems related to linear algebra [1].

We also study the CWP for the matrix group $\mathsf{UT}_d(\mathbb{Z})$ for the case that the dimension d is not fixed, i.e., part of the input (Sec. 7). In this case, the CWP turns out to be complete for the counting class $\mathsf{C}_=\mathsf{LogCFL}$, which is the LogCFL-analogue of $\mathsf{C}_=\mathsf{L}$.

Finally, in Sec. 8 we move from nilpotent groups to polycyclic groups. These are solvable groups, where every subgroup is finitely generated. By results of

Maltsev, Auslander, and Swan these are exactly the solvable subgroups of $\mathsf{GL}_d(\mathbb{Z})$ for some d. We prove that polynomial identity testing for skew arithmetical circuits reduces to the CWP for a specific 2-generator polycyclic group of Hirsch length 3. A skew arithmetical circuit is an arithmetic circuit (as defined in the first paragraph of the introduction) such that for every multiplication gate, one of its input gates is an input gate of the circuit, i.e., a variable or a constant. These circuits exactly correspond to algebraic branching programs. Even for skew arithmetical circuits, no polynomial time algorithm is currently known (although the problem belongs to coRNC).

Full proofs can be found in the long version [14].

2 Arithmetical Circuits

We use the standard notion of (division-free) arithmetical circuits. Let us fix a set $X = \{x_1, x_2, \ldots\}$ of variables. An *arithmetical circuit* is a triple $C = (V, S, \mathsf{rhs})$, where (i) V is a finite set of *gates*, (ii) $S \in V$ is the *output gate*, and (iii) for every gate A, $\mathsf{rhs}(A)$ (the *right-hand side of* A) is either a variable from X, one of the constants -1, 0, 1, or an expression of the form $B + C$ (then A is an addition gate) or $B \cdot C$ (then A is a multiplication gate), where B and C are gates. Moreover, there must exist a linear order $<$ on V such that $B < A$ whenever B occurs in $\mathsf{rhs}(A)$. A gate A with $\mathsf{rhs}(A) \in X \cup \{0, -1, 1\}$ is an *input gate*. Over a fixed ring $(R, +, \cdot)$ (which will be $(\mathbb{Z}, +, \cdot)$ in most cases) we can evaluate every gate $A \in V$ to a polynomial $\mathsf{val}_{\mathcal{C}}(A)$ with coefficients from R and variables from X (val stands for "value"). Moreover let $\mathsf{val}(\mathcal{C}) = \mathsf{val}_{\mathcal{C}}(S)$ be the polynomial to which $\mathcal{C}$ evaluates. Two arithmetical circuits $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent over the ring $(R, +, \cdot)$ if $\mathsf{val}(\mathcal{C}_1) = \mathsf{val}(\mathcal{C}_2)$.

Fix an arithmetical circuit $\mathcal{C} = (V, S, \mathsf{rhs})$. We can view $\mathcal{C}$ as a directed acyclic graph (dag), where every node is labelled with a variable or a constant or an operator $+$, $\cdot$. If $\mathsf{rhs}(A) = B \circ C$ (for $\circ$ one of the operators), then there is an edge from B to A and C to A. The *depth* $\mathsf{depth}(A)$ (resp., *multiplication depth* $\mathsf{mdepth}(A)$) of the gate A is the maximal number of gates (resp., multiplication gates) along a path from an input gate to A. So, input gates have depth one and multiplication depth zero. The *depth* (resp., *multiplication depth*) of $\mathcal{C}$ is $\mathsf{depth}(\mathcal{C}) = \mathsf{depth}(S)$ (resp., $\mathsf{mdepth}(\mathcal{C}) = \mathsf{mdepth}(S)$). The *formal degree* $\mathsf{deg}(A)$ of a gate A is 1 if A is an input gate, $\max\{\mathsf{deg}(B), \mathsf{deg}(C)\}$ if $\mathsf{rhs}(A) = B+C$, and $\mathsf{deg}(B) + \mathsf{deg}(C)$ if $\mathsf{rhs}(A) = B \cdot C$. The formal degree of $\mathcal{C}$ is $\mathsf{deg}(\mathcal{C}) = \deg(S)$. A *positive circuit* is an arithmetical circuit without input gates labelled by the constant -1. An *addition circuit* is a positive circuit without multiplication gates. A *variable-free circuit* is a circuit without variables. It evaluates to an element of the underlying ring. A *skew circuit* is an arithmetical circuit such that for every multiplication gate A with $\mathsf{rhs}(A) = B \cdot C$, one of the gates B, C is an input gate.

In the rest of the paper we will also allow more complicated expressions in right-hand sides for gates. For instance, we may have a gate with $\mathsf{rhs}(A) = (B + C) \cdot (D + E)$. When writing down such a right-hand side, we implicitly assume that there are additional gates in the circuit, with (in our example) right hand sides $B + C$ and $D + E$, respectively. The proof of the following lemma uses standard ideas.

Lemma 1. *Given an arithmetical circuit $\mathcal{C}$ one can compute in logspace positive circuits $\mathcal{C}_1, \mathcal{C}_2$ such that* $\mathsf{val}(\mathcal{C}) = \mathsf{val}(\mathcal{C}_1) - \mathsf{val}(\mathcal{C}_2)$ *for every ring. Moreover, for $i \in \{1, 2\}$ we have* $\deg(\mathcal{C}_i) \leq \deg(\mathcal{C})$*,* $\mathsf{depth}(\mathcal{C}_i) \leq 2 \cdot \mathsf{depth}(\mathcal{C})$*, and* $\mathsf{mdepth}(\mathcal{C}_i) \leq \mathsf{mdepth}(\mathcal{C})$*.*

Polynomial identity testing (PIT) for a ring R is the following computational problem: Given an arithmetical circuit $\mathcal{C}$ (with variables $x_1, \ldots, x_n$), does $\mathsf{val}(\mathcal{C}) = 0$ hold, i.e., does $\mathcal{C}$ evaluate to the zero-polynomial in $R[x_1, \ldots, x_n]$? It is an outstanding open problem in complexity theory, whether PIT for $\mathbb{Z}$ can be solved in polynomial time.

3 Complexity Classes

The counting class $\#\mathsf{L}$ consists of all functions $f : \Sigma^* \to \mathbb{N}$ for which there is a logspace bounded nondeterministic Turing machine M such that for every $w \in \Sigma^*$, $f(w)$ is the number of accepting computation paths of M on input x. The class $\mathsf{C}_=\mathsf{L}$ contains all languages A for which there are two functions $f_1, f_2 \in \#\mathsf{L}$ such that for every $w \in \Sigma^*$, $w \in A$ if and only if $f_1(w) = f_2(w)$. The class $\mathsf{C}_=\mathsf{L}$ is closed under logspace many-one reductions. The canonical $\mathsf{C}_=\mathsf{L}$-complete problem is the following: The input consists of two dags G_1 and G_2 and vertices s_1, t_1 (in G_1) and s_2, t_2 (in G_2), and it is asked whether the number of paths from s_1 to t_1 in G_1 is equal to the number of paths from s_2 to t_2 in G_2. A reformulation of this problem is: Given two variable-free addition circuits $\mathcal{C}_1$ and $\mathcal{C}_2$, does $\mathsf{val}(\mathcal{C}_1) = \mathsf{val}(\mathcal{C}_2)$ hold?

We use standard definitions concerning circuit complexity, see e.g. [26]. In particular we will consider the class TC^0 of all problems that can be solved by a polynomial size circuit family of constant depth that uses NOT-gates and unbounded fan-in AND-gates, OR-gates, and majority-gates. For $\mathsf{DLOGTIME}$-uniform TC^0 it is required in addition that for binary coded gate numbers u and v, one can (i) compute the type of gate u in time $O(|u|)$ and (ii) check in time $O(|u| + |v|)$ whether u is an input gate for v. Note that the circuit for inputs of length n has at most $p(n)$ gates for a polynomial $p(n)$. Hence, the binary codings u and v have length $O(\log n)$, i.e., the above computations can be done in $\mathsf{DTIME}(\log n)$. This is the reason for using the term "$\mathsf{DLOGTIME}$-uniform". If majority gates are not allowed, we obtain the class ($\mathsf{DLOGTIME}$-uniform) AC^0. The class ($\mathsf{DLOGTIME}$-uniform) NC^1 is defined by ($\mathsf{DLOGTIME}$-uniform) polynomial size circuit families of logarithmic depth that use NOT-gates and fan-in-2 AND-gates and OR-gates. A language A is AC^0-reducible to languages $B_1, \ldots, B_k$ if A can be solved with a $\mathsf{DLOGTIME}$-uniform polynomial size circuit family of constant depth that uses NOT-gates and unbounded fan-in AND-gates, OR-gates, and B_i-gates ($1 \leq i \leq k$). Here, a B_i-gate (it is also called an oracle gate) receives an ordered tuple of inputs $x_1, x_2, \ldots, x_n$ and outputs 1 if and only if $x_1 x_2 \cdots x_n \in B_i$. Sometimes, also the term "uniform constant depth reducibility" is used for this type of reductions. In the same way, the weaker NC^1-reducibility can be defined. Here, one counts the depth of a B_i-gate with inputs $x_1, x_2, \ldots, x_n$ as $\log n$. The class DET contains all problems that are NC^1-reducible to the computation of the determinant of an integer matrix, see [8]. It is known that $\mathsf{C}_=\mathsf{L} \subseteq \mathsf{DET} \subseteq \mathsf{NC}^2$, see e.g. [4, Sec.4].

An *NAuxPDA* is a nondeterministic Turing machine with an additional pushdown store. The class $\mathsf{LogCFL} \subseteq \mathsf{NC}^2$ is the class of all languages that can be accepted by a polynomial time bounded NAuxPDA whose work tape is logarithmically bounded (but the pushdown store is unbounded). If we assign to the input the number of accepting computation paths of such an NAuxPDA, we obtain the counting class $\#\mathsf{LogCFL}$. In [25] it is shown that a function $f : \{0,1\}^* \to \mathbb{N}$ belongs to $\#\mathsf{LogCFL}$ if and only if there exists a logspace-uniform family $(\mathcal{C}_n)_{n\geq 1}$ of positive arithmetic circuits such that $\mathcal{C}_n$ computes the mapping f restricted to $\{0,1\}^n$ and there is a polynomial $p(n)$ such that the formal degree of $\mathcal{C}_n$ is bounded by $p(n)$. The class $\mathsf{C}_=\mathsf{LogCFL}$ contains all languages A for which there are two functions $f_1, f_2 \in \#\mathsf{LogCFL}$ such that for every $w \in \Sigma^*$, $w \in A$ if and only if $f_1(w) = f_2(w)$. We need the following lemma, whose proof is based on folklore ideas:

Lemma 2. *There is an NAuxPDA $\mathcal{P}$ that gets as input a positive variable-free arithmetic circuit $\mathcal{C}$ and such that the number of accepting computations of $\mathcal{P}$ on input $\mathcal{C}$ is $\mathsf{val}(\mathcal{C})$. Moreover, the running time is bounded polynomially in $\mathsf{depth}(\mathcal{C}) \cdot \mathsf{deg}(\mathcal{C})$.*

4 Matrices and Groups

In this paper we are concerned with certain subclasses of *linear groups*. A group is linear if it is isomorphic to a subgroup of $\mathsf{GL}_d(F)$ (the group of all invertible $(d \times d)$-matrices over the field F) for some field F.

A (n-step) solvable group G is a group G, which has a a subnormal series $G = G_n \rhd G_{n-1} \rhd G_{n-2} \rhd \cdots \rhd G_1 \rhd G_0 = 1$ (i.e., G_i is a normal subgroup of G_{i+1} for all $0 \leq i \leq n-1$) such that every quotient G_{i+1}/G_i is abelian ($0 \leq i \leq n-1$). If every quotient G_{i+1}/G_i is cyclic, then G is called *polycyclic*. The number of $0 \leq i \leq n-1$ such that $G_{i+1}/G_i \cong \mathbb{Z}$ is called the *Hirsch length* of G; it does not depend on the chosen subnormal series. If $G_{i+1}/G_i \cong \mathbb{Z}$ for all $0 \leq i \leq n-1$ then G is called *strongly polycyclic*. A group is polycyclic if and only if it is solvable and every subgroup is finitely generated. Polycyclic groups are linear. Auslander and Swan [5,24] proved that the polycyclic groups are exactly the solvable groups of integer matrices.

For a group G its *lower central series* is the series $G = G_1 \rhd G_2 \rhd G_3 \rhd \cdots$ of subgroups, where $G_{i+1} = [G_i, G]$, which is the subgroup generated by all commutators $[g, h]$ with $g \in G_i$ and $h \in G$. Indeed, G_{i+1} is a normal subgroup of G_i. The group G is *nilpotent*, if its lower central series terminates after finitely many steps in the trivial group 1. Every f.g. nilpotent group is polycyclic.

Let G be a f.g. group and let G be finitely generated as a group by Σ. Then, as a monoid G is finitely generated by $\Sigma \cup \Sigma^{-1}$ (where $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$ is a disjoint copy of Σ and a^{-1} stands for the inverse of the generator $a \in \Sigma$). Recall that the *word problem* for G is the following computational problem: Given a string $w \in (\Sigma \cup \Sigma^{-1})^*$, does w evaluate to the identity of G. Kharlampovich proved that there exist finitely presented 3-step solvable groups with an undecidable word problem. On the other hand, for every f.g. linear group the word problem can be solved in deterministic logarithmic space by results of Lipton and Zalcstein [16] and Simon [23]. This applies in particular to polycyclic groups. Robinson proved in his thesis that the word problem for a polycyclic group belongs to TC^0 [21], but his circuits are not uniform. Waack considered in [27] arbitrary f.g. solvable linear groups (which include the polycyclic groups) and proved that their word problems belong to logspace-uniform NC^1. In [14] we combine Waack's technique with the famous division breakthrough result by Hesse, Allender, and Barrington [9] to show that for every f.g. solvable linear group the word problem belongs to $\mathsf{DLOGTIME}$-uniform TC^0.

5 Straight-Line Programs and the Compressed Word Problem

A straight-line program (briefly, SLP) is basically a multiplicative circuit over a monoid. We define an SLP over the finite alphabet Σ as a triple $\mathcal{G} = (V, S, \mathsf{rhs})$, where V is a finite set of variables (or gates), $S \in V$ is the start variable (or output gate), and rhs maps every variable to a right-hand side $\mathsf{rhs}(A)$, which is either a symbol $a \in \Sigma$, or of the form BC, where $B, C \in V$. As for arithmetical circuits we require that there is a linear order $<$ on V such that $B < A$, whenever B occurs in $\mathsf{rhs}(A)$. The terminology "(start) variable" (instead of "(output) gate") comes from the fact that an SLP is quite often defined as a context-free grammar that produces a single string over Σ. This string is defined in the obvious way by iteratively replacing variables by the corresponding right-hand sides, starting with the start variable. We denote this string with $\mathsf{val}(\mathcal{G})$. The unique string over Σ, derived from the variable $A \in V$, is denoted with $\mathsf{val}_{\mathcal{G}}(A)$. We will also allow more general right-hand sides from $(V \cup \Sigma)^*$, but by introducing new variables we can always obtain an equivalent SLP in the above form.

If we have a monoid M, which is finitely generated by the set Σ, then there exists a canonical monoid homomorphism $h : \Sigma^* \to M$. Then, an SLP $\mathcal{G}$ over the alphabet Σ can be evaluated over the monoid M, which yields the monoid element $h(\mathsf{val}(\mathcal{G}))$. In this paper, we are only interested in the case that the monoid M is a f.g. group G. Let G be finitely generated as a group by Σ. An SLP over the alphabet $\Sigma \cup \Sigma^{-1}$ is also called an SLP over the group G. In this case, we will quite often identify the string $\mathsf{val}(\mathcal{G}) \in (\Sigma \cup \Sigma^{-1})^*$ with the group element $g \in G$ to which it evaluates. We will briefly write "$\mathsf{val}(\mathcal{G}) = g$ in G" in this situation.

The main computational problem we are interested in is the *compressed word problem* for a f.g. group G (with a finite generating set Σ), briefly $\mathsf{CWP}(G)$. The input for this problem is an SLP $\mathcal{G}$ over the alphabet $\Sigma \cup \Sigma^{-1}$, and it is asked whether $\mathsf{val}(\mathcal{G}) = 1$ in G (where of course 1 denotes the group identity). The term "compressed word problem" comes from the fact that this problem can be seen as a succinct version of the classical word problem for G, where the input is an explicitly given string $w \in (\Sigma \cup \Sigma^{-1})^*$ instead of an SLP-compressed string.

The compressed word problem is related to the classical word problem. For instance, the classical word problem for a f.g. subgroup of the automorphism group of a group G can be reduced to the compressed word problem for G, and similar results are known for certain group extensions, see [19] for more details. There are several important classes of groups, for which the compressed word problem can be solved in polynomial time, and for finitely generated linear groups the compressed word problem belongs to co-randomized polynomial time, see the introduction. In [6] the parallel complexity of the compressed word problem (there, called the circuit evaluation problem) for finite groups was studied, and the following result was shown:

Theorem 1 ([6]). *Let G be a finite group. If G is solvable, then $\mathsf{CWP}(G)$ belongs to the class $\mathsf{DET} \subseteq \mathsf{NC}^2$. If G is not solvable, then $\mathsf{CWP}(G)$ is P-complete.*

6 CWP for Finitely Generated Nilpotent Groups

In [19] it was shown that the compressed word problem for a finitely generated nilpotent group can be solved in polynomial time. The main result of this section is:

Theorem 2. *Let $G \neq 1$ be a f.g. torsion-free nilpotent group. Then $\mathsf{CWP}(G)$ is complete for the class $\mathsf{C}_=\mathsf{L}$.*

For the lower bound let G be a non-trivial f.g. torsion-free nilpotent group. Since $G \neq 1$, G contains $\mathbb{Z}$. Hence, it suffices to prove the following:

Lemma 3. *$\mathsf{CWP}(\mathbb{Z})$ is hard for $\mathsf{C}_=\mathsf{L}$.*

Proof. An SLP $\mathcal{G}$ over the generator 1 of $\mathbb{Z}$ and its inverse -1 is nothing else than a variable-free arithmetical circuit $\mathcal{C}$ without multiplication gates. Using Lemma 1 we can construct in logspace two addition circuits $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathsf{val}(\mathcal{C}) = 0$ if and only if $\mathsf{val}(\mathcal{C}_1) = \mathsf{val}(\mathcal{C}_2)$. Checking the latter is complete for $\mathsf{C}_=\mathsf{L}$ as remarked in Sec. 3. $\square$

For the upper bound in Thm. 2, we use the fact that every torsion-free f.g. nilpotent group can be represented by unitriangluar integer matrices. Let A be a $(d \times d)$-matrix over $\mathbb{Z}$. With $A[i,j]$ we denote the entry of A in row i and column j. The matrix A is *triangular* if $A[i,j] = 0$ whenever $i > j$, i.e., all entries below the main diagonal are 0. A *unitriangular matrix* is a triangular matrix A such that $A[i,i] = 1$ for all $1 \leq i \leq d$, i.e., all entries on the main diagonal are 1. We denote the set of unitriangular $(d \times d)$-matrices over $\mathbb{Z}$ with $\mathsf{UT}_d(\mathbb{Z})$. This is a group with respect to matrix multiplication. Let $1 \leq i < j \leq d$. With $T_{i,j}$ we denote the matrix from $\mathsf{UT}_d(\mathbb{Z})$ such that $T_{i,j}[i,j] = 1$ and $T_{i,j}[k,l] = 0$ for all k,l with $1 \leq k < l \leq d$ and $(k,l) \neq (i,j)$. The notation $T_{i,j}$ does not specify the dimension d of the matrix, but the dimension will be always clear from the context. The group $\mathsf{UT}_d(\mathbb{Z})$ is generated by the finite set $\Gamma_d = \{T_{i,i+1} \mid 1 \leq i < d\}$, see e.g. [7]. For every torsion-free f.g. nilpotent group G there exists some $d \geq 1$ such that $G \leq \mathsf{UT}_d(\mathbb{Z})$ [13, Thm.17.2.5]. Hence, the upper bound in Thm. 2 follows from:

Lemma 4. *For every $d \geq 1$, $\mathsf{CWP}(\mathsf{UT}_d(\mathbb{Z}))$ belongs to $\mathsf{C_{=}L}$.*

For the rest of this section let us fix a number $d \geq 1$ and consider the unitriangluar matrix group $\mathsf{UT}_d(\mathbb{Z})$. Consider an SLP $\mathcal{G} = (V, S, \mathsf{rhs})$ over the alphabet $\Gamma_d \cup \Gamma_d^{-1}$, where Γ_d is the finite generating set of $\mathsf{UT}_d(\mathbb{Z})$ from Sec. 4. Note that for every variable $A \in V$, $\mathsf{val}_{\mathcal{G}}(A)$ is a word over the alphabet $\Gamma_d \cup \Gamma_d^{-1}$. We identify in the following this word with the matrix to which it evaluates. Thus, $\mathsf{val}_{\mathcal{G}}(A) \in \mathsf{UT}_d(\mathbb{Z})$.

Assume we have given an arithmetical circuit $\mathcal{C}$. A partition $\biguplus_{i=1}^m V_i$ of the set of all multiplication gates of $\mathcal{C}$ is called *structure-preserving* if for all multiplication gates u, v of $\mathcal{C}$ the following holds: If there is a non-empty path from u to v in (the dag corresponding to) $\mathcal{C}$ then there exist $1 \leq i < j \leq d$ such that $u \in V_i$ and $v \in V_j$. In a first step, we transform our SLP $\mathcal{G}$ in logspace into a variable-free arithmetical circuit $\mathcal{C}$ of multiplication depth at most d such that $\mathcal{G}$ evaluates to the identity matrix if and only if $\mathcal{C}$ evaluates to 0. Moreover, we also compute a structure-preserving partition of the multiplication gates of $\mathcal{C}$. This partition will be needed for the further computations. The degree bound in the following lemma will be needed in Sec. 7.

Lemma 5. *From the SLP $\mathcal{G} = (V, S, \mathsf{rhs})$ we can compute in logspace a variable-free arithmetical circuit $\mathcal{C}$ with $\mathsf{mdepth}(\mathcal{C}) \leq d$ and $\deg(\mathcal{C}) \leq 2(d-1)$, such that $\mathsf{val}(\mathcal{G}) = \mathsf{Id}_d$ if and only if $\mathsf{val}(\mathcal{C}) = 0$. In addition we can compute in logspace a structure-preserving partition $\biguplus_{i=1}^d V_i$ of the set of all multiplication gates of $\mathcal{C}$.*

Proof. The set of gates of $\mathcal{C}$ is $W = \{A_{i,j} \mid A \in V, 1 \leq i < j \leq d\} \uplus \{T\}$, where T is the output gate. The idea is simple: Gate $A_{i,j}$ will evaluate to the matrix entry $\mathsf{val}_{\mathcal{G}}(A)[i,j]$. To achieve this, we define the right-hand side mapping of the circuit $\mathcal{G}$ (which we denote again with rhs) as follows: If $\mathsf{rhs}(A) = M \in \Gamma_d \cup \Gamma_d^{-1}$, then $\mathsf{rhs}(A_{i,j}) = M[i,j] \in \{-1, 0, 1\}$, and if $\mathsf{rhs}(A) = BC$, then $\mathsf{rhs}(A_{i,j}) = B_{i,j} + C_{i,j} + \sum_{i<k<j} B_{i,k} \cdot C_{k,j}$ (which is the rule for matrix multiplication taking into account that all matrices are unitriangular). Finally, we set $\mathsf{rhs}(T) = \sum_{1 \leq i < j \leq d} S_{i,j}^2$. Then, $\mathsf{val}(\mathcal{C}) = 0$ iff $\mathsf{val}_{\mathcal{G}}(S)[i,j] = 0$ for all $1 \leq i < j \leq d$ iff $\mathsf{val}(\mathcal{G})$ is the identity matrix.

Concerning the multiplication depth, note that the multiplication depth of the gate $A_{i,j}$ is bounded by $j - i$: The only multiplications in $\mathsf{rhs}(A_{i,j})$ are of the form $B_{i,k}C_{k,j}$ (and these multiplications are not nested). Hence, by induction, the multiplication depth of $A_{i,j}$ is bounded by $1 + \max\{k-i, j-k \mid i < k < j\} =$

$j-i$. It follows that every gate $S_{i,j}$ has multiplication depth at most $d-1$, which implies that the output gate T has multiplication depth at most d. Similarly, it can be shown by induction that $\mathsf{deg}(A_{i,j}) \leq j-i$. Hence, $\mathsf{deg}(A_{i,j}) \leq d-1$ for all $1 \leq i < j \leq d$, which implies that the formal degree of the circuit is bounded by $2(d-1)$.

The structure-preserving partition $\biguplus_{i=1}^{d} V_i$ of the set of all multiplication gates of $\mathcal{C}$ can be defined as follows: All gates corresponding to multiplications $B_{i,k} \cdot C_{k,j}$ in $\mathsf{rhs}(A_{i,j})$ are put into the set V_{j-i}. Finally, all gates corresponding to multiplications $S_{i,j}^2$ in $\mathsf{rhs}(T)$ are put into V_d. It is obvious that this partition is structure-preserving. □

In a second step we apply Lemma 1 and construct from the above circuit $\mathcal{C}$ two variable-free positive circuits $\mathcal{C}_1$ and $\mathcal{C}_2$, both having multiplication depth at most d such that $\mathsf{val}(\mathcal{C}) = \mathsf{val}(\mathcal{C}_1) - \mathsf{val}(\mathcal{C}_2)$. Hence, our input SLP $\mathcal{G}$ evaluates to the indentity matrix if and only if $\mathsf{val}(\mathcal{C}_1) = \mathsf{val}(\mathcal{C}_2)$. Moreover, using the construction from Lemma 1 it is straightforward to compute in logspace a structure-preserving partition $\biguplus_{i=1}^{d} V_{k,i}$ of the the set of all multiplication gates of $\mathcal{C}_k$ $(k \in \{1,2\})$.

The following lemma concludes the proof that $\mathsf{CWP}(\mathsf{UT}_d(\mathbb{Z}))$ belongs to $\mathsf{C_{=}L}$. For the proof one eliminates in a single phase all multiplication gates in a layer. This can be achieved by a logspace reduction, and since the total number of layers is constant, the whole elimination procedure works in logspace.

Lemma 6. *Let d be constant. From a given variable-free positive circuit $\mathcal{C}$ of multiplication depth d together with a structure-preserving partition $\biguplus_{i=1}^{d} V_i$ of the set of all multiplication gates of $\mathcal{C}$, we can compute in logarithmic space a variable-free addition circuit $\mathcal{D}$ such that* $\mathsf{val}(\mathcal{C}) = \mathsf{val}(\mathcal{D})$.

So far, we have restricted to *torsion-free* f.g. nilpotent groups. For general f.g. nilpotent groups, we use the fact that every f.g. nilpotent group contains a torsion-free normal f.g. nilpotent subgroup of finite index [13, Thm. 17.2.2], in order to show that the compressed word problem for every f.g. nilpotent group belongs to the complexity class DET: To do this we need the following result. For the proof one can adopt the proof of [19, Thm.4.4], where the statement is shown for polynomial time many-one reducibility instead of AC^0-reducibility.

Theorem 3. *Let G be a finitely generated group. For every normal subgroup H of G with a finite index,* $\mathsf{CWP}(G)$ *is* AC^0*-reducible to* $\mathsf{CWP}(H)$ *and* $\mathsf{CWP}(G/H)$.

We can now show:

Theorem 4. *For every f.g. nilpotent group, the compressed word problem is in* DET.

Proof. Let G be a f.g. nilpotent group. If G is finite, then the result follows from Thm. 1 (every nilpotent group is solvable). If G is infinite, then G has a f.g. torsion-free normal subgroup H of finite index [13, Thm. 17.2.2]. Subgroups and quotients of nilpotent groups are nilpotent too [22, Chapter5], hence H and G/H are nilpotent; moreover H is finitely generated. By Thm. 2, $\mathsf{CWP}(H)$ belongs to

$\mathsf{C}_=\mathsf{L} \subseteq \mathsf{DET}$. Moreover, by Thm. 1, $\mathsf{CWP}(G/H) \in \mathsf{DET}$ as well. Finally, Thm. 3 implies $\mathsf{CWP}(G) \in \mathsf{DET}$. $\square$

Actually, Thm. 4 can be slightly extended to groups that are (f.g. nilpotent)-by-(finite solvable) (i.e., groups that have a normal subgroup, which is f.g. nilpotent, and where the quotient is finite solvable. This follows from Thm. 3 and the fact that the compressed word problem for a finite solvable group belongs to DET (Thm. 1).

7 The Uniform CWP for Unitriangular Groups

For Lemma 4 it is crucial that the dimension d is a constant. In this section, we consider a uniform variant of the compressed word problem for $\mathsf{UT}_d(\mathbb{Z})$. We denote this problem with $\mathsf{CWP}(\mathsf{UT}_*(\mathbb{Z}))$. The input consists of a unary encoded number d and an SLP, whose terminal symbols are generators of $\mathsf{UT}_d(\mathbb{Z})$ or their inverses. Alternatively, we can assume that the terminal symbols are arbitrary matrices from $\mathsf{UT}_d(\mathbb{Z})$ with binary encoded entries (given such a matrix M, it is easy to construct an SLP over the generator matrices that produces M). The question is whether the SLP evaluates to the identity matrix. We show that this problem is complete for the complexity class $\mathsf{C}_=\mathsf{LogCFL}$.

Theorem 5. *The problem* $\mathsf{CWP}(\mathsf{UT}_*(\mathbb{Z}))$ *is complete for* $\mathsf{C}_=\mathsf{LogCFL}$.

Proof. We start with the upper bound. Consider an SLP $\mathcal{G}$, whose terminal symbols are generators of $\mathsf{UT}_d(\mathbb{Z})$ or their inverses. The dimension d is clearly bounded by the input size. Consider the variable-free arithmetic circuit $\mathcal{C}$ constructed from $\mathcal{G}$ in Lemma 5 and let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the two variable-free positive arithmetic circuits obtained from $\mathcal{C}$ using Lemma 1. Then $\mathcal{G}$ evaluates to the identity matrix if and only if $\mathsf{val}(\mathcal{C}_1) = \mathsf{val}(\mathcal{C}_2)$. Moreover, the formal degrees $\deg(\mathcal{C}_1)$ and $\deg(\mathcal{C}_2)$ are bounded by $2(d-1)$, i.e., polynomially bounded in the input length. Finally, we compose a logspace machine that computes from the input SLP $\mathcal{G}$ the circuit $\mathcal{C}_i$ with the NAuxPDA from Lemma 2 to get an NAuxPDA $\mathcal{P}_i$ such that the number of accepting computation paths of $\mathcal{P}_i$ on input $\mathcal{G}$ is exactly $\mathsf{val}(\mathcal{C}_i)$. Moreover, the running time of $\mathcal{P}_i$ on input $\mathcal{G}$ is bounded polynomially in $(2d-1)\cdot\mathsf{depth}(\mathcal{C}_i) \in O(d\cdot|\mathcal{G}|)$.

Let us now show that $\mathsf{CWP}(\mathsf{UT}_*(\mathbb{Z}))$ is hard for $\mathsf{C}_=\mathsf{LogCFL}$. Let $(\mathcal{C}_{1,n})_{n\geq 0}$ and $(\mathcal{C}_{2,n})_{n\geq 0}$ be two logspace-uniform families of positive arithmetical circuits of polynomially bounded size and formal degree. Let $w = a_1a_2\cdots a_n \in \{0,1\}^n$ be an input for the circuits $\mathcal{C}_{1,n}$ and $\mathcal{C}_{2,n}$. Let $\mathcal{C}_i$ be the variable-free positive arithmetical circuit obtained from $\mathcal{C}_{i,n}$ by replacing every x_j-labelled input gate by $a_j \in \{0,1\}$. By [3, Lemma 3.2] we can assume that every gate of $\mathcal{C}_i$ is labelled by its formal degree. By adding if necessary additional multiplication gates, where one input is set to 1, we can assume that $\mathcal{C}_1$ and $\mathcal{C}_2$ have the same formal degree $d \leq p(n)$ for a polynomial p. Analogously, we can assume that if A is an addition gate in $\mathcal{C}_1$ or $\mathcal{C}_2$ with right-hand side $B + C$, then $\deg(B) = \deg(C) = \deg(A)$. All these preprocessing steps can be carried out in logarithmic space.

We will construct in logarithmic space an SLP $\mathcal{G}$ over the alphabet $\Gamma_{d+1} \cup \Gamma_{d+1}^{-1}$, where Γ_{d+1} is our canonical generating set for the matrix group $\mathsf{UT}_{d+1}(\mathbb{Z})$, such that $\mathcal{G}$ evaluates to the identity matrix if and only if $\mathsf{val}(\mathcal{C}_1) = \mathsf{val}(\mathcal{C}_2)$. Let v_i be the output value of $\mathcal{C}_i$. We first construct in logspace an SLP $\mathcal{G}_1$ that evaluates to the matrix $T_{1,d}^{v_1}$. In the same way we can construct in logspace a second SLP $\mathcal{G}_2$ that evaluates to $T_{1,d}^{-v_2}$. Then, by concatenating the two SLPs $\mathcal{G}_1$ and $\mathcal{G}_2$ we obtain the desired SLP.

The variables of $\mathcal{G}_1$ are $A_{i,j}^b$, where A is a gate of $\mathcal{C}_1$, $b \in \{-1, 1\}$, and $1 \le i < j \le d$ such that $j - i$ is the formal degree of A. The SLP $\mathcal{G}_1$ will be constructed in such a way that $\mathsf{val}_{\mathcal{G}_1}(A_{i,j}^b) = T_{i,j}^{b \cdot v}$, where $v = \mathsf{val}_{\mathcal{C}_1}(A)$. If $\mathsf{rhs}_{\mathcal{C}_1}(A) = 0$, then we set $\mathsf{rhs}_{\mathcal{G}_1}(A_{i,j}^b) = \mathsf{Id}$ and if $\mathsf{rhs}_{\mathcal{C}_1}(A) = 1$, then we set $\mathsf{rhs}_{\mathcal{G}_1}(A_{i,j}^b) = T_{i,j}^b$. If $\mathsf{rhs}_{\mathcal{C}_1}(A) = B + C$, then we set $\mathsf{rhs}_{\mathcal{G}_1}(A_{i,j}^b) = B_{i,j}^b C_{i,j}^b$. Correctness follows immediately by induction. Note that $\deg(B) = \deg(C) = \deg(A) = j - i$, which implies that the gates $B_{i,j}^b$ and $C_{i,j}^b$ exist. Finally, if $\mathsf{rhs}_{\mathcal{C}_1}(A) = B \cdot C$, then we set $\mathsf{rhs}_{\mathcal{G}_1}(A_{i,j}^1) = B_{i,k}^{-1} C_{k,j}^{-1} B_{i,k}^1 C_{k,j}^1$ and $\mathsf{rhs}_{\mathcal{G}_1}(A_{i,j}^{-1}) = C_{k,j}^{-1} B_{i,k}^{-1} C_{k,j}^1 B_{i,k}^1$, where k is such that $\deg(B) = k - i$ and $\deg(B) = j - k$. Such a k exists since $j - i = \deg(A) = \deg(B) + \deg(C)$. Correctness follows by induction and the simple fact that $T_{i,j}^{-a}, T_{j,k}^{-b} T_{i,j}^{a}, T_{j,k}^{b} = T_{i,k}^{ab}$ for all $a, b \in \mathbb{Z}$ and $1 \le i < j < k \le d$; see [20]. $\square$

8 CWP for Polycyclic Groups

In this section we look at the compressed word problem for polycyclic groups. Since every polycyclic group is f.g. linear, the compressed word problem for a polycyclic group can be reduced to PIT. Here, we show a lower bound: There is a polycyclic group G such that PIT for skew arithmetical circuits can be reduced to $\mathsf{CWP}(G)$. In this context, it is interesting to note that PIT for arbitrary circuits can be reduced to the compressed word problem to the linear (but not polycyclic) group $\mathsf{SL}_3(\mathbb{Z})$ [19, Thm.4.16].

Let us start with a specific example of a polycyclic group. Consider the two matrices

$$g_a = \begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix} \text{ and } h = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \tag{1}$$

where $a \in \mathbb{R}$, $a \ge 2$. Let $G_a = \langle g_a, h \rangle \le \mathsf{GL}_2(\mathbb{R})$. Let us remark that, for instance, the group G_2 is not polycyclic, see e.g. [28, p. 56]. On the other hand, we have:

Proposition 1. *The group $G = G_{1+\sqrt{2}}$ is polycyclic.*

The main result of this section is:

Theorem 6. *Let $a \ge 2$. Polynomial identity testing for skew arithmetical circuits is logspace-reducible to the compressed word problem for the group G_a.*

In particular, there exist polycyclic groups for which the compressed word problem is at least as hard as polynomial identity testing for skew circuits. Recall that it is not known, whether there exists a polynomial time algorithm for polynomial identity testing restricted to skew arithmetical circuits.

For the proof of Thm. 6, we use the following result from [2] (see the proof of [2, Prop. 2.2], where the result is shown for $a = 2$, but the proof works for any $a \geq 2$):

Lemma 7. *Let $\mathcal{C}$ be an arithmetical circuit of size n with variables $x_1, \ldots, x_m$ and let $p(x_1, \ldots, x_m) = \mathsf{val}(\mathcal{C})$. Let $a \geq 2$ be a real number. Then $p(x_1, \ldots, x_n)$ is the zero-polynomial if and only if $p(\alpha_1, \ldots, \alpha_n) = 0$, where $\alpha_i = a^{2^{i \cdot n^2}}$ for $1 \leq i \leq m$.*

Proof of Thereom 6. Let us fix a skew arithmetical circuit $\mathcal{C}$ of size n with m variables $x_1, \ldots, x_m$. We will define an SLP $\mathcal{G}$ over the alphabet $\{g_a, g_a^{-1}, h, h^{-1}\}$ such that $\mathsf{val}(\mathcal{G}) = \mathsf{Id}$ in G_a if and only if $\mathsf{val}(\mathcal{C}) = 0$. First of all, using iterated squaring, we can construct an SLP $\mathcal{H}$ with variables $A_1, A_1^{-1} \ldots, A_m, A_m^{-1}$ (and some other auxiliary variables) such that

$$\mathsf{val}_{\mathcal{H}}(A_i) = g_a^{2^{i \cdot n^2}} = \begin{pmatrix} \alpha_i & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathsf{val}_{\mathcal{H}}(A_i^{-1}) = g_a^{-2^{i \cdot n^2}} = \begin{pmatrix} \alpha_i^{-1} & 0 \\ 0 & 1 \end{pmatrix}.$$

We now construct the SLP $\mathcal{G}$ as follows: The set of variables of $\mathcal{G}$ consists of the gates of $\mathcal{C}$ and the variables of $\mathcal{H}$. We copy the right-hand sides from $\mathcal{H}$ and define the right-hand side for a gate A of $\mathcal{C}$ as follows: (i) $\mathsf{rhs}_{\mathcal{G}}(A) = h^n$ if $\mathsf{rhs}_{\mathcal{C}}(A) = n \in \{0, -1, 1\}$, (ii) $\mathsf{rhs}_{\mathcal{G}}(A) = BC$ if $\mathsf{rhs}_{\mathcal{C}}(A) = B + C$, and (iii) $\mathsf{rhs}_{\mathcal{G}}(A) = A_i B A_i^{-1}$ if $\mathsf{rhs}_{\mathcal{C}}(A) = x_i \cdot B$.

A straightforward induction shows that for every gate A of $\mathcal{C}$ we have the following, where we denote for better readability the polynomial $\mathsf{val}_{\mathcal{C}}(A)$ to which gate A evaluates with p_A:

$$\mathsf{val}_{\mathcal{G}}(A) = \begin{pmatrix} 1 & p_A(\alpha_1, \ldots, \alpha_n) \\ 0 & 1 \end{pmatrix}$$

We finally take the output gate S of the skew circuit $\mathcal{C}$ as the start variable of $\mathcal{G}$. Then, $\mathsf{val}(\mathcal{G})$ yields the identity matrix in the group G_a if and only if $p_S(\alpha_1, \ldots, \alpha_n) = 0$. By Lemma 7 this is equivalent to $\mathsf{val}(\mathcal{C}) = p_S(x_1, \ldots, x_n) = 0$. $\square$

Actually, we can carry out the above reduction for a class of arithmetical circuits that is slightly larger than the class of skew arithmetical circuits. Let us define a *powerful skew circuit* as an arithmetical circuit, where for every multiplication gate A, $\mathsf{rhs}(A)$ is of the form $x_i^e \cdot B$, where $e \geq 0$ is a binary coded number. Such a circuit can be converted into an ordinary arithmetical circuit, which, however is no longer skew. To extend the reduction from the proof of Thereom 6 to powerful skew circuits, we set for a gate A with $\mathsf{rhs}_{\mathcal{C}}(A) = x_i^e \cdot B$: $\mathsf{rhs}_{\mathcal{G}}(A) = A_i^e B A_i^{-e}$. The powers A_i^e and A_i^{-e} can be defined using additional multiplication gates. In our recent paper [15], we introduced powerful skew circuits, and proved that for this class, PIT belongs to coRNC. We applied this result to the compressed word problem for wreath products.

Let us look again at the group $G = G_{1+\sqrt{2}}$ from Prop. 1. A closer inspection (see [14]) shows that $[G, G] \cong \mathbb{Z} \times \mathbb{Z}$ and $G/[G, G] \cong \mathbb{Z} \times \mathbb{Z}_2$. Hence, G has a subnormal series of the form $G \rhd H \rhd \mathbb{Z} \times \mathbb{Z} \rhd \mathbb{Z} \rhd 1$, where H has index 2 in G and $H/(\mathbb{Z} \times \mathbb{Z}) \cong \mathbb{Z}$. The group H is strongly polycyclic and has Hirsch length 3. By Thm. 3 we obtain:

Corollary 1. *There is a strongly polycyclic group H of Hirsch length 3 such that polynomial identity testing for skew circuits is polynomial time reducible to* $\mathsf{CWP}(H)$.

References

1. Allender, E., Beals, R., Ogihara, M.: The complexity of matrix rank and feasible systems of linear equations. Comput. Complex. **8**(2), 99–126 (1999)
2. Allender, E., Bürgisser, P., Kjeldgaard-Pedersen, J., Miltersen, P.B.: On the complexity of numerical analysis. SIAM J. Comput. **38**(5), 1987–2006 (2009)
3. Allender, E., Jiao, J., Mahajan, M., Vinay, V.: Non-commutative arithmetic circuits: Depth reduction and size lower bounds. Theor. Comput. Sci. **209**(1–2), 47–86 (1998)
4. Àlvarez, C., Jenner, B.: A very hard log-space counting class. Theor. Comput. Sci. **107**(1), 3–30 (1993)
5. Auslander, L.: On a problem of Philip Hall. Annals of Mathematics **86**(2), 112–116 (1967)
6. Beaudry, M., McKenzie, P., Péladeau, P., Thérien, D.: Finite monoids: From word to circuit evaluation. SIAM J. Comput. **26**(1), 138–152 (1997)
7. Biss, D.K., Dasgupta, S.: A presentation for the unipotent group over rings with identity. Journal of Algebra **237**(2), 691–707 (2001)
8. Cook, S.A.: A taxonomy of problems with fast parallel algorithms. Inform. Control **64**, 2–22 (1985)
9. Hesse, W., Allender, E., Barrington, D.A.M.: Uniform constant-depth threshold circuits for division and iterated multiplication. J. Comput. System Sci. **65**, 695–716 (2002)
10. Ibarra, O.H., Moran, S.: Probabilistic algorithms for deciding equivalence of straight-line programs. J. Assoc. Comput. Mach. **30**(1), 217–228 (1983)
11. Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: derandomizing the XOR lemma. In: Proc. STOC 1997, pp. 220–229. ACM Press (1997)
12. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Comput. Complex. **13**(1–2), 1–46 (2004)
13. Kargapolov, M.I., Merzljakov, J.I.: Fundamentals of the Theory of Groups. Springer (1979)
14. König, D., Lohrey, M.: Evaluating matrix circuits (2015). arXiv.org
15. König, D., Lohrey, M.: Parallel identity testing for algebraic branching programs with big powers and applications (2015). arXiv.org
16. Lipton, R.J., Zalcstein, Y.: Word problems solvable in logspace. J. Assoc. Comput. Mach. **24**(3), 522–526 (1977)
17. Lohrey, M.: Word problems and membership problems on compressed words. SIAM J. Comput. **35**(5), 1210–1240 (2006)

18. Lohrey, M.: Algorithmics on SLP-compressed strings: A survey. Groups Complexity Cryptology **4**(2), 241–299 (2012)
19. Lohrey, M.: The Compressed Word Problem for Groups. SpringerBriefs in Mathematics. Springer (2014)
20. Lohrey, M.: Rational subsets of unitriangular groups. International Journal of Algebra and Computation (2015). doi:10.1142/S0218196715400068
21. Robinson, D.: Parallel Algorithms for Group Word Problems. Ph.D. thesis, UCSD (1993)
22. Rotman, J.J.: An Introduction to the Theory of Groups, 4th edn. Springer (1995)
23. Simon, H.-U.: Word problems for groups and contextfree recognition. In: Proceedings of Fundamentals of Computation Theory, FCT 1979, pp. 417–422. Akademie-Verlag (1979)
24. Swan, R.: Representations of polycyclic groups. Proc. Am. Math. Soc. **18**, 573–574 (1967)
25. Vinay, V.: Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In: Proc. Structure in Complexity Theory Conference, pp. 270–284. IEEE Computer Society (1991)
26. Vollmer, H.: Introduction to Circuit Complexity. Springer (1999)
27. Waack, S.: On the parallel complexity of linear groups. ITA **25**(4), 265–281 (1991)
28. Wehrfritz, B.A.F.: Infinite Linear Groups. Springer (1977)

Computing and Graph

Approximation and Nonapproximability for the One-Sided Scaffold Filling Problem

Haitao Jiang(✉), Jingjing Ma, Junfeng Luan, and Daming Zhu

School of Computer Science and Technology, Shandong University, Jinan 250101, People's Republic of China
{htjiang,jfluan,dmzhu}@sdu.edu.cn, mjjdha@163.com

Abstract. Scaffold filling is an interesting combinatorial optimization problem from genome sequencing. The one-sided scaffold filling problem can be stated as: given an incomplete scaffold with some genes missing and a reference scaffold, the purpose is to insert the missing genes back into the incomplete scaffold(called "filling the scaffold"), such that the number of common adjacencies between the filled scaffold and the reference scaffold is maximized. This problem is NP-hard for genome with duplicated genes, and can be approximated within 1.25 by a very complicated combinatorial method. In this paper, we firstly improve the approximation factor to 6/5 by not-oblivious local search; then we show that this problem is MAX-SNP-complete.

1 Introduction

Motivation
Scaffold filling is a necessary step in the Next Generation Sequencing (NGS), which has greatly improved the speed of genome sequencing. Though the speed and quantity grows hugely by NGS, the accuracy is still not high enough. The bottleneck is that these sequences are often only a part of the complete genome, but concatenating them together to a whole genome, in general, still an intractable problem in the sense of computer algorithm. Currently, most sequencing results for genomes usually are in the form of scaffolds or contigs. Sometimes, applying these incomplete genomes for genomic analysis will introduce unnecessary errors. So it is natural to fill the missing gene fragments into the incomplete genome in a combinatorial way, and to obtain an 'augmented' genome which is closer to some reference genome.

Known Results
In reference [13], Muñoz *et al.* proposed the one-sided permutation scaffold filling problem, and devised an exact algorithm to minimize the genome rearrangement distance, i.e., the Double Cut and Join distance. Subsequently, Jiang *et al.* considered the permutation scaffold filling problem without any reference genome, they show that it is polynomially solvable under the breakpoint distance.

For genomes containing some duplicated genes, their scaffold filling becomes more difficult. Firstly, to measure the similarity between genomes with gene

DOI: 10.1007/978-3-319-21398-9_20

repetition, there are three general criterion: the exemplar genomic distance [14], the minimum common string partition (MCSP) distance [3] and the maximum number of common string adjacencies [1,11,12]. Unfortunately, unless P=NP, the former two criterion are hard to be computed and even hard to be approximated [2–4,7–9]. Therefore, the measure of the maximum number of common adjacenciesis is meaningful in itself.

Though the breakpoint distance on genomes with duplicated genes makes no sense, its complementary distance, the number of common adjacencies, formly defined by Jiang and Zhu [11,12], shows very good computable properties. Under the measure of common adjacencies, there are a series of related works. For the one-sided scaffold filling problem, Jiang *et al.* proved that its decision problem is NP-complete, and designed a 1.33-approximation algorithm with a greedy strategy [11,12]. Recently, Liu *et al.* improve the approximation factor to 1.25 by a combinatorial method, their key idea is a step to compute a maximum matching of a constructed bipartite graph, as well as a step of local improvement [15]. For the two-sided scaffold filling problem, as a generalization of the one-sided problem, it is also NP-hard. In [16], Liu *et al.* propose the first non-trivial polynomial time approximation algorithm with a ratio 3/2.

Our Contributions

1. An 6/5-approximation algorithm for the one-sided scaffold filling problem by a not-oblivious local search method. The "not-oblivious" local search was systematic described in [5]. The main idea is not improving the objective function directly , but improving another function that contains the same parameters to objective function but different indexes. With respect to local search, we just use 1-substitution;
2. We prove that the scaffold filling problem is MAX-SNP-complete by an L-reduction from the Maximum 3-Dimensional Matching problem [17].

2 Preliminaries

Firstly, we review some necessary definitions, which are also defined in [12]. Throughout this paper, we only consider unsigned genes and genomes. Given a gene family Σ, a string P is called *permutation* if each element in Σ appears exactly once in P. We use *c(P)* to denote the set of elements in permutation P. A string S is called *sequence* if some genes appear more than once in S, and $c(S)$ denotes genes of S, which is a multi-set of elements in Σ. For example, $\Sigma = \{a, b, c, d\}$, $S = acdabcd$, $c(S) = \{a, a, b, c, c, d, d\}$. A *scaffold* is an *sequence*, typically obtained by some sequencing and assembling process. A substring with m genes is called an *m-substring*, and a 2-substring is also called a *pair*, as the genes are unsigned, the relative order of the two genes of a pair does not matter, i.e., the pair xy is equal to the pair yx. Given a scaffold S=$s_1s_2s_3\cdots s_n$, let $PAIR_S = \{s_1s_2, s_2s_3, \ldots, s_{n-1}s_n\}$ be the set of pairs in S.

Given two scaffolds S=$s_1s_2\cdots s_n$ and T=$t_1t_2\cdots t_m$, if $s_is_{i+1} = t_jt_{j+1}$ (or $s_is_{i+1} = t_{j+1}t_j$),where $s_is_{i+1} \in PAIR_S$ and $t_jt_{j+1} \in PAIR_T$, we say that

s_is_{i+1} and t_jt_{j+1} (or $t_{j+1}t_j$) are matched to each other. Once s_is_{i+1} is matched to a pair t_jt_{j+1} in $PAIR_T$,it will never be matched to another pair t_kt_{k+1} $(k \neq j)$ in $PAIR_T$. Also, we claim that each pair can be matched at most one time, and any pair should be matched if possible.We denote the match between $PAIR_S$ and $PAIR_T$ as $R = \{(s,t)|\ s \text{ is matched to } t, \text{ and } s \in PAIR_S, t \in PAIR_T\}$. If the cardinality of R is maximized, we call it maximum match. We will base it to identify a pair in S and T as a common adjacency or a breakpoint.

Definition 1. *Given two scaffolds S=$s_1s_2 \cdots s_n$ and T=$t_1t_2 \cdots t_m$, let R be a maximum match between $PAIR_S$ and $PAIR_T$. A matched pair in S or T with respect to R is called an **adjacency**, and an unmatched pair with respect to R is called a **breakpoint** in S and T respectively.*

It follows from the definition that scaffolds S and T contain the same adjaciencies but distinct breakpoints. The maximum matched pairs in T(or equally, in S) is called the *adjacency set* between S and T, denoted as $a(S,T)$. By $b_S(S,T)$ and $b_T(S,T)$ we denote the set of breakpoints in S and T respectively. We will illuminate the above involved definitions by Fig.1.

$$\begin{aligned}
scaffold\ \ T &= \langle a\ c\ d\ a\ b\ c\ d\ \rangle \\
scaffold\ \ S &= \langle a\ b\ d\ c\ a\ c\ d\rangle \\
PAIR_T &= \{ac, cd, da, ab, bc, cd\} \\
PAIR_S &= \{ab, bd, dc, ca, ac, cd\} \\
matched\ \ pairs &: \ (ac \leftrightarrow ca), (cd \leftrightarrow dc), (ab \leftrightarrow ab), (cd \leftrightarrow cd) \\
a(S,T) &= \{ac, cd, ab, cd\} \\
b_T(S,T) &= \{da, bc\} \\
b_S(S,T) &= \{bd, ac\}
\end{aligned}$$

Fig. 1. An example for adjacency, breakpoint and related definitions

Given two scaffolds S=$s_1s_2 \cdots s_n$ and T=$t_1t_2 \cdots t_m$, as we can see, each gene except the four ending ones is involved in two adjacencies or two breakpoints or one adjacency and one breakpoint. To get rid of this unbalance, we add "$*$" to both the ends of S and T. Then, in the following part of this paper, we assume that S=$s_0s_1 \cdots s_ns_{n+1}$ and T=$t_0t_1 \cdots t_mt_{m+1}$, where s_0=s_{n+1}=t_0=t_{m+1}=$*$.

For a sequence S and a multi-set of elements X, let $S + X$ be the set of all possible resulting sequences after filling all the elements in X into S. Now, we put forward the problems we study in this paper formally.

Definition 2. *Two-sided Scaffold Filling Problem(TSSF-MNA problem).*
Input*: two scaffolds S and T over a gene set Σ and two multi-sets of elements X and Y, where $X = c(T) - c(S)$ and $Y = c(S) - c(T)$.*
Question*: Find $S' \in S + X$ and $T' \in T + Y$ such that $|a(S',T')|$ is maximized.*

The one-sided SF-MNA problem is a special instance of the TSSF-MNA problem where Y is empty, and is the problem we studied in this paper.The definition of it is described below.

Definition 3. *One-sided Scaffold Filling Problem(OSSF-MNA problem).*
Input*: a complete scaffold T and an incomplete scaffold S over a gene set Σ, a multi-set $X = c(T) - c(S) \neq \varnothing$, while $c(S) - c(T) = \varnothing$.*
Question*: Find $S' \in S + X$ such that $|a(S', T)|$ is maximized.*

Note that while the TSSF-MNA problem is more general and more difficult, the OSSF-MNA is more practical as a lot of genome analysis are based on some reference genome [13].

3 An Approximation Algorithm by Local Search

In this section, we show a local search algorithm for the one-sided scaffold filling problem, before that, we recall a key algorithm presented in [15] by Liu et al., which is actually a method of filling a scaffold, guaranteeing that on average each insertion of a single gene would generate at least one common adjacency. In our algorithm, after the local search step, we call that algorithm *Insert-Whole-String* in [15].

Theorem 1. *There exists a polynomial time algorithm to fill a scaffold, guaranteeing that the number of adjacencies increased is not smaller than the number of genes inserted.*

3.1 An Equivalent Goal

The goal of solving this problem is to insert the genes of X into scaffold $S$and obtain as many adjacencies as possible. No matter in what order the genes are inserted, they appears in groups in the final $S' \in S + X$, so we can consider that S' is obtained by inserting strings (composed of genes of X) into S.

Obviously, inserting a string of length one (i.e., a single gene) will generate at most two more adjacencies, and inserting a string of length m will generate at most m+1 more adjacencies. Therefore, there would be two types of inserted strings.

- good string: a string of k missing genes $x_1, x_2, \ldots, x_k$ are inserted in between $s_i s_{i+1}$ in scaffold S to obtain k+1 adjacencies (i.e., $s_i x_1, x_1 x_2, \ldots, x_{k-1} x_k, x_k s_{i+1}$), where $s_i s_{i+1}$ is a breakpoint.
  In this case, $x_1, x_2, \ldots, x_k$ is called a k-good string, $s_i s_{i+1}$ is called a *dock*, and we also say that $s_i s_{i+1}$ *docks* the corresponding k-good string $x_1 \ldots x_k$.
- bad string: a sequence of l missing genes $y_1, y_2, \ldots, y_l$ are inserted in between $s_j s_{j+1}$ in scaffold S to obtain l adjacencies (i.e., $s_j y_1$ or $y_l s_{j+1}, y_1 y_2, \ldots, y_{l-1} y_l$), where $s_j s_{j+1}$ is a breakpoint;
  or a sequence of l missing genes $y_1, y_2, \ldots, y_l$ are inserted in between $s_j s_{j+1}$ in scaffold S to obtain l+1 adjacencies (i.e., $s_j y_1, y_1 y_2, \ldots, y_{l-1} y_l, y_l s_{j+1}$), where $s_j s_{j+1}$ is an adjacency.

$$\begin{aligned}
&\textit{scaffold}\ \ T = \langle ...a\ x\ b\ ...a\ y\ z\ b...c\ y\ z\ d...c\ y\ d\ \rangle\\
&\textit{scaffold}\ \ S = \langle a\ b...c\ d...c\ z\ d...a....b...y...y...\rangle\\
&\textit{missing genes}\ \ X = \{x, y, z\}\\
&\textit{good strings}\ \ :\ \{x\ :\textit{docked by } ab,\ yz :\ \textit{docked by } ab, y :\ \textit{docked by } cz\}\\
&\textit{some bad strings}\ \ :\ \{x : \textit{inserted right after the second } a, y : \textit{inserted inbetween } zd\}
\end{aligned}$$

Fig. 2. An example for good strings and bad strings

The following figure shows some examples of good strings and bad strings.

From Theorem 1 and the definition of good strings, we can describe the optimal solution in the following formula. Let $N_0 = |a(S,T)|$ be the number of adjacencies between S and T, $|OPT|$ be the number of adjacencies in some optimal solution,and b_i be the number of i-good strings in this optimal solution. Assume the length of good strings is at most p. Then we have,

$$|OPT| = N_0 + |X| + \sum_{i=0}^{p} b_i$$

Since Theorem 1 guarantees at least $N_0 + |X|$ adjacencies are increased, to obtain a better solution, we focus on searching enough number of good strings. The following Lemma shows the bound for the optimal number of adjacencies,and how many good strings we should obtain to reach a approximation factor of 6/5.

Lemma 1. *Let $|APP|$ be the number of adjacencies obtained by an algorithm and b'_i be the number of i-good strings in this algorithm, then the approximation factor is 6/5, provided that $4b_1 + 3b_3 + 2b_3 + b_4 \leq 6b'_1 + 6b'_2 + 6b'_3 + 6b'_4$.*

Proof. Since $|OPT| = N_0 + |X| + \sum_{i=0}^{p} b_i$ and $|APP| = N_0 + |X| + \sum_{i=0}^{q} b'_i$, where q is the greatest length of good strings by the algorithm. Then we have,

$$\begin{aligned}
|OPT| &= N_0 + |X| + \sum_{i=1}^{q} b_i\\
&= N_0 + |X| + b_1 + b_2 + b_3 + b_4 + \sum_{i=5}^{q} b_i\\
&\leq N_0 + |X| + b_1 + b_2 + b_3 + b_4 + (|X| - b_1 - 2b_2 - 3b_3 - 4b_4)/5\\
&\leq N_0 + \frac{6}{5}(|X| + \frac{4}{6}b_1 + \frac{3}{6}b_2 + \frac{2}{6}b_3 + \frac{1}{6}b_4)\\
&\leq N_0 + \frac{6}{5}(|X| + b'_1 + b'_2 + b'_3 + b'_4)\\
&\leq \frac{6}{5}|APP|
\end{aligned}$$

□

From Lemma 1, we will obtain the approximation factor of 6/5, as long as the number of good strings in our algorithm is at least $4/6b_1 + 3/6b_2 + 2/6b_3 + 1/6b_4$, i.e., $4b_1 + 3b_3 + 2b_3 + b_4 \leq 6b'_1 + 6b'_2 + 6b'_3 + 6b'_4$.

3.2 Description of the Local Search Algorithm

From Lemma 1, the key purpose of our algorithm is to obtain enough number of good strings of length 1, 2, 3, and 4. Let b'_i be the number of i-good strings computed by our algorithm. There are three stages in our algorithm: the first is a greedy stage, which provides an initial solution for the local search step; then the algorithm improves the solution by a local search method, during which, the algorithm iteratively optimizes the function $D = b'_1 + b'_2/2 + b'_3/6$ by 1-substitution; finally, the algorithm inserts the remaining genes, guaranteeing that each insertion of a gene will generate at least one more adjacency. we show the algorithm as follows.

Algorithm 1. Scaffold Filling by Local Search

```
 1: for ($i$ from 1 to 4) do
 2:     Identify the breakpoints and adjacencies in $T$ and $S$, $S_{i-good} = \phi$
 3:     while ($X \neq \phi$, and there is an $i$-string $x_1 \cdots x_i$ composed of $i$ genes in $X$ is
        docked at $s_w s_{w+1}$, where $s_w s_{w+1} \in b_T(S,T)$ ) do
 4:         { insert the i-string $x_1, \cdots, x_i$ between $s_w$ and $s_{w+1}$ in $S$, and obtain $S'$.
 5:         $S = S'$, $S_{i-good} = S_{i-good} \cup \{x_1, \cdots, x_i\}$
 6:         $b_T(S,T) = b_T(S,T) - \{s_w s_{w+1}\}$, $X = X - \{x_1, \cdots, x_i\}$ }
 7:     end while
 8: end for
 9: Set $b'_i$ be the number of $i$-good strings currently.
10: $b'_i$ be the number of $i$-good strings currently.
11: $S_{good} = S_{1-good} \cup S_{2-good} \cup S_{3-good}$
12: for ($i$ from 1 to 4 ) do
13:     $b'_i = |S_{i-good}|$
14: end for
15: Set $D = b'_1 + b'_2/2 + b'_3/6$.
16: while there exists a good string $\alpha$, whose deletion will bring at least one other
    good string $\beta$, and increase $D$ do
17:     Delete $\alpha$ from $S_{good}$; add the new good strings $\alpha$ brings to $S_{good}$.
18: end while
19: Insert the remaining genes according to the Insert-Whole-String() method in [15].
```

4 Proof of the Approximation Factor

In our algorithm, we try to find good strings as many as possible. But a good string (say I_s) found by our algorithm may make other good strings in the optimal solution infeasible, we say I_s *destroys* them. The following lemma shows that the number of good strings that could be destroyed by a given good string is bounded.

Lemma 2. *An i-good string can destroy at most* $i + 1$ *good strings in some optimal solution.*

Proof. Assume that an i-good string I_s is inserted in between some breakpoint $s_j s_{j+1}$ in S. Then each of its genes, if was not occupied by I_s, can be part of a distinct good string respectively in some optimal solution. Also, there may exist another good string that could be inserted in between the breakpoint $s_j s_{j+1}$ in the optimal solution. Totally, at most i+1 good strings in the optimal solution could be destroyed by I_s. □

Corollary 1. *An i-good string in the optimal solution can be destroyed at most* $i + 1$ *times by good strings from other solutions.*

To prove the approximation factor, we only need to compare the solution of our algorithm with a fixed optimal solution. If a good string appears both in our solution and the optimal solution, it will not affect our analysis, so we only consider the distinct good strings between our solution and the optimal solution.

Let S'_i be the set of i-good strings found by our algorithm, and $|S'_i| = b'_i$, $i \in \{1, 2, 3, 4\}$. Let S_i be the set of i-good strings in the optimal solution, and $|S_i| = b_i$, $i \in \{1, 2, 3, 4\}$. To analyze the relationship between the number of good strings we found and good strings in the optimal solution, we construct an imaginary bipartite graph $G = (L, R, E)$. $L = \bigcup_{i=1}^{4} S'_i$, $R = \bigcup_{i=1}^{4} S_i$, and if a good string $\alpha \in L$ destroys a good string $\beta \in R$, there would be an edge between them in G. (See Fig-3.)

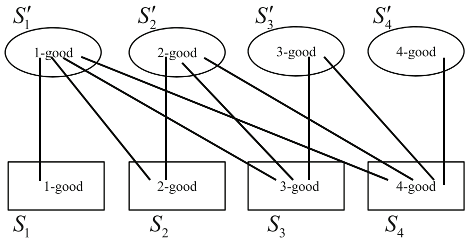


Fig. 3. Sketch of the imaginary graph

For a vertex $\alpha \in L \cup R$, let $d(\alpha)$ be the degree of α in G. We found that the graph G has the following properties.

Lemma 3. *In R, there is no isolated vertex.*

Proof. Since the *Scaffold Filling by Local Search* algorithm searches for good strings greedily first, any good string of length 1,2,3 and 4 in the optimal solution would other be chosen or be destroyed. If chose, it will not appear in R; If destroyed,it will appear in R, connecting to some good strings in L. The local search stage will not left a good string unchosen. □

Lemma 4. *For any vertex $c'_j \in S'_j \subseteq L$ (j=1,2,3), no vertex $c_i \in S_i$ with $i < j$ and $d(c_i) = 1$ can connect to c'_j, and at most one vertex $c_j \in S_j$ with $d(c_j) = 1$ can connect to c'_j.*

Proof. The local search stage aims at maximizing $D = b'_1 + b'_2/2 + b'_3/6$ by 1-substitution, so whenever there is a good string of length less than j whose degree is one, or more than one good strings of length j whose degree are one connecting to c'_j, a 1-substitution would be applied. □

Lemma 5. *For any vertex $c'_j \in S'_j \subseteq L$ ($j = 1, 2, 3$), at most $j + 1$ vertices in R, the degree of which is one and the length of which are greater than j, can connect to c'_j.*

Proof. From Lemma 4, at most one j-good strings with one degree in R can connect to a j-good string in L. The local search stage aims at maximizing $D = b'_1 + b'_2/2 + b'_3/6$ by 1-substitution, so if there is a j-good string in R, it would be impossible that a good string of length less than j whose degree is one in R connect to it. From Lemma 2, c'_j can destroy at most $j + 1$ good strings of the optimal solution. So the lemma follows. □

Lemma 6. *For any vertex $c_i \in S_i \subseteq R(i = 1, 2, 3, 4)$, c_i is connecting to at least one vertex $c'_j \in S'_j \subseteq L(j = 1, 2, 3, 4)$ with $j \leq i$.*

Proof. From the construction of the imaginary graph ,we know c_i is connecting to at least one vertex $c'_j \in S'_j$. Assume to the contrary that there exists a vertex $c_i \in S_i$, which connects to some vertices ($\in S'_j$) with $j \leq i$. Then c_i is destroyed by some good strings of length greater than i. Since the algorithm *Scaffold Filling by Local Search* search for good strings greedily first, then c_i would other be chosen or be destroyed by at least one chosen good strings of length i or less. Once it was chosen, since it would destroy at most $i + 1$ good strings, from the function $D = b'_1 + b'_2/2 + b'_3/6$, it will not be replaced by some good strings of length greater than i, so the exclusive possibility to replace it is replacing it by a good strings of length i or less and some other good strings. So no matter how many times it was chosen and replaced, it will keep connecting to at least one good string of length i or less. □

Let X_i be the number of i-good strings in the optimal solution, and Y_i be the number of other i-good strings, i.e., $b_i = X_i + Y_i$. Let K_{ij} be the number of j-good stings whose degree are one in the optimal solution connecting to i-good stings in our solution. from Lemma 6, we know that $1 \leq i \leq j \leq 4$. Let H_{ij} be the number of edges between i-good stings in our solution and j-good stings whose degree are not one in the optimal solution, where $j > i$.

Lemma 7. $4b_1 + 3b_2 + 2b_3 + b_4 \leq 6b'_1 + 6b'_2 + 5b'_3 + 5b'_4$.

Proof. From Lemma 4,5, we have,

$$X_1 \leq b'_1 - (K_{12} + K_{13} + K_{14})/2 \tag{1}$$

Since other 1-good strings has degree at least 2, and from Lemma 6, all the degrees are contributed by the 1-good string in our solution, and each 1-good sting can contribute at most 2 degree. Thus we have,

$$2Y_1 + X_1 \leq 2b'_1 - (K_{12} + K_{13} + K_{14} + H_{12} + H_{13} + H_{14}) \tag{2}$$

Similar argument holds for X_2, Y_2, X_3, Y_3, then we have,

$$X_2 - K_{12} \leq b'_2 - (K_{23} + K_{24})/3 \tag{3}$$
$$2Y_2 + X_2 - K_{12} - H_{12} \leq 3b'_2 - (K_{23} + K_{24} + H_{23} + H_{24}) \tag{4}$$
$$X_3 - K_{13} - K_{23} \leq b'_3 - K_{34}/4 \tag{5}$$
$$2Y_3 + X_3 - K_{13} - K_{23} - H_{13} - H_{23} \leq 4b'_3 - (K_{34} + H_{34}) \tag{6}$$

Use $\frac{(1)+(2)}{2}$, $\frac{(3)+(4)}{2}$,$\frac{(5)+(6)}{2}$, we have,

$$b_1 \leq 3b'_1/2 - 3(K_{12} + K_{13} + K_{14})/4 - (H_{12} + H_{13} + H_{14})/2 \tag{7}$$
$$b_2 \leq 2b'_2 + K_{12} - 2(K_{23} + K_{24})/3 - (H_{23} + H_{24})/2 + H_{12}/2 \tag{8}$$
$$b_3 \leq 5b'_2/2 + K_{13} + K_{23} - 5K_{34}/8 - H_{34}/2 + (H_{13} + H_{23})/2 \tag{9}$$

As for the 4-good strings, since each 4-good string in our solution can connect to at most 5 4-good strings of the optimal solution, thus we have

$$b_4 \leq 5b'_4 + (K_{14} + K_{24} + K_{34}) + (H_{14} + H_{24} + H_{34})/2 \tag{10}$$

Use $4 \times (7) + 3 \times (8) + 2 \times (9) + (10)$, we will have,

$$4b_1 + 3b_2 + 2b_3 + b_4 \leq 6b'_1 + 6b'_2 + 5b'_3 + 5b'_4 \tag{11}$$

Directly form Lemma 1 and Lemma 7, we have,

Theorem 2. *The algorithm* Scaffold Filling by Local Search *guarantees an approximation factor of 6/5.*

5 The Scaffold Filling Problem is MAX-SNP-complete

In reference [12], Jiang et al., has proved that the Scaffold Filling Problem is NP-hard. In this section, we show that this problem is MAX-SNP-complete, which makes a polynomial time approximation schemes nearly impossible. To complete this proof, we need to construct an L-reduction from a known MAX-SNP-complete problem (the Maximum 3-Dimensional Matching problem in this paper), to the One-sided Scaffold Filling Problem. Though the reduction method here is similar to that of [12], we conduct a stricter analysis, and produce a stronger negative result.

Definition 4. *Maximum 3-Dimensional Matching* [17].
Instance:*Set $T \subseteq X \times Y \times Z$, where X, Y, and Z are disjoint.*
Solution: *A matching for T, i.e., a subset $M \subseteq T$ such that no elements in M agree in any coordinate.*
Measure: *Cardinality of the matching, i.e., $|M|$.*

The variation in which the number of occurrences of any element in X, Y or Z is bounded by 3, is called the 3-bounded Maximum 3-Dimensional Matching problem, abbreviated as $3DM - 3$. The 3-bounded Maximum 3-Dimensional Matching problem is Shown to be MAX SNP-complete in [18].

We consider a more special variation of the 3-bounded Maximum 3-Dimensional Matching problem, in which each pair of triples in T can share at most one common element, we call this problem 3-bounded 1-common Maximum 3-Dimensional Matching, abbreviated as $(3DM - 3)^1$.

Theorem 3. *$(3DM - 3)^1$ is MAX SNP-complete.*

Proof. It suffices to show that the 3-bounded Maximum 3-Dimensional Matching problem can be L-reduced to $(3DM-3)^1$. Construct an instance of $(3DM-3)^1$ from an instance of 3-bounded Maximum 3-Dimensional Matching as follows. Given an instance I of 3-bounded Maximum 3-Dimensional Matching with $|X| = |Y| = |Z| = q$ and m triples $(3q \geq m)$, for each triple $T_u = (x_i, y_j, z_k)$, construct the following 5 triples: $T'_u = \{(x_i, y_{u,1}, z_{u,2}), (x_{u,1}, y_{u,2}, z_k), (x_{u,2}, y_j, z_{u,1}), (x_{u,1}, y_{u,1}, z_{u,1}), (x_{u,2}, y_{u,2}, z_{u,2})\}$. Then, in the $(3DM-3)^1$ instance I', we have $|X'| = |Y'| = |Z'| = q + 2m$ and $5m$ triples. It remains to show that this reduction satisfies the two inequalities of an L-reduction.

Firstly, if T_u was chosen as a triple of $OPT(I)$, we can choose the former three triples of T'_u into $OPT(I')$, note that this is the unique way to choose three disjoint triples from T'_u; and if not, we can choose the latter two triples of T'_u into $OPT(I')$. So, $|OPT(I')| = |OPT(I)| + 2m$. Since each triple of I can share common elements with at most 6 other triples, then $OPT(I) \geq m/7$. Thus, $|OPT(I')| \leq |OPT(I)| + 14|OPT(I)| = 15|OPT(I)|$.

For a solution $c(I')$ of I', we choose T_u into a corresponding solution $c(I)$ of I if and only if $c(I')$ contains three triples of T'_u. If $|c(I')| \leq 2m$, we have $|OPT(I)| - |c(I)| \leq |OPT(I)| - (|OPT(I')| - 2m) \leq |OPT(I')| - |c(I')|$. On the other hand, if $|c(I')| > 2m$, since there are only m groups of T'_us, there should be at least $|c(I')| - 2m$ groups of T'_us, from which we choose 3 triples, so $|c(I)| \geq |c(I')| - 2m$. Therefore, $|OPT(I)| - |c(I)| \leq (|OPT(I')| - 2m) - (|c(I')| - 2m) = |OPT(I')| - |c(I')|$. This completes the proof. □

Now, we L-reduce $(3DM - 3)^1$ to the One-sided Scaffold Filling problem.

Theorem 4. *The One-sided Scaffold Filling problem is MAX SNP-complete.*

Proof. Suppose that we are given a $(3DM - 3)^1$ instance I with $|X| = |Y| = |Z| = q$ and m triples $(3q \geq m)$. Assume that each element appears at least once in the triples,i.e., $q \leq m$. We construct an instance J of the One-sided Scaffold Filling problem as follows. Let the elements to be inserted are $X \cup Y \cup Z$. For each triple $T_u = (x_i, y_j, z_k)$, we construct two substrings $T'_u = g_u f_u x_i y_j z_k f'_u g'_u$ and $M'_u = g'_u f_u f'_u g_u$. For an element of X, say x_i, appearing $l \leq 3$ times in the triples, we denote each appearance by x_i^1, x_i^2,..., x_i^l respectively. Similar denotations are used for the elements of Y and Z. There are also $3m - 3q + 4$ new elements as separators. We denote the concatenation of strings by Π.

$$S = \Pi_{i=1}^{q}(x_i^1 r_x^1 x_i^2 r_x^2 y_i^1 r_y^1 y_i^2 r_y^2 z_i^1 r_z^1 z_i^2 r_z^2) R_1 R_2 \Pi_{i=1}^{\lceil m/2 \rceil}(M'_{2i-1}) R_3 R_4 \Pi_{i=1}^{\lfloor m/2 \rfloor}(M'_{2i})$$

$$T = \Pi_{i=1}^{q}(r_x^1 r_x^2 r_y^1 r_y^2 r_z^1 r_z^2) R_3 R_2 R_4 R_1 \Pi_{i=1}^{m}(T'_i)$$

Note that $x_i^1 = x_i^2 = x_i$, and if x_i appears only once, then $x_i^1, r_x^1, x_i^2, r_x^2$ are all empty strings; if x_i appears twice, then then x_i^2, r_x^2 are all empty strings. We can observe that there is no common adjacency between S and T, and S contains $10m - 6q + 4$ elements while T has $10m - 3q + 4$ elements. Next, we show that this reduction fulfills the two inequalities of an L-reduction.

Firstly, if T_u was chosen as a triple of $OPT(I)$, we can insert x_i, y_j, z_k into M'_u, which brings 4 common adjacencies. For these elements not covered by $OPT(I)$, Theorem 1 ensures that each insertion of an element will bring at least one common adjacencies. Since the common adjacencies of $OPT(J)$ must be the following four forms: $f'_u x_i, x_i y_j, y_j z_k, z_k f'_u$, since each pair of triples shares at most one element, then $x_i y_j, y_j z_k$ can be obtained at most once, the unique way to obtain such four common adjacencies, is to insert the three elements of T_u into M'_u.

Then, we prove that there is no good strings of length greater than 3 in $OPT(J)$. Observe that the unique to obtain one more common adjacency to to insert some elements into a M'_u. Assume to the contrary that $s_1 s_2 \cdots s_r$ ($r \geq 4$) be a r-good string by inserting it into M'_v, then $s_1 \in X \cap T_v, s_r \in Z \cap T_v$, which means that $s_2 \in Y \cap T_v$ and $s_{r-1} \in Y \cap T_v$, a contradiction. Therefore, $|OPT(J)| = |OPT(I)| + 3q \leq |OPT(I)| + 3m$. Since each triple of I can share common elements with at most 6 other triples, then $OPT(I) \geq m/7$. Thus, $|OPT(J)| \leq |OPT(I)| + 21|OPT(I)| = 22|OPT(I)|$.

For a solution $c(J)$ of J, we choose T_u into a corresponding solution $c(I)$ of I if and only if $c(J)$ contains four common adjacencies $f'_u x_i, x_i y_j, y_j z_k, z_k f'_u$. If $|c(J)| \leq 3q$, we have $|OPT(I)| - |c(I)| \leq |OPT(I)| = |OPT(J)| - 3q \leq |OPT(J)| - |c(J)|$. On the other hand, if $|c(J)| > 3q$, there should be at least $|c(J)| - 3q$ M'_us, from which we obtain 4 common adjacencies by insertions of the three elements of a triple, so $|c(I)| \geq |c(J)| - 3q$. Therefore, $|OPT(I)| - |c(I)| \leq (|OPT(J)| - 3q) - (|c(J)| - 3q) = |OPT(J)| - |c(J)|$. This completes the proof. □

6 Conclusion

In this paper, we investigate the One-sided Scaffold Filling problem, we improve the approximation factor from 1.25 to 1.2 by a not-oblivious local search method.

Our algorithm is quite implementable since we only use 1-substitution. A meaningful future work is to analyze the performance by c-substitution for any constant c, We believe that, with c increasing, the approximation ratio could be improved slightly. but this would stop for some c.

We define a term called "good string", which help us describe the optimal solution very well, upon this, by intuition, it seems that this problem could admit a $PTAS$. But our MAX-SNP-complete proof prevents it.

Acknowledgments. This research is partially supported NSF of China under grant 61202014 and 61472222, by NSF of Shandong Provence (China) under grant ZR2012FQ008 and ZR2012FZ002, by China Postdoctoral Science Foundation funded project under grant 2011M501133 and 2012T50614.

References

1. Angibaud, S., Fertin, G., Rusu, I., Thevenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. J. Graph Algorithms and Applications **13**(1), 19–53 (2009)
2. Blin, G., Fertin, G., Sikora, F., Vialette, S.: The EXEMPLAR BREAKPOINT DISTANCE for non-trivial genomes cannot be approximated. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 357–368. Springer, Heidelberg (2009)
3. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. In: Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA 2002), pp. 667–676 (2002)
4. Chen, Z., Fowler, R., Fu, B., Zhu, B.: On the inapproximability of the exemplar conserved interval distance problem of genomes. J. Combinatorial Optimization **15**(2), 201–221 (2008)
5. Khanna, S., Motwani, R., Madhu, S., Umesh, V.: On syntactic versus computational views of approximability. SIAM Journal on Computing **28**(1), 164–191 (1998)
6. Chen, Z., Fu, B., Xu, J., Yang, B., Zhao, Z., Zhu, B.: Non-breaking similarity of genomes with gene repetitions. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 119–130. Springer, Heidelberg (2007)
7. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 291–302. Springer, Heidelberg (2006)
8. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partitioning problem: hardness and approximations. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 484–495. Springer, Heidelberg (2004). also in: The Electronic Journal of Combinatorics **12** (2005), paper R50
9. Jiang, M.: The zero exemplar distance problem. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 74–82. Springer, Heidelberg (2010)
10. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint distance. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 83–92. Springer, Heidelberg (2010)
11. Jiang, H., Zhong, F., Zhu, B.: Filling scaffolds with gene repetitions: maximizing the number of adjacencies. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 55–64. Springer, Heidelberg (2011)

12. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint and related distances. IEEE/ACM Trans. Bioinformatics and Comput. Biology **9**(4), 1220–1229 (2012)
13. Muñoz, A., Zheng, C., Zhu, Q., Albert, V., Rounsley, S., Sankoff, D.: Scaffold filling, contig fusion and gene order comparison. BMC Bioinformatics **11**, 304 (2010)
14. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics **15**(11), 909–917 (1999)
15. Liu, N., Jiang, H., Zhu, D., Zhu, B.: An Improved Approximation Algorithm for Scaffold Filling to Maximize the Common Adjacencies. IEEE/ACM Trans. Comput. Biology Bioinform. **10**(4), 905–913 (2013)
16. Liu, N., Zhu, D.: The algorithm for the two-sided scaffold filling problem. In: Chan, T.-H.H., Lau, L.C., Trevisan, L. (eds.) TAMC 2013. LNCS, vol. 7876, pp. 236–247. Springer, Heidelberg (2013)
17. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
18. Kann, V.: Maximum bounded 3-dimensional matching is MAX SNP-complete. Inform. Process. Lett. **37**, 27–35 (1991)

Packing Cubes into a Cube in (*D*>3)-Dimensions

Yiping Lu[1(✉)], Danny Z. Chen[2], and Jianzhong Cha[1]

[1] School of Mechanical, Electronic and Control Engineering, Beijing Jiaotong University, Beijing 100044, China
{yplu,jzcha}@bjtu.edu.cn
[2] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
dchen@cse.nd.edu

Abstract. The problem of determining whether a set of cubes can be orthogonally packed into a cube has been studied in 2-diminson, 3-dimension, and (*d*>3)-dimensions. Open questions were asked on whether this problem is NP-complete in three articles in 1989, 2005, and 2009, respectively. In 1990, the problem of packing squares into a square was shown to be NP-complete by Leung et al. Recently, the problem of packing cubes into a cube in 3-D was shown to be NP-complete by Lu et al. In this paper, we show that the problem in (*d*>3)-dimensions is NP-complete in the strong sense, thus settling the related open question posed by previous researchers.

Keywords: Packing problems · Cube packing · NP-completeness

1 Introduction

2-D square packing and 3-D and (d>3)-D cube packing have been intensely studied (e.g., see [1, 2, 3, 4, 5, 7, 8, 9, 10, 11]). All known solutions to the cube packing problem are mainly based on approximation or heuristic methods. Determining the complexity of the cube packing problem is needed for developing effective algorithms for it. The common belief is that for all (d ≥2)-D, the cube packing problem is NP-complete. In fact, NP-complete proofs have been given for the problem in 2-D and 3-D. However, it should be pointed out that, unlike many other geometric problems, the known NP-complete proofs of the cube packing problem in 2-D and 3-D do not immediately translate into a satisfactory NP-complete proof for the version in (d>3)-D. Thus, proving the NP-completeness of the problem in (d>3)-D is still needed.

The NP-completeness of the problem of 'packing multiple rectangles into a rectangle' follows trivially because one of its special cases can be degenerated to the well-known knapsack problem [6].

In 1989, Li and Cheng [10] showed that both the problem of 'packing multiple squares into a rectangle' and the problem of 'packing multiple rectangles into a square' are NP-complete, but posted a question on the NP-completeness of the problem of 'packing squares into a square'.

Li and Cheng's question [10] was answered by Leung et al. [9] in 1990, who proved that the problem of 'packing squares into a square' is indeed NP-complete;

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 264–276, 2015.
DOI: 10.1007/978-3-319-21398-9_21

but, the NP-completeness of the problem version in (d>2)-D (i.e., the problem of packing multiple cubes into a cube in (d>2)-dimension) was still not settled.

In 2005 and 2009, Epstein and van Stee [5] and Harren [7], respectively, posed the determination of NP-completeness of this cube packing problem in (d>2)-D as an open problem.

The 3-D version of the problem was proved to be NP-complete recently [11].

In this paper, we show that the (d>3)-D version of the problem is strongly NP-complete, thus completely settling the open problem posed by Epstein and van Stee [5] and Harren [7]. Our proof is based on a unified construction scheme for the problem in all (d≥2)-dimensions, which actually simplifies the proofs for the 2-D and 3-D cases in [9, 11].

1.1 The Problem of Packing Multiple Cubes into a Cube

The problem of packing multiple cubes into a cube in (d>1)-D is that of orthogonally packing a set of small cubic items into a big cubic volume without any interior overlapping among the small cubic items. The problem is formally defined as follows.

THE PROBLEM OF PACKING CUBES INTO A CUBE

INSTANCE: A big cube C and a set of small cubes, $L=\{c_1,c_2,...,c_n\}$, whose sizes are all positive integers.

QUESTION: Can all cubes of L be packed into C orthogonally without any intersection of the interior points between any two cubes of L?

1.2 The 3-Partition Problem

In this paper, we reduce the 3-partition problem to the problem of packing cubes into a cube. The 3-partition problem is known to be NP-complete in the strong sense [6], which is formally defined as follows.

THE 3-PARTITION PROBLEM

INSTANCE: A set of $3m$ ($m>1$) positive integers, $S=\{a_1,a_2,...,a_{3m}\}$, and a bound B, such that for every i, $1\le i\le 3m$, $B/4<a_i<B/2$, and $\sum_{i=1}^{3m}a_i=mB$.

QUESTION: Can S be partitioned into m disjoint subsets $S_1,S_2,...,S_m$, such that for each j with $1\le j\le m$, the sum of all elements in S_j is exactly B? Note that the constraints of the problem require that each subset S_j should have exactly 3 elements of S.

Note that even if we assume that m is a multiple of 3, the 3-partition problem is still NP-complete in the strong sense [11]. In the rest of this paper, we assume that m is a multiple of 3, and write $m=3m'$ when it is needed, where m' is a positive integer.

The rest of this paper is organized as follows. In Section 2, we review the related results. In Sections 3, we give new NP-completeness proofs of the cube packing problem in 2-D. This proof is much simplified in comparison with those in [9, 11], and thus it is much easier to be extended to the higher dimensional version. In Section 4, we show that the cube packing problem in $(d > 2)$-D is NP-complete in the strong sense. Section 5 concludes the paper.

2 Review of Related Work

2.1 Li and Cheng's Work on Packing Squares into a Rectangle

In 1989, Li and Cheng [10] proved that the problem of packing squares into a rectangle is NP-complete. Their proof is based on a reduction from the 3-partition problem. Given an instance I of the 3-partition problem, they constructed an instance I_p of packing squares into a rectangle in the following way (see Fig. 1):

$$C = (3A+B, m(A+B/2)), \quad L = \{(A+a_1), (A+a_2), \ldots, (A+a_{3m})\},$$

where $A > (m-1)B/4$, C is a rectangle of size $(3A+B) \times m(A+B/2)$, and L is a list of $3m$ squares of sizes $A+a_1$, $A+a_2$, …, $A+a_{3m}$, respectively.

If the instance I has a partition, let $S_i = \{a_{i1}, a_{i2}, a_{i3}\}$, then it is clear that the instance I_p has a packing as shown in Fig. 1.

If the instance I_p has a packing, then at most m squares can be packed into C's vertical dimension of length $m(A+B/2)$. This is because if $m+1$ squares are packed into C's vertical dimension, then the total height of these squares will be bigger than $(m+1)(A+B/4) = m(A+B/2) + (A-(m-1)B/4) > m(A+B/2)$. For similar reason, at most 3 squares can be packed into C's horizontal dimension. Since the total number of squares is $3 \cdot m$ and no overlap among the packed squares is allowed, the only packing that is possible will have m levels of squares with 3 squares per level. Hence, we can obtain a partition that divides the $3m$ squares into m groups $G_i\,(i = 1, \ldots, m)$, where each group contains 3 squares.

Let $G_i = \{A+a_{i1}, A+a_{i2}, A+a_{i3}\}$. Then it must be true that

$$A+a_{i1}+A+a_{i2}+A+a_{i3} \le 3A+B$$

i.e.,

$$a_{i1}+a_{i2}+a_{i3} \le B.$$

Since $\sum_{i=1}^{m}(a_{i1}+a_{i2}+a_{i3}) = mB$, it has to be true that $a_{i1}+a_{i2}+a_{i3} = B$ for every i = 1, 2, …, m, which implies that the 3-partition problem instance I has a partition.

2.2 Leung et al.'s Work on Packing Squares into a Square

In 1990, Leung et al. [9] showed that packing multiple squares into a square is NP-complete. Their proof is also based on a reduction from the 3-partition problem.

Leung et al.'s construction is shown in Fig. 2. In the big square C, multiple small squares are packed. The area α in C is a rectangular area that is similar to Li and Cheng's construction [10] shown in Fig. 1 (the only difference is that the former is rotated by 90 degrees). To the left of the area α, some bigger squares are packed in the area β A very big square is packed into the square area γ that is below the area $\alpha \cup \beta$. Finally, in the area ε that is to the left of the area $\alpha \cup \beta \cup \gamma$ and in the area δ that is on top of the area $\alpha \cup \beta \cup \gamma \cup \varepsilon$, other squares are packed as tightly as possible.

Leung et al. showed that if there is a packing for the constructed instance, then the packing in the area α is just like the one constructed by Li and Cheng, and thus the instance of the 3-partition problem has a partition.

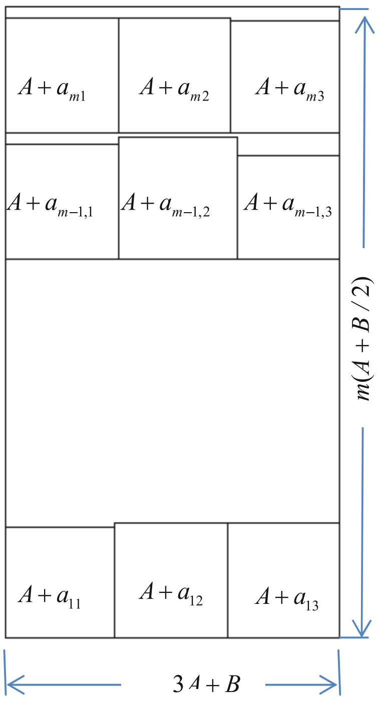


Fig. 1. Packing squares into a rectangle [10]

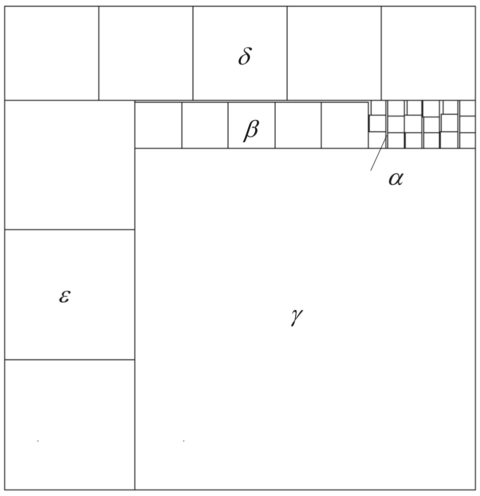


Fig. 2 Packing squares into a square [9]

2.3 Lu et al.'s Work on Packing Cubes into a Cube in 3-D

Recently, Lu et al. [11] showed that packing multiple cubes into a cube in 3-D is NP-complete. The proof of Lu et al. can be viewed as a 3-D extension of the proof by Leung et al. [9]. In their construction, Lu et al. [11] made some simplification of the proof in [9], and also used one big cubic item in the constructed packing instance, corresponding to the big square γ in Leung et al.'s construction (see Fig. 2).

In the remaining of this paper, we further simplify Leung et al.'s construction [9], in order to make it into the basis of the construction for our proof that can be easily extended to (d≥3)-dimensions. In this simplified construction, unlike the ones in [9, 11], we no longer use the "big" cubic item in the packing instance.

3 Packing Cubes into a Cube in 2-D

We first simplify Leung et al.'s construction for the 2-D case [9]. Given an instance I of the 3-partition problem, $B, S = \{a_1, a_2, ..., a_{3m}\}$, where $m = 3m'$ is a multiple of 3 and $m \geq 8$, we construct an instance I' of the square packing problem in 2-D, as follows.

Let $p = \left\lceil \sqrt{m/2} \right\rceil$, $A = 3p(p+1)B + 1$, and $l = 3A + B$. Note that because $m \geq 8$, we have $p \geq 2$.

The big square C (the packing space) is of size $p(p+1)l \times p(p+1)l$.

There are 4 types of square items to be packed into C, as follows (see Fig. 3).

- ✧ $3m$ α-squares whose sizes are $A + a_i, i = 1, 2, ..., 3m$, respectively. Because of their sizes, if I has a partition, then the α-squares can be all packed into the rectangular area α whose size is $l \times m(A + B/2)$. Note that the packing of the α-squares into area α is the same as that constructed by Li and Cheng [10] shown in Fig. 1.
- ✧ $p(p+1) - m'$ β-squares whose sizes are all of $l_\beta = l - m'(B/2)/(p(p+1) - m')$. Because of their size, the β-squares can clearly be packed into the rectangular area β whose size is $l \times (p(p+1)l - m(A + B/2))$. Note that the size of a β-square is larger than $3A$, because
$$l_\beta = l - m'(B/2)/(p(p+1) - m') = 3A + B - m'(B/2)/(p(p+1) - m')$$
$$> 3A + B - m'(B/2)/(m/2 - m') = 3A + B - m'(B/2)/(3m'/2 - m') = 3A.$$
- ✧ $p+1$ γ-squares whose sizes are all of pl. All γ-squares can be packed into the rectangular area γ whose size is $pl \times p(p+1)l$, as shown in Fig. 3.
- ✧ $p(p-1)$ δ-squares whose sizes are all $(p+1)l$. All δ-squares can be packed into the rectangular area δ whose size is $(p-1)(p+1)l \times p(p+1)l$, as shown in Fig. 3.

Lemma 1. The packing instance I' can be constructed from I in polynomial time of m. If I has a partition, then I' has a packing.

Lemma 2. In the instance I', if all α-squares and β-squares can be packed into the rectangular area $\alpha \cup \beta$ (or any rectangular area of size $l \times p(p+1)l$ or $p(p+1)l \times l$), then I has a partition.

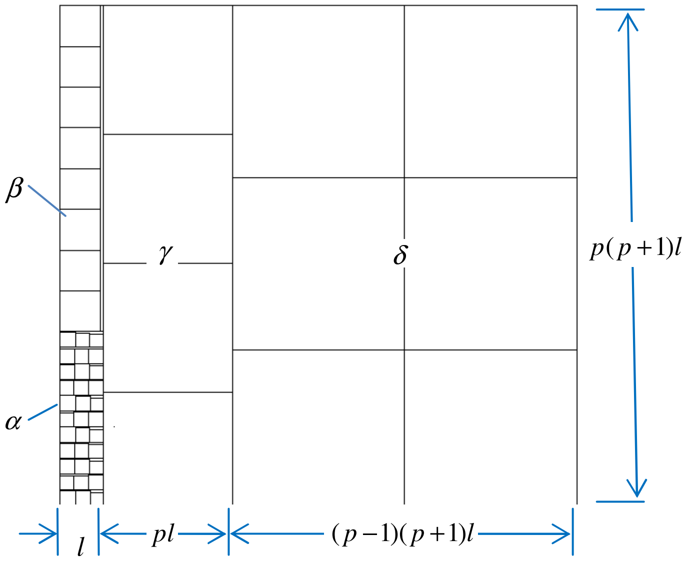


Fig. 3. Packing squares into a square

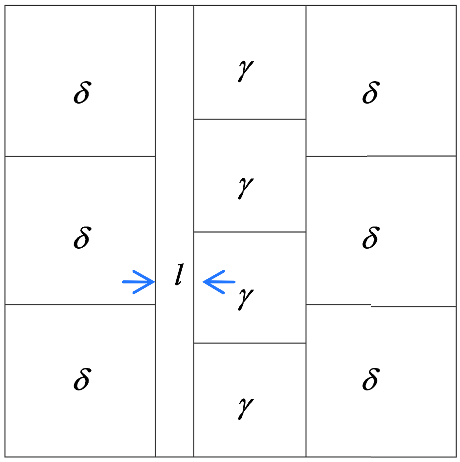
Proof: Without loss of generality, we assume that the packing area is $\alpha \cup \beta$ as shown in Fig. 3. To simplify the discussion, we imagine a situation in which every β-square is divided into 9 identical squares of size $l_\beta / 3$ each. This will create $9p(p+1) - 9m' = 9p(p+1) - 3m$ such squares whose (same) sizes are all larger than A since $l_\beta > 3A$. Because each of the $3m$ α-squares is of a size larger than A, we now have totally $9p(p+1)$ squares whose sizes are all larger than A, and they will be packed into the area $\alpha \cup \beta$. We call all these squares the $\underline{A}$*-squares*. Note that if all α-squares and β-squares can be packed into the area $\alpha \cup \beta$, then clearly all the $9p(p+1)$ $\underline{A}$-squares can be packed into the area $\alpha \cup \beta$ as well.

Note that no more than $3p(p+1)$ $\underline{A}$-squares can be packed side by side along a vertical line inside $\alpha \cup \beta$. This is because if $3p(p+1)+1$ $\underline{A}$-squares are packed in this way, then the packing space needed in the vertical direction will be larger than the size of the area $\alpha \cup \beta$ in the vertical dimension, due to

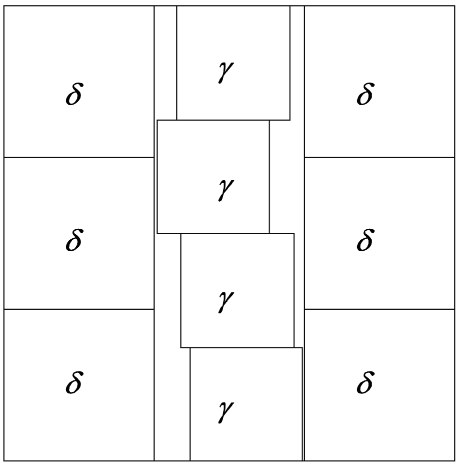
$$(3p(p+1)+1)A = 3p(p+1)A + A > 3p(p+1)A + 3p(p+1)B > p(p+1)l$$

Also, no more than 3 $\underline{A}$-squares can be packed side by side along a horizontal line inside $\alpha \cup \beta$ since $4A > 3A + B = l$. Thus, the number of $\underline{A}$-squares that are packed into the area $\alpha \cup \beta$ cannot be bigger than $3 \cdot 3p(p+1)$ which is exactly the total number of $\underline{A}$-squares. Hence, if a packing of $\alpha \cup \beta$ is achievable, then the $\underline{A}$-squares must be put approximately at the cell positions of a (roughly) uniform grid of size $3 \times 3p(p+1)$, with $3p(p+1)$ layers along the vertical direction and 3 cells along the horizontal direction. Subject to the space constraint in the horizontal direction, this packing means that the $\underline{A}$-squares are partitioned into $3p(p+1)$ sets each of which contains 3 $\underline{A}$-squares whose size sum is no bigger than l.

The $\underline{A}$-squares that are divided from one β-square should actually be packed together, meaning that all $\underline{A}$-squares divided from the β-squares are partitioned among themselves. Thus, the α-squares are partitioned into m sets each of which contains 3 α-squares whose size sum is no bigger than l. By referring to the discussions in Section 2.1, the instance I has a partition. □



(a) A case with $w = pl$



(b) A case with $w \in [pl, (p+1)l]$

Fig. 4. Examples of packing the γ-squares

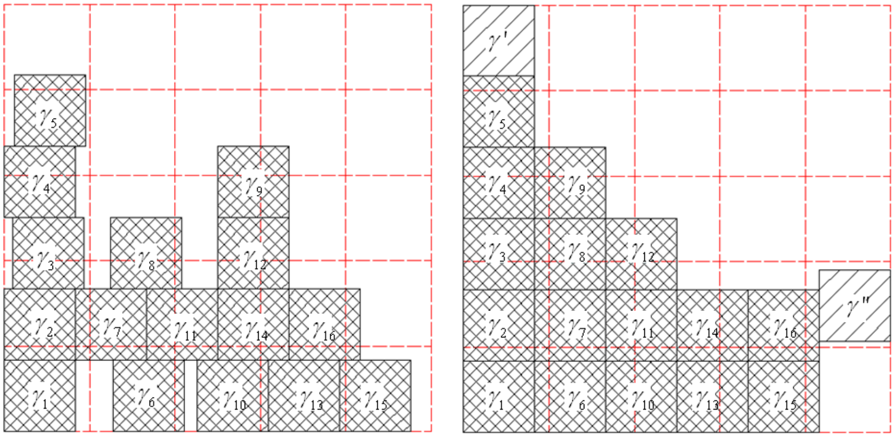
Lemma 3. In any achievable packing of the instance I', all γ-squares must be packed into a rectangular area of size $w \times p(p+1)l$ or $p(p+1)l \times w$, where $w \in [pl, (p+1)l]$ (see Fig. 4).

Proof: In this proof, we consider only the γ-squares and the δ-squares. In this situation, if a packing of I' is achievable, then clearly all $p+1$ γ-squares and all $p(p-1)$ δ-squares can be packed into C without any overlapping in their interior.

Note that the space of C can hold at most p^2 δ-squares. We call a packing that packs p^2 δ-squares into C a δ-packing. Suppose we first construct a δ-packing (a δ-packing is certainly achievable). Then at most p δ-squares can be removed from C to make room for packing the $p+1$ γ-squares (otherwise, the number of δ-squares packed into C will be smaller than $p(p-1)$).

Assume C is first packed tightly using p^2 δ-squares (i.e., a δ-packing), and there are still γ-squares to be packed into C in a certain manner. We now analyze how many δ-squares must be removed from C.

As shown in Fig. 5(a), suppose γ-squares $\gamma_1, \gamma_2, \ldots$ are to be packed into C which is already tightly packed by p^2 δ-squares. We want to determine the lower bound N_δ of how many δ-squares must be removed from C to avoid any interior overlap. We can shift each γ-square that is packed into C to the left (and to the bottom) until it touches the boundary of C or another γ-square without affecting the value of N_δ. Note that if a γ-square is shifted to the left (bottom), it will not cause more δ-squares to be removed and it will leave more space for the δ-square(s) to its right (top) such that they are less likely to be removed. For example, the N_δ value of Fig. 5(a) is no bigger than the N_δ value of Fig. 5(b). To determine N_δ, we can enumerate γ-squares row by row from left to right.



(a) γ-squares are packed in a δ-packing. (b) γ-squares are shifted to low-left.

Fig. 5. Packing γ-squares into a δ-packing

More specifically, we focus on each row of γ-squares and see how many δ-squares should be removed. Let m_j be the number of γ-squares in the jth row, and n_j be the lower bound of the number of δ-squares that must be removed due to the packing of the j^{th} row of γ-squares. Then, $m_j pl \le n_j(p+1)l$ must hold (otherwise, there will not be enough space for m_j γ-squares in the j^{th} row). When $j=1$, we have

$$n_1 = \begin{cases} m_1 & \text{, if } m_1 < p+1 \\ m_1 - 1 & \text{, if } m_1 = p+1 \end{cases}. \tag{1}$$

Note that because of the edge lengths of γ-squares and δ-squares, for any $j < p+1$, the j^{th} row of γ-squares overlaps with the j^{th} row of δ-squares. Thus Equ. (1) also holds for all $j < p+1$. But, for $j = p+1$, the j^{th} row of γ-squares can use the space provided by δ-squares in the $(j-1)^{\text{th}}$ row; thus no more δ-square is needed to be removed due to the j^{th} row of γ-squares.

Thus, we have

$$n_j = \begin{cases} m_j & \text{, if } m_j < p+1 \text{ and } j < p+1 \\ m_j - 1 & \text{, if } m_j = p+1 \text{ and } j < p+1 \\ 0 & \text{, if } j = p+1 \end{cases}. \tag{2}$$

We can explain Equ. (2) in the following way: For packing one γ-square into a δ-packing, we generally need to remove one δ-square; but for every fully packed row or column, one δ-square can be saved. For example, in Fig. 5(b), if there is a γ'-square at the top of the first column, then it will not increase N_δ; neither the γ''-square located at the $(p+1)^{\text{th}}$ column will increase N_δ.

Now returning to our 2-D square packing problem, there are $p^2 - p$ δ-squares and $p+1$ γ-squares to be packed into C. This is equivalent to that we remove from C at most p δ-squares from a δ-packing for the packing of the $p+1$ γ-squares into C (in other words, we can save one δ-square). Note that this is possible only if we pack all these γ-squares into one row or one column, i.e., all $p+1$ γ-squares must be packed into a rectangular area of size $w \times p(p+1)l$ or $p(p+1)l \times w$, where $w \in [pl, (p+1)l]$. □

Lemma 4. If there is a packing for the instance I', then all α-squares and β-squares can be packed into the rectangular area $\alpha \cup \beta$ (or any area of size $l \times p(p+1)l$ or $p(p+1)l \times l$).

Theorem 1. The problem of packing multiple squares into a square is NP-complete in the strong sense.

Proof: The problem of packing multiple squares into a square is in NP because a non-deterministic Turing machine can guess a solution and check whether the squares overlap in polynomial time of the square numbers, which is a polynomial of m. The 3-partition problem is NP-complete in the strong sense. Based on Lemmas 1, 2, 3, and 4, the theorem follows. □

4 Packing Cubes into a Cube in $(d > 2)$-D

4.1 Notation

In $(d > 2)$-D, we use $(r_1, r_2, ..., r_d)$ to represent a boxy region (or boxy item) whose size in the i th dimension is r_i ($i = 1, 2, ..., d$). When the location of the region is significant, we use $(e_1, e_2, ..., e_d)(r_1, r_2, ..., r_d)$ to represent a boxy region defined in the i th dimension by the open interval $(e_i, e_i + r_i)$. We call $(e_1, e_2, ..., e_d)$ the origin of the region. We also use $r_1 \times r_2 \times ... \times r_d$ to represent the boxy region $(r_1, r_2, ..., r_d)$ when we focus only on its size.

For a point $X = \{x_1, x_2, ..., x_d\}$, we say X is inside a region $(e_1, e_2, ..., e_d)$ $(r_1, r_2, ..., r_d)$ if $x_i \in (e_i, e_i + r_i)$ for all $i = 1, 2, ..., d$.

A region $(e_1, e_2, ..., e_d)(r_1, r_2, ..., r_d)$ is defined as a set of (infinitely many) points in ($d > 3$)-D $\{X = \{x_1, x_2, ..., x_d\} \mid \forall x_i \in (e_i, e_i + r_i)\}$. We say that two regions overlap if and only if they share some common inside points. Region R_1 is inside region R_2 if and only if all inside points of R_1 are also inside points of R_2.

We say that the union of two boxy regions is also a boxy region if and only if they have different sizes in no more than one dimension. For example, the union of two boxy regions

$\varepsilon_1 = (0, 0, ..., 0)(r_1, ..., r_{i-1}, r_i', r_{i+1}, ..., r_d)$ and

$\varepsilon_2 = (0, ..., 0, r_i', 0, ..., 0)(r_1, ..., r_{i-1}, r_i'', r_{i+1}, ..., r_d)$

is the boxy region $\varepsilon_1 \cup \varepsilon_2 = (0, 0, ..., 0)(r_1, ..., r_{i-1}, r_i' + r_i'', r_{i+1}, ..., r_d)$ because they have different sizes only in the i th dimension. In the rest of this paper, we only discuss unions of boxy regions if the result is also a boxy region.

For simplicity, we will continue to use r to represent a cube's size (i.e., its size in every dimension is r), and call the direction of the i th dimension the i th direction.

4.1.1 The Packing Instance in ($d > 2$)-D

Given an instance I of the 3-partition problem, $B, S = \{a_1, a_2, ..., a_{3m}\}$, where $m = 3m'$ is a multiple of 3 and $m \geq 8$, we construct an instance I''' of the cube packing problem in $(d > 2)$-D, as follows.

I. Parameters

Let $p = \left\lceil \sqrt{m/2} \right\rceil, A = 3p(p+1)B + 1,\ l = 3A + B, h = A + B/2$, and $w = 3h$.

II. The packing space

The packing space C is a cubic region of size $q = p(p+1)l$. Without loss of generality, let the origin of C be $(0,0,...,0)$. Writing in the boxy region notation, $C = (0,0,...,0)(q,q,...,q)$. We construct C by a series of unions of the following boxy regions:

- ✧ The region $\alpha = (0,0,...,0)(l,mh,w,w,...,w)$, which is the union of the following regions:

  $a_1 = (0,0,0,...,0)(l,mh,h,...,h)$,

  $a_2 = (0,0,h,0,...,0)(l,mh,2h,h,...,h)$,

  $a_3 = (0,0,0,h,0,...,0)(l,mh,w,2h,h,...,h)$,

  …,

  $a_{d-1} = (0,0,0,...,0,h)(l,mh,w,...,w,2h)$.

  It can be verified that $((\alpha_1 \cup \alpha_2) \cup ...) \cup \alpha_{d-1} = \alpha$.

- ✧ The Region β which includes the following regions (not by the union operations):

  $\beta_0 = (0,mh,0,...,0)(l,q-mh,w,...,w,w)$,

  $\beta_1 = (0,0,w,0,...,0)(l,q,q-w,w,...,w)$,

  $\beta_2 = (0,0,0,w,0,...,0)(l,q,q,q-w,w,...,w)$,

  …,

  $\beta_{d-2} = (0,0,0,0...,0,w)(l,q,q,q,...,q-w)$.

- ✧ The Region γ: $\gamma = (l,0,0,...,0)(pl,q,q,...,q)$.
- ✧ The Region δ: $\delta = (l+pl,0,0,...,0)((p-1)(p+1)l,q,q,...,q)$.

The union of the above regions is $C = (0,0,...,0)(q,q,...,q)$.

III. The cubic item list L

The list L of packing cubes includes the following items.

- ✧ The α-cubes: $3^{d-1}m$ cubes, which include:
  - ➢ The α_1-cubes: $3m$ cubes of sizes $A+a_i$, $i = 1,2,...,3m$, respectively;
  - ➢ The α_2-cubes: $3^{d-2}m - m$ cubes, all of size $A+B$;
  - ➢ The α_3-cubes: $2 \cdot 3^{d-2}m - 2m$ cubes, all of size A.
- ✧ The β-cubes, which consist of:
  - ➢ The β_0 -cubes: $p(p+1)-m'$ cubes, all of size $(q-mh)/(p(p+1)-m')$;
  - ➢ The β_1 -cubes: $p(p+1)(p(p+1)-1)$ cubes, all of size $l_\beta = (q-w)/(p(p+1)-1)$;
  - ➢ The β_2-cubes: $p^2(p+1)^2(p(p+1)-1)$ cubes, all of size l_β;

    … …
  - ➢ The β_{d-2}-cubes: $p^{d-2}(p+1)^{d-2}(p(p+1)-1)$ cubes, all of size l_β.
- ✧ The γ-cubes: $(p+1)^{d-1}$ cubes, all of size pl.
- ✧ The δ-cubes: $p^d - p^{d-1}$ cubes, all of size $(p+1)l$.

Lemma 5. The packing instance I''' can be constructed in polynomial time of m, and if I has a partition, then I''' has a packing.

Lemma 6. In the packing instance I''', if all α-cubes and all β-cubes can be packed into the region $\alpha \cup \beta$ (or any region of size $(l,q,...,q)$, $(q,l,q,...,q)$, …, or $(q,...,q,l)$), then the instance I has a partition.

Lemma 7. In any realizable packing of the instance I''', all γ-cubes must be packed into a boxy region of size $w \times p(p+1)l \times ... \times p(p+1)l$, $p(p+1)l \times w \times p(p+1)l \times ... \times p(p+1)l$,…, or $p(p+1)l \times ... \times p(p+1)l \times w$, where $w \in [pl,(p+1)l]$.

Lemma 8. If the instance I''' has an achievable packing, then all α-cubes and all β-cubes can be packed into the region $\alpha \cup \beta$ (or any region of size $l \times p(p+1)l \times ... \times p(p+1)l$, $p(p+1)l \times l \times p(p+1)l \times ... \times p(p+1)l$, …, or $p(p+1)l \times ... \times p(p+1)l \times l$).

Theorem 2. The problem of packing multiple cubes into a cube in $(d > 3)$-D is NP-complete in the strong sense.

5 Conclusions

We have proved that the problem of packing cubes into a cube in $(d > 3)$-D is NP-complete in the strong sense. Together with the 2-D and 3-D results in [9, 11], this completely settles the open question posted in [5, 7].

Acknowledgement. The research of D. Z. Chen was supported in part by NSF under Grant CCF-1217906.

References

1. Bansal, N., Correa, J.R., Kenyon, C., Sviridenko, M.: Bin Packing in Multiple Dimensions: Inapproximability Results and Approximation Schemes. Mathematics of Operations Research **31**(1), 31–49 (2006)
2. Caprara, A., Lodi, A., Monaci, M.: Fast Approximation Schemes for Two-stage, Two-dimensional Bin Packing. Mathematics of Operations Research **30**, 136–156 (2005)
3. Chung, F.R.K., Garey, M.R., Johnson, D.S.: On Packing Two-dimensional Bins. SIAM Journal on Algebraic and Discrete Methods **3**, 66–76 (1982)
4. Correa, J.R., Kenyon, C.: Approximation schemes for multidimensional packing. In: Proc. 15th ACM–SIAM Symposium on Discrete Algorithms, pp. 179–188 (2004)
5. Epstein, L., van Stee, R.: Online Square and Cube Packing. Acta Informatica **41**(9), 595–606 (2005)
6. Garey, M., Johnson, D.: Computer and Intractability – A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

7. Harren, R.: Approximation Algorithms for Orthogonal Packing Problems for Hypercubes. Theoretical Computer Science **410**(44), 4504–4532 (2009)
8. Kohayakawa, Y., Miyazawa, F.K., Raghavan, P., Wakabayashi, Y.: Multidimensional Cube Packing. Algorithmica **40**, 173–187 (2004)
9. Leung, J.Y.-T., Tam, W.T., Wong, C.S., Chin, F.Y.L.: Packing Squares into a Square. Journal of Parallel and Distributed Computing **10**, 271–275 (1990)
10. Li, K., Cheng, K.H.: Complexity of resource allocation and job scheduling problems in partitionable mesh connected systems. In: Proc. of 1st Annual IEEE Symposium of Parallel and Distributed Processing, pp. 358–365. Silver Spring, MD (1989)
11. Lu, Y., Chen, D.Z., Cha, J.: Packing Cubes into a Cube is NP-complete in the Strong Sense. Journal of Combinatorial Optimization **29**(1), 197–215 (2015)
12. Miyazawa, F.K., Wakabayashi, Y.: Cube Packing. Theoretical Computer Science **297**, 355–366 (2003)

Towards Flexible Demands in Online Leasing Problems

Shouwei Li, Alexander Mäcker, Christine Markarian(✉), Friedhelm Meyer auf der Heide, and Sören Riechers

Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany
{shouwei.li,alexander.maecker,christine.markarian, fmadh,soeren.riechers}@uni-paderborn.de

Abstract. We consider online leasing problems in which demands arrive over time and need to be served by leasing resources. We introduce a new model for these problems such that a resource can be leased for K different durations each incurring a different cost (longer leases cost less per time unit). Each demand i can be served anytime between its arrival a_i and its deadline $a_i + d_i$ by a leased resource. The objective is to meet all deadlines while minimizing the total leasing costs. This model is a natural generalization of Meyerson's ParkingPermitProblem (FOCS 2005) in which $d_i = 0$ for all i. We propose an online algorithm that is $\Theta(K + \frac{d_{max}}{l_{min}})$-competitive where d_{max} and l_{min} denote the largest d_i and the shortest available lease length, respectively. We also extend the SetCoverLeasing problem by deadlines and give a competitive online algorithm which also improves on existing solutions for the original SetCoverLeasing problem.

Keywords: Online algorithms · Leasing · Infrastructure problems · Parking permit problem · Deadlines

1 Introduction

Typical infrastructure problems consider scenarios where one has to buy certain resources (e.g., facilities, network nodes, or network connections) in order to generate or improve a given infrastructure (e.g., a supply network). Classical examples of these problems are FacilityLocation, SetCover, and SteinerTree. These problems have a large number of applications in networks, business, logistics and planning. From a theoretical point of view, they are not only widely studied in the offline, but also in the online setting, where the decision of when and which resources to buy must be taken as soon as demands are revealed without knowledge about future demands [2–4]. A common assumption

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center 'On-The-Fly Computing' (SFB 901) and the International Graduate School 'Dynamic Intelligent Systems'.

in related literature is that a resource that is bought once can be used forever without additional costs. This, however, is not true in many real-world scenarios. Instead, the leasing concept is gaining more and more importance in practice. Consider, for example, cloud vendors providing resources to clients. Using leasing, clients are being provided with cheaper, more flexible resources and cloud providers are increasing their profits by reducing their upfront and administrative costs [5]. In 2005, Meyerson [1] introduced the first theoretical Leasing Model with the ParkingPermitProblem. Here, each day, we want to use the car if it is raining, whereas we walk if it is sunny. If we take the car, we must have a valid parking permit, and there are K different types of parking permits (leases), each with its own duration and cost. The goal is to buy a set of parking permits in order to cover all rainy days and minimize the total cost of purchases (without using weather forecasts). Similar models were later studied for infrastructure problems including FacilityLocation, SetCover, and SteinerTree [1,6–9].

We introduce a new model where, in contrast to related work, demands do not have to be served immediately. As a natural extension, demands can be postponed up to some fixed period of time resulting in a deadline for each demand. For instance, consider a travel agency that offers guided tours to tourists. Each day, new tourists may arrive with deadlines that represent the day their vacation ends. Tourists arriving want to attend the guided tour before leaving. Now, the travel agency pays for each time a guide (tour) is needed. To optimize its profit, the agency must make wise decisions regarding when to hire a guide and for how long such that the longer (more consecutive days) a guide is hired, the lower the costs per day will be. We will represent this by different lease types representing different lease lengths. The decision of buying a lease cannot be modified during the process, i. e. we cannot tell the guide to stay shorter or longer after having hired her.

Another application can be found in the cloud, where cloud clients, who are flexible regarding when to use the resources (e.g., any day within two weeks will do), will be happy to be offered better resource prices for a later day by cloud vendors.

Our Contribution. In the light of capturing such scenarios, we extend the line of leasing by introducing deadlines. We propose the OnlineLeasingWithDeadlines problem (OLD) for which we give a deterministic (optimal) algorithm. OLD captures the scenario introduced above, where a travel agency has to provide guided tours for tourists arriving and leaving their vacation at different times. We give an $\mathcal{O}(K + \frac{d_{max}}{l_{min}})$-competitive online algorithm for the OLD problem, where K is the number of leases, d_{max} the longest client length (i. e., the longest vacation of a tourist) and l_{min} the shortest lease length (i. e., the minimum length a guide can be hired for).

In the second part of the paper, we also extend the SetCoverLeasing model [6,9], and introduce the SetCoverLeasingWithDeadlines problem (SCLD) which we solve by extending techniques we develop for OLD. Here, we give an $\mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max})$-competitive online algorithm. Our results also imply an improvement for the SetCoverLeasing problem by removing the time dependency from the competitive factor.

Organization of the Paper. Section 2 presents the state of the art and gives some preliminaries. Section 3 introduces the new model and gives a deterministic algorithm. Section 4 introduces SCLD and presents a randomized algorithm for the latter. Section 5 concludes the paper with suggestions for future work.

2 Related Work and Preliminaries

In this section, we give an overview of leasing problems and present some preliminaries.

Related Work. Meyerson [1] gave a deterministic $\mathcal{O}(K)$-competitive and a randomized $\mathcal{O}(\log K)$-competitive algorithm along with matching lower bounds for the ParkingPermitProblem. Inspired by Meyerson's work, Anthony and Gupta [6] generalized his idea by introducing the first leasing variants of FacilityLocation, SetCover, and SteinerTree: FacilityLeasing, SetCoverLeasing, and SteinerTreeLeasing, respectively. They gave offline algorithms for these problems, resulting from an interesting relationship between infrastructure leasing problems and stochastic optimization. They achieved $\mathcal{O}(K)$, $\mathcal{O}(\log n)$, and $\mathcal{O}(\min(K, \log n))$-approximations for the offline variants of metric FacilityLeasing, SetCoverLeasing, and SteinerTreeLeasing, respectively. Nagarajan and Williamson [7] later improved the $\mathcal{O}(K)$-approximation to a 3-approximation and gave an $\mathcal{O}(K \log n)$-competitive algorithm for the online variant of metric FacilityLeasing. Kling et al. [8] improved the results by Nagarajan and Williamson [7] for FacilityLeasing by removing the dependency on n (and thereby on time), where n is the number of clients. They gave an $\mathcal{O}(l_{max} \log(l_{max}))$-competitive algorithm where l_{max} is the maximum lease length. Abshoff et al. [9] gave the first online algorithm for SetCoverLeasing and improved previous results for online non-leasing variants of SetCover.

Preliminaries. Meyerson [1] showed that it is sufficient to have a simplified Leasing Model in which (1) lease lengths l_k are powers of two and (2) leases of the same type do not overlap. Instances/solutions having these properties are said to obey the *interval model*. Similar properties have been used in related settings [1,8] and we detail these insights for completeness' sake. The following lemma states the effect of this model on the approximation guarantee.

Lemma 1. *Any c-competitive algorithm for the interval model can be transformed to a $4c$-competitive algorithm for the general leases model.*

Proof. Consider a leasing problem instance I for general leases and construct a new instance I' by rounding each lease length $l_k \in \mathbb{N}$ to the next larger power of two. That is, the lease lengths of I' are $l'_k := 2^{\lceil \log l_k \rceil}$. Let S' denote the solution constructed by the c-approximation algorithm for the interval model when given I'. From S', we construct a solution S for I as follows: for each lease of type k bought at time t in solution S', buy two consecutive leases of type k at times t and $t + l_k$. Since $l_k + l_k \geq l'_k$, any lease pair in S covers at least all the demands covered by the original lease in S'. Moreover, we have $\text{cost}(S) = 2\,\text{cost}(S') \leq 2c \cdot \text{cost}(\textsc{Opt}')$, where $\textsc{Opt}$' denotes an optimal solution for I' in the interval model. Now, note that an optimal solution $\textsc{Opt}$ for I in the general model yields a solution $\tilde{S}$ for I' in the interval model as follows: for each lease of type k bought at time t in solution $\textsc{Opt}$, buy two leases of type k at times $\lfloor t/l'_k \rfloor \cdot l'_k$ and $\lceil t/l'_k \rceil \cdot l'_k$. These leases cover at least all the demands of the original lease and obey the interval model. Thus, we get $\text{cost}(\textsc{Opt}') \leq \text{cost}(\tilde{S}) = 2\,\text{cost}(\textsc{Opt})$. The lemma's statement follows by combining both inequalities.

3 Online Leasing with Deadlines

In this section, we introduce the OnlineLeasingWithDeadlines problem (OLD) and give a deterministic primal-dual algorithm.

3.1 Problem Definition

On each day t, a number of clients with deadlines $t + d_i$ (we say a client with *interval* $[t, t + d_i]$, where each day corresponds to a distance of 1 in the interval) arrives. There are K different types of leases, each with its own duration and cost (longer leases tend to cost less per day). A client arriving on day t with deadline $t + d$ is *served* if there is a lease which covers at least one day of its interval. This also implies that we can replace all clients arriving on a day t by only the client with the lowest deadline that arrives on that day. Thus, throughout this chapter, we will assume without loss of generality that on every day t, either (i) no client or (ii) only one client with deadline $t + d$ arrives. The goal is to buy a set of leases such that all arriving clients are served while minimizing the total cost of purchases.

A lease of type k has cost c_k and length l_k. l_{min} and l_{max} denote the shortest and the longest lease length, respectively. We denote by d_{max} and d_{min} the longest and the shortest interval length of the clients, respectively. An online algorithm now does not only need to serve clients while minimizing cost, but also needs to decide when to serve a client. Since resources expire after some time, decisions regarding when to serve a client are critical. An online algorithm may decide to serve a client on some day just to realize later on that postponing it would have been a better choice because a later lease could have served more clients. Or, the opposite is true, where an online algorithm may decide to postpone serving a client whereas serving it earlier by enlarging a lease that has been bought would have cost less.

We formulate OLD using integer linear programming (ILP) (see Figure 1). We refer to a type k lease starting at time t as (k, t), a client arriving at time

t with deadline $t+d$ as (t,d), and an interval $[a, a+b]$ as I_a^b. The collection of all leases is L and the collection of all clients is D. We denote by L_t all leases covering day t. We say a lease $(k,t') \in L$ is a *candidate* to client $(t,d) \in D$ if $I_t^d \cap I_{t'}^{l_k} \neq \emptyset$. The sum in the objective function represents the costs of buying the leases. The indicator variable $X_{(k,t)}$ tells us whether lease (k,t) is bought or not. The primal constraints guarantee that each client $(t,d) \in D$ is served. A dual variable $Y(t,d)$ is assigned to each client (t,d).

$$\min \sum_{(k,t)\in L} X_{(k,t)} \cdot c_k$$

$$\text{Subject to: } \forall (t,d) \in D : \sum_{(k,t')\in L, I_t^d \bigcap I_{t'}^{l_k} \neq \emptyset} X_{(k,t')} \geq 1$$

$$\forall (k,t) \in L : X_{(k,t)} \in \{0,1\}$$

$$\max \sum_{(t,d)\in D} Y_{(t,d)}$$

$$\text{Subject to: } \forall (k,t) \in L : \sum_{(t',d)\in D, I_{t'}^d \bigcap I_t^{l_k} \neq \emptyset} Y_{(t',d)} \leq c_k$$

$$\forall (t,d) \in D : Y_{(t,d)} \geq 0$$

Fig 1. ILP Formulation of OLD

3.2 Deterministic Algorithm

In this section, we present a deterministic primal-dual algorithm for OLD, for which we give the analysis in the following section.

We adopt the *interval model* (Lemma 1) in which leases of type k are available only at times t that are a multiple of the corresponding lease length l_k. Thus, any day t can be covered by exactly K different leases. Therefore, when a client $(t,d) \in D$ arrives, our algorithm needs to decide on which day $t' \in [t, t+d]$ to serve it and it needs to specify one of the K leases in $L_{t'}$. For every lease $(k,t) \in L$, we define its *contribution* to be the sum of the values of the dual variables corresponding to clients having (k,t) as a candidate. We say (t',d) *contributes* to (k,t) if (k,t) is a candidate of (t',d) and $Y_{(t',d)} > 0$. Two clients (t',d') and (t,d) with $t' < t$ *intersect* if their corresponding intervals $I_{t'}^{d'}$ and I_t^d intersect at $t'+d'$.

Algorithm. When a client (t,d) arrives, if it does not intersect any client (t',d') with a non-zero dual variable where $t' < t$, we perform the following two steps.

Step 1: We increase the dual variable $Y_{(t,d)}$ of the client until the constraint of some candidate (k, t') becomes tight, i.e.,

$$\sum_{(t,d)\in D: I_t^d \cap I_{t'}^{l_k} \neq \emptyset} Y_{(t,d)} = c_k.$$

We then buy all the leases in L_t with a tight constraint (we set their primal variable to 1). At this point, the following proposition holds.

Proposition 1. *There exists at least one lease with a tight constraint that covers t.*

Proof. Assume, for contradiction, that there is no lease with a tight constraint in L_t. Then according to the algorithm, there must be a lease with a tight constraint in L_j, $j \in [t+1, t+d]$ (we do not stop increasing the client's dual variable until some constraint becomes tight). Moreover, before (t, d) arrives, the contribution to every lease in L_t is at least the contribution to its corresponding lease in L_j, $j \in [t+1, t+d]$. To show that the latter is true, assume, for contradiction, that there is a lease (k, t') in L_j, $j \in [t+1, t+d]$, with a contribution greater than that of its corresponding lease (k, t'') in L_t. Then, there must be a client which has contributed to (k, t') and not to (k, t'') (a client contributes the same to all its candidates). This is only possible if this client has arrived after (t, d), which is a contradiction. Hence, if the constraint of some lease in L_j, $j \in [t+1, t+d]$, becomes tight when (t, d) arrives, then the constraint of its corresponding lease in L_t must become tight as well (at any day, there are exactly K lease types). $\square$

Step 2: By the proposition above, we have that the algorithm buys at least one lease in L_t. Even though the client is now served, we do one more step. We buy the lease(s) from L_{t+d} which correspond(s) to what is bought in Step 1 from L_t (we set the primal variable(s) to 1).

3.3 Analysis

We show that the primal-dual algorithm above is $\mathcal{O}(K)$-competitive for *uniform* OLD and $\mathcal{O}(K + \frac{d_{max}}{l_{min}})$-competitive for *non-uniform* OLD. We also show that the analysis of our algorithm is tight. This also implies an $\mathcal{O}(K)$-competitive factor for the ParkingPermitProblem (we just set d_{max} to 0) which coincides with the tight result given by Meyerson [1].

Proposition 2. *Both the primal and the dual solutions constructed by the algorithm are feasible.*

Proof. It is easy to see that the dual constraints are never violated since the algorithm stops increasing the dual variables as soon as some constraint becomes tight. As for the primal solution, we show that each client $(t, d) \in D$ is served. When a client (t, d) arrives, we have two possibilities: either (t, d) intersects a previous client or it does not. If it does not, then our algorithm makes sure it is served in Step 1. Otherwise if it intersects a previous client (t', d) with $Y_{(t',d)}$ being zero, our algorithm makes sure it serves (t, d) in Step 1. If $Y_{(t',d)}$ is greater than zero, then our algorithm already covered days t' and $t' + d$ to serve (t', d). Since (t, d) and (t', d) intersect at $t' + d$, (t, d) is therefore served as well. □

Theorem 1. *The primal-dual algorithm achieves an optimal $\mathcal{O}(K)$- and an $\mathcal{O}(K + \frac{d_{max}}{l_{min}})$-competitive ratio for uniform and non-uniform OLD respectively.*

Proof. Let $P \subseteq L$ denote the primal solution constructed by the algorithm. Because the dual constraint is tight for each $(k, t) \in P$, we have

$$c_k = \sum_{(t',d) \in D : I_{t'}^d \cap I_t^{l_k} \neq \emptyset} Y_{(t',d)}.$$

Hence,

$$\sum_{(k,t)\in P} c_k = \sum_{(k,t)\in P} \sum_{(t',d)\in D : I_{t'}^d \cap I_t^{l_k} \neq \emptyset} Y_{(t',d)} = \sum_{(t',d)\in D} Y_{(t',d)} \sum_{(k,t)\in P : I_{t'}^d \cap I_t^{l_k} \neq \emptyset} 1.$$

Whenever the algorithm buys leases to serve $(t', d) \in D$, it only buys candidates from $L_{t'}$ (Step 1) and $L_{t'+d}$ (Step 2). Since there are exactly K leases at any day, it therefore buys at most $2K$ candidates. If the algorithm does not buy any further candidates of (t', d), we get an $\mathcal{O}(K)$-competitive ratio by weak duality theorem (both primal and dual solutions are feasible) since

$$\sum_{(k,t)\in P : I_{t'}^d \cap I_t^{l_k} \neq \emptyset} 1 \leq 2K.$$

This will be the case for *uniform* OLD since any client sharing common candidates with (t', d) intersects (t', d) at $t' + d$ thus being served at $t' + d$ and the algorithm does not buy any further candidates of (t', d). As for *non-uniform* OLD, the algorithm may buy more of (t', d)'s candidates when new clients sharing common candidates with (t', d) arrive in the coming days. We upper bound the total number of these candidates as follows.

$$\sum_{(k,t)\in P : I_{t'}^d \cap I_t^{l_k} \neq \emptyset} 1 \leq \sum_{i=t'}^{t'+d} |L_i| \leq \sum_{j=1}^{K} \left\lceil \frac{d_{max}}{l_j} \right\rceil.$$

By Lemma 1 we have that l_j's are increasing and powers of two. Hence, the right sum above can be bounded by the sum of a geometric series with ratio half.

$$\sum_{j=1}^{K} \left\lceil \frac{d_{max}}{l_j} \right\rceil \leq K + d_{max} \left[\frac{1}{l_1} \left(\frac{1 - (1/2)^K}{1 - 1/2} \right) \right] = K + d_{max} \left[\frac{2}{l_1} \left(1 - (1/2)^K \right) \right].$$

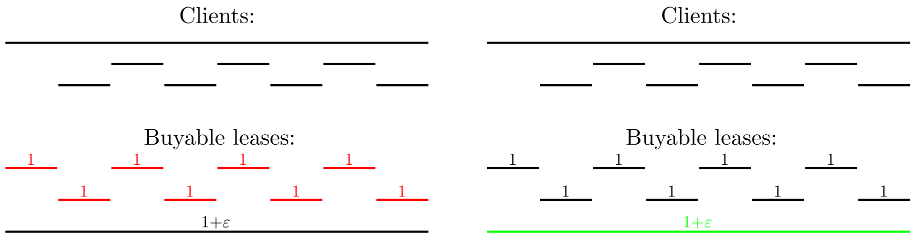
Since $K \geq 1$ we have

$$K + d_{max}\left[\frac{2}{l_1}\left(1 - (1/2)^K\right)\right] \leq K + \frac{d_{max}}{l_1}.$$

Therefore,

$$\sum_{(k,t)\in P: I_{t'}^{d} \cap I_t^{l_k} \neq \emptyset} 1 \leq K + \frac{d_{max}}{l_{min}}$$

since the algorithm can not buy more than $K + \frac{d_{max}}{l_{min}}$ candidates. □



(a) Leases bought by our primal-dual algorithm are marked in red.

(b) Leases bought by an optimal algorithm are marked in green.

Fig. 2. Comparison of our primal-dual and an optimal algorithm for a specific instance of our problem. It is easy to see that the primal-dual algorithm pays almost d_{max}/l_{min} times what the optimal algorithm would pay.

Proposition 3. *The analysis of the aforementioned algorithm is tight.*

Proof. A lower bound of $\Omega(K)$ follows immediately from the lower bound of $\Omega(K)$ for the ParkingPermitProblem by setting $d_{max} = 0$. We now give a tight example for $\Omega(d_{max}/l_{min})$ for the non-uniform case. Let d_{max} and l_{min} be arbitrary. For our problem instance, we start with a client $(0, d_{max})$ and add clients $((i-1)\cdot l_{min}, i \cdot l_{min})$ for $i \in \{2, \ldots, \lfloor d_{max}/l_{min} \rfloor\}$. Similarly, we add 2 different lease types, one with length l_{min} and cost 1, and one with length $2^{\lceil \log_2(d_{max}) \rceil}$ and cost $1 + \varepsilon$. See Figure 2 for a visualization. Now, in order to cover client $(0, d_{max})$, the dual variable of this client is increased until

$$\sum_{(t,d)\in D: I_t^{d} \cap I_{t'}^{l_k} \neq \emptyset} Y_{(t,d)} = c_k, \tag{1}$$

and this happens at the same time for all leases of length l_{min} in the interval $I_0^{d_{max}}$. The algorithm then only buys the leases at the start and at the end point. However, to cover clients $((i-1)\cdot l_{min}, i \cdot l_{min})$ for $i \in \{2, \ldots, \lfloor d_{max}/l_{min} \rfloor\}$, the algorithm buys all the short leases, as constraint (1) is already tight from the prior step. This leads to an overall cost of at least $\lfloor d_{max}/l_{min} \rfloor$, whereas the optimal algorithm only buys the long lease with cost $1 + \varepsilon$. □

4 Application to Set Cover Leasing

In this section, we introduce the SetCoverLeasingWithDeadlines problem (SCLD) and give an $\mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max})$-competitive algorithm.

4.1 Problem Definition

SCLD is a generalization of SetCoverLeasing in which elements arrive over time and must be covered by sets from a family of subsets of these elements. Each set can be leased for K different periods of time. Leasing a set S for a period k incurs a cost c_S^k and allows S to cover its elements for the next l_k time steps. The objective is to minimize the total cost of the sets leased, such that elements arriving at any time t are covered by sets which contain them and are leased during time t. SCLD extends SetCoverLeasing by allowing elements to have deadlines and be covered any time before their deadline. We define SCLD analogously to *non-uniform* OLD and formulate it using ILP (see Figure 3).

We denote by p the maximum number of sets an element belongs to, by n the number of elements, and by m the number of sets. We refer to a set S with lease type k starting on day t as (S, k, t) and an element e arriving on day t with deadline $t + d$ as (e, t, d). The collection of all set triples is F and the collection of all element triples is U. We say $(S, k, t') \in F$ is a *candidate* to $(e, t, d) \in U$ if $e \in S$ and $I_t^d \cap I_{t'}^{l_k} \neq \emptyset$. The sum in the objective function represents the costs of buying the sets. The indicator variable $X_{(S,k,t)}$ tells us whether (S, k, t) is bought or not. An element is *covered* if at least one of its candidates is bought. The primal constraints guarantee that each $(e, t, d) \in U$ is covered.

$$\min \sum_{(S,k,t)\in F} X_{(S,k,t)} \cdot c_S^k$$

$$\text{Subject to: } \forall (e,t,d) \in U : \sum_{(S,k,t')\in F, I_t^d \bigcap I_{t'}^{l_k} \neq \emptyset, e \in S} X_{(S,k,t')} \geq 1$$

$$\forall (S,k,t) \in F : X_{(S,k,t)} \in \{0,1\}$$

Fig 3. ILP of SCLD

4.2 Randomized Algorithm

In this section, we present a randomized algorithm for SCLD. We denote by $F_{(e,t,d)}$ the collection of all candidates of (e, t, d). Our algorithm first solves the LP of SCLD and then rounds it to solve its ILP. The algorithm maintains for each set $(S, k, t) \in U$, $2 \lceil \log(l_{max}) \rceil$ independent random variables $r_{(Skt)(q)}$, $1 \leq q \leq 2 \lceil \log(l_{max}) \rceil$, distributed uniformly in the interval $[0, 1]$. We define $\mu_{Skt} := \min\{r_{(Skt)(q)}\}$.

Algorithm 1. SetCoverLeasingWithDeadlines

```
When an element $(e, t, d)$ arrives,
(i) (LP solution) *while* $\sum_{(S,k,t)\in F_{(e,t,d)}} X_{(S,k,t)} < 1$;
        $X_{(S,k,t)} = X_{(S,k,t)} \cdot (1 + 1/c_S^k) + \frac{1}{|F_{(e,t,d)}| \cdot c_S^k}$
(ii) (ILP solution) Round $X_{(S,k,t)}$ to 1 if $X_{(S,k,t)} > \mu_{Skt}$ and if $(e, t, d)$ is not yet
covered, buy the cheapest $(S, k, t) \in F_{(e,t,d)}$ (set its primal variable to 1).
```

4.3 Analysis

We show that the algorithm above is $\mathcal{O}(\log(p \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max}) = \mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max})$-competitive for SCLD.

It is easy to see that Algorithm 1 constructs a feasible solution ILP to SetCoverLeasing. To compute the total expected cost C_{ILP} of ILP, we first bound the cost of the LP solution C_{LP} by $\mathcal{O}(\log(p \cdot (K + \frac{d_{max}}{l_{min}}))) = \mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}}))) \cdot \text{Opt}$, where Opt is the optimal solution cost of ILP. Then, we show that C_{ILP} is at most $O(\log l_{max})$ times C_{LP} and hence deduce the expected $\mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max})$-competitive factor of the algorithm.

To do so, we partition the time horizon into intervals of length l_{max}. Due to the interval model (Lemma 1), all leases of all sets end on days $i : i = 0 \mod l_{max}$. Hence, we bound C_{ILP} over any interval of length l_{max} by $\mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max})) \cdot \text{Opt}_{l_{max}}$, where $\text{Opt}_{l_{max}}$ is the optimum over the corresponding interval of length l_{max}. Summing up over all such intervals yields our competitive factor for SCLD.

Lemma 2. *The cost $C_{LP(l_{max})}$ of the LP solution over an interval of length l_{max} is at most $\mathcal{O}(\log(p \cdot (K + \frac{d_{max}}{l_{min}}))) \cdot \text{Opt}_{l_{max}} = \mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}}))) \cdot \text{Opt}_{l_{max}}$ where $\text{Opt}_{l_{max}}$ is the cost of the optimal solution over this interval.*

Proof. We fix any interval of length l_{max} from our partition. Any set (S_{OPT}, k, t') in the optimum solution over this interval has been a candidate for some element (e, t, d). When (e, t, d) arrives, our algorithm increases the primal variables of (e, t, d)'s candidates until they sum up to one. After $\mathcal{O}(c_{S_{OPT}}^k \cdot \log |F_{(e,t,d)}|)$ increases, $X_{(S_{OPT},k,t')}$ becomes greater than one and the algorithm makes no further increases. Furthermore, these increases never add a total of more than 2 to the primal variables. This is because

$$\sum_{(S,k,t)\in F_{(e,t,d)}} c_S^k \cdot \left(\frac{X_{(S,k,t)}}{c_S^k} + \frac{1}{c_S^k} \cdot |F_{(e,t,d)}| \right) \leq 2,$$

since $\sum_{(S,k,t)\in F_{(e,t,d)}} X_{(S,k,t)} < 1$ before the increase. The same holds for any other set in the optimum solution over this interval. Using a similar argument as in OLD, we can bound $|F_{(e,t,d)}|$ by $p \cdot (K + \frac{d_{max}}{l_{min}})$ (there are at most $\frac{d_{max}}{l_{min}}$ leases for each of the at most p candidate sets). This completes the proof of the lemma. □

Lemma 3. *The cost $C_{ILP(l_{max})}$ of the ILP solution over an interval of length l_{max} is at most $\mathcal{O}(\log l_{max}) \cdot C_{LP(l_{max})}$, where $C_{LP(l_{max})}$ is the cost of the LP solution over this interval.*

Proof. We fix any interval of length l_{max} from our partition. The probability to buy a set $(S, k, t) \in F$ in this interval is proportional to the value of its primal variable. Hence, $C_{ILP(l_{max})}$ is upper bounded by

$$\sum_{(S,k,t)\in F} 2\log(l_{max}+1) \cdot c_S^k \cdot X_{(S,k,t)}.$$

To guarantee feasibility, every time an element is not covered, the algorithm buys the cheapest candidate, which is a lower bound to $\textsc{Opt}_{l_{max}}$. The probability that an element is not covered is at most $1/(l_{max})^2$. Since the random variables are drawn independently, we can add the expected costs incurred by the corresponding at most l_{max} elements and deduce a negligible expected cost of $l_{max} \cdot 1/(l_{max})^2 \cdot \textsc{Opt}_{l_{max}}$ which concludes the proof of the lemma. □

From the two lemmas above, we deduce the following theorem.

Theorem 2. *There is an online randomized algorithm for SCLD with a competitive factor of*

$$\mathcal{O}(\log(p \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max}) = \mathcal{O}(\log(m \cdot (K + \frac{d_{max}}{l_{min}})) \log l_{max}).$$

SetCoverLeasing is nothing but a special case of SCLD if we set $d_{max} = 0$. Hence, we deduce the following corollary thereby improving the previous result for SetCoverLeasing [9] from $\mathcal{O}(\log(m \cdot K) \log n)$ to $\mathcal{O}(\log(m \cdot K) \log l_{max})$ by removing the dependency on n and therefore on time.

Corollary 1. *There is an online randomized algorithm for* SetCoverLeasing *that has a time-independent $\mathcal{O}(\log(p \cdot K) \log l_{max}) = \mathcal{O}(\log(m \cdot K) \log l_{max})$-competitive factor.*

5 Conclusion

We have extended the line of leasing by introducing a new model for online leasing problems. As a first infrastructure leasing problem, we have defined SetCoverLeasing with the new model and proceeding in this direction, we plan to study other infrastructure leasing problems starting with FacilityLeasing and SteinerTreeLeasing.

Our model introduces *flexibility* to demands, thus capturing more general applications. Demands in our model have the flexibility of having a deadline. It will be interesting to extend this work to include models that handle other flexibilities (e.g., can be served on specific days within some period of time). Furthermore, demands in our model require a single day to be served. Allowing demands that require more than one day to be served can be a natural extension of our model. Even though the techniques used in this paper do not carry over directly to this extension, they still give first insights.

Along the same line of leasing lies the important unanswered question of what the price of online leasing is. All algorithms for online leasing problems so far build upon the algorithms for the non-leasing variants of their corresponding problems and the ParkingPermitProblem algorithm. Since all these problems generalize the ParkingPermitProblem, the only lower bound we have for these problems is $\Omega(K + f(\cdot))$, where K is the lower bound for the ParkingPermitProblem and $f(\cdot)$ the lower bound for the underlying non-leasing variant of the problem. It is still not known whether we can prove stronger lower bounds for these problems.

Another interesting direction would be to have a stochastic view of online leasing problems, where demands and/or their deadlines are given according to some probability distribution.

References

1. Meyerson, A.: The parking permit problem. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science(FOCS), pp. 274–284 (2005)
2. Meyerson, A.: Online facility location. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science(FOCS), pp. 426–431 (2001)
3. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The online set cover problem. In: Proceedings of the 35th Annual ACM Symposium on the Theory of Computation(STOC), pp. 100–105 (2003)
4. Berman, P., Coulston, C.: Online algorithms for Steiner tree problems. In: Proceedings of the 29th Annual ACM Symposium on the Theory of Computation(STOC), pp. 344–353 (1997)
5. Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., Tsafrir, D.: Deconstructing amazon EC2 spot instance pricing. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science(Cloud-Com), pp. 304–311 (2011)
6. Anthony, B.M., Gupta, A.: Infrastructure leasing problems. In: Fischetti, M., Williamson, D.P. (eds.) IPCO 2007. LNCS, vol. 4513, pp. 424–438. Springer, Heidelberg (2007)
7. Nagarajan, C., Williamson, D.P.: Offline and online facility leasing. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 303–315. Springer, Heidelberg (2008)
8. Kling, P., Meyer auf der Heide, F., Pietrzyk, P.: An algorithm for online facility leasing. In: Even, G., Halldórsson, M.M. (eds.) SIROCCO 2012. LNCS, vol. 7355, pp. 61–72. Springer, Heidelberg (2012)
9. Abshoff, S., Markarian, C., Meyer auf der Heide, F.: Randomized online algorithms for set cover leasing problems. In: Zhang, Z., Wu, L., Xu, W., Du, D.-Z. (eds.) COCOA 2014. LNCS, vol. 8881, pp. 25–34. Springer, Heidelberg (2014)

Lower Bounds for the Size of Nondeterministic Circuits

Hiroki Morizumi(✉)

Interdisciplinary Graduate School of Science and Engineering,
Shimane University, Matsue, Shimane 690-8504, Japan
morizumi@cis.shimane-u.ac.jp

Abstract. Nondeterministic circuits are a nondeterministic computation model in circuit complexity theory. In this paper, we prove a $3(n-1)$ lower bound for the size of nondeterministic U_2-circuits computing the parity function. It is known that the minimum size of (deterministic) U_2-circuits computing the parity function exactly equals $3(n-1)$. Thus, our result means that nondeterministic computation is useless to compute the parity function by U_2-circuits and cannot reduce the size from $3(n-1)$. To the best of our knowledge, this is the first nontrivial lower bound for the size of nondeterministic circuits (including formulas, constant depth circuits, and so on) with unlimited nondeterminism for an explicit Boolean function. We also discuss an approach to proving lower bounds for the size of deterministic circuits via lower bounds for the size of nondeterministic restricted circuits.

1 Introduction

Proving lower bounds for the size of Boolean circuits is a central topic in circuit complexity theory. The gate elimination method is one of well-known proof techniques to prove lower bounds for the size of Boolean circuits, and has been used to prove many linear lower bounds including the best known lower bounds for the size of Boolean circuits over the basis B_2 [2][3] and the basis U_2 [6][4].

In this paper, we show that the gate elimination method works well also for nondeterministic circuits. For deterministic circuits, it is known that the minimum size of U_2-circuits computing the parity function exactly equals $3(n-1)$ [7]. The proof of the lower bound is based on the gate elimination method and has been known as a typical example that the method is effective. In this paper, we prove a $3(n-1)$ tight lower bound for the size of nondeterministic U_2-circuits computing the parity function, which means that nondeterministic computation is useless to compute the parity function by U_2-circuits and cannot reduce the size from $3(n-1)$.

To the best of our knowledge, our result is the first nontrivial lower bound for the size of nondeterministic circuits (including formulas, constant depth circuits, and so on) with unlimited nondeterminism for an explicit Boolean function.

This work was supported by JSPS KAKENHI Grant Number 15K11986.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 289–296, 2015.
DOI: 10.1007/978-3-319-21398-9_23

In this paper, we show that, for U_2-circuits, a known proof technique (i.e., the gate elimination method) for deterministic circuits is applicable to the nondeterministic case. This implies the possibility that proving lower bounds for the size of nondeterministic circuits may not be so difficult in contrast with the intuition and known proof techniques might be applicable to the nondeterministic case also for other circuits. One of motivations to prove lower bounds for the size of nondeterministic circuits is from some relations between the size of deterministic circuits and nondeterministic circuits. We also discuss an approach to proving lower bounds for the size of deterministic circuits via lower bounds for the size of nondeterministic restricted circuits.

2 Preliminaries

2.1 Definitions

Circuits are formally defined as directed acyclic graphs. The nodes of in-degree 0 are called *inputs*, and each one of them is labeled by a variable or by a constant 0 or 1. The other nodes are called *gates*, and each one of them is labeled by a Boolean function. The *fan-in* of a node is the in-degree of the node, and the *fan-out* of a node is the out-degree of the node. There is a single specific node called *output*.

We denote by B_2 the set of all Boolean functions $f : \{0,1\}^2 \to \{0,1\}$. By U_2 we denote $B_2 - \{\oplus, \equiv\}$, i.e., U_2 contains all Boolean functions over two variables except for the XOR function and its complement. A Boolean function in U_2 can be represented as the following form:

$$f(x,y) = ((x \oplus a) \wedge (y \oplus b)) \oplus c,$$

where $a, b, c \in \{0,1\}$. A U_2*-circuit* is a circuit in which each gate has fan-in 2 and is labeled by a Boolean function in U_2. A B_2*-circuit* is similarly defined.

A *nondeterministic circuit* is a circuit with *actual inputs* $(x_1, \ldots, x_n) \in \{0,1\}^n$ and some further inputs $(y_1, \ldots, y_m) \in \{0,1\}^m$ called *guess inputs*. A nondeterministic circuit computes a Boolean function f as follows: For $x \in \{0,1\}^n$, $f(x) = 1$ iff there exists a setting of the guess inputs $\{y_1, \ldots, y_m\}$ which makes the circuit output 1. In this paper, we call a circuit without guess inputs a *deterministic circuit* to distinguish it from a nondeterministic circuit.

The *size* of a circuit is the number of gates in the circuit. The *depth* of a circuit is the length of the longest path from an input to the output in the circuit. We denote by $size^{\mathrm{dc}}(f)$ the size of the smallest deterministic U_2-circuit computing a function f, and denote by $size^{\mathrm{ndc}}(f)$ the size of the smallest nondeterministic U_2-circuit computing a function f.

While we mainly consider U_2-circuits in this paper, we also consider other circuits in Section 4. A *formula* is a circuit whose fan-out is restricted to 1. The parity function of n inputs $x_1, \ldots, x_n$, denoted by Parity$_n$, is 1 iff $\sum x_i \equiv 1 \pmod 2$.

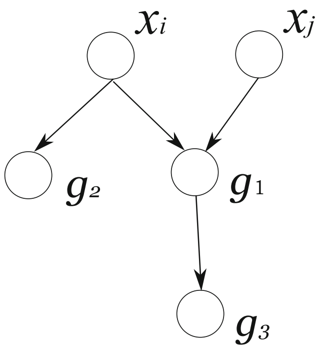


Fig. 1. Proof of Theorem 1

2.2 The Gate Elimination Method

The proof of our main result is based on the gate elimination method, and based on the proof of the deterministic case. In this subsection, we have a quick look at them.

Consider a gate g which is labeled by a Boolean function in U_2. Recall that any Boolean function in U_2 can be represented as the following form:

$$f(x, y) = ((x \oplus a) \land (y \oplus b)) \oplus c,$$

where $a, b, c \in \{0, 1\}$. If we fix one of two inputs of g so that $x = a$ or $y = b$, then the output of g becomes a constant c. In such case, we call that g is *blocked.*

Theorem 1 (Schnorr [7]).

$$size^{\text{dc}}(\text{Parity}_n) = 3(n - 1).$$

Proof. Assume that $n \geq 2$. Let C be an optimal deterministic U_2-circuit computing Parity_n. Let g_1 be a top gate in C, i.e., whose two inputs are connected from two inputs x_i and x_j, $1 \leq i, j \leq n$. Then, x_i must be connected to another gate g_2, since, if x_i is connected to only g_1, then we can block g_1 by an assignment of a constant to x_j and the output of C becomes independent from x_i, which contradicts that C computes Parity_n. By a similar reason, g_1 is not the output of C. Let g_3 be a gate which is connected from g_1. See Figure 1.

We prove that we can eliminate at least 3 gates from C by an assignment to x_i. We assign a constant 0 or 1 to x_i such that g_1 is blocked. Then, we can eliminate g_1, g_2 and g_3. If g_2 and g_3 are the same gate, then the output of g_2 $(= g_3)$ becomes a constant, which means that g_2 $(= g_3)$ is not the output of C and we can eliminate another gate which is connected from g_2 $(= g_3)$. Thus, we can eliminate at least 3 gates and the circuit come to compute Parity_{n-1} or $\neg\text{Parity}_{n-1}$. For deterministic circuits, it is obvious that $size^{\text{dc}}(\text{Parity}_{n-1}) = size^{\text{dc}}(\neg\text{Parity}_{n-1})$. Therefore,

$$\begin{aligned} size^{\mathrm{dc}}(\mathrm{Parity}_n) &\geq size^{\mathrm{dc}}(\mathrm{Parity}_{n-1}) + 3 \\ &\vdots \\ &\geq 3(n-1). \end{aligned}$$

$x \oplus y$ can be computed with 3 gates by the following form:

$$(x \wedge \neg y) \vee (\neg x \wedge y).$$

Therefore, $size^{\mathrm{dc}}(\mathrm{Parity}_n) \leq 3(n-1)$. □

3 Proof of the Main Result

In this section, we prove the main theorem. For deterministic circuits, there must be a top gate whose two inputs are connected from two (actual) inputs x_i and x_j, $1 \leq i, j \leq n$. However, for nondeterministic circuits, there may be no such gate, since there are not only actual inputs but also guess inputs in nondeterministic circuits. We need to defeat the difficulty. See Section 2.2 for the definition of "block".

Theorem 2.

$$size^{\mathrm{ndc}}(\mathrm{Parity}_n) = 3(n-1).$$

Proof. By theorem 1,

$$size^{\mathrm{ndc}}(\mathrm{Parity}_n) \leq size^{\mathrm{dc}}(\mathrm{Parity}_n) = 3(n-1).$$

Assume that $n \geq 2$. Let C be an optimal nondeterministic U_2-circuit computing Parity_n. We prove that we can eliminate at least 3 gates from C by an assignment of a constant 0 or 1 to an actual input.

Case 1. There is an actual input x_i, $1 \leq i \leq n$, which is connected to at least two gates.

Let g_1 and g_2 be gates which are connected from x_i. Since we can block g_1 by an assignment of a constant to x_i, g_1 is not the output of C and there is a gate g_3 which is connected from g_1. See Figure 2.

We prove that we can eliminate at least 3 gates from C by an assignment to x_i. We assign a constant 0 or 1 to x_i such that g_1 is blocked. Then, we can eliminate g_1, g_2 and g_3. If g_2 and g_3 are the same gate, then the output of g_2 ($= g_3$) becomes a constant, which means that g_2 ($= g_3$) is not the output of C and we can eliminate another gate which is connected from g_2 ($= g_3$). Thus, we can eliminate at least 3 gates.

Case 2. Every actual input is connected to at most one gates.

Let g_1 be a gate in C such that one of two inputs is connected from an actual input x_i and the other is connected from a node v whose output is dependent

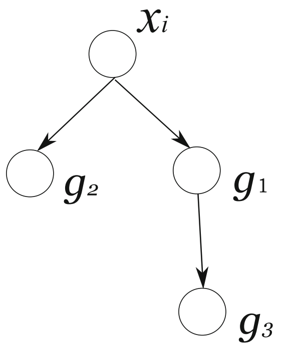


Fig. 2. Case 1

on only guess inputs and independent from actual inputs. (v may be a gate and may be a guess input.) Consider that an assignment to actual inputs and guess inputs is given. Then, if the value of the output of v blocks g_1 by the assignment, then the output of C must be 0, since, if the output of C is 1, then the value of the Boolean function which is computed by C becomes independent from x_i, which contradicts that C computes Parity$_n$. (Note that the output of C can be 0. The difference is from the definition of nondeterministic circuits.) We use the fact above and reconstruct C as follows.

Let c be a constant 0 or 1 such that if the output of v is c, then g_1 is blocked. We fix the input of g_1 from v to $\neg c$ and eliminate g_1. We prepare a new output gate g_2 and connect the two inputs of g_2 from the old output gate and v. g_2 is labeled by a Boolean function in U_2 so that the output of g_2 is 1 iff the input from the old output gate is 1 and the input from v is $\neg c$. Let C' be the reconstructed circuit. See Figure 3.

In the reconstruction, we eliminated one gate (g_1) and added one gate (g_2). Thus, the size of C' equals the size of C. In C', if the output of v is c, then the output of C' becomes 0 by g_2. If the output of v is $\neg c$, then the output of C' equals the output of the old output gate and g_1 has been correctly eliminated since we fixed the input of g_1 from v to $\neg c$ in the reconstruction. Thus, C and C' compute a same Boolean function. We repeat such reconstruction until the reconstructed circuit satisfies the condition of Case 1. The repetition must be ended, since one repetition increases continuous gates whose one input is dependent on only guess inputs (i.e., g_2) at the output. Note that g_1 is not included in the continuous gates, since the output of C must depend on at least two actual inputs.

Thus, we can eliminate at least 3 gates for all cases and the circuit come to compute Parity$_{n-1}$ or $\neg$Parity$_{n-1}$.

Lemma 1.
$$size^{\mathrm{ndc}}(\mathrm{Parity}_{n-1}) = size^{\mathrm{ndc}}(\neg\mathrm{Parity}_{n-1}).$$

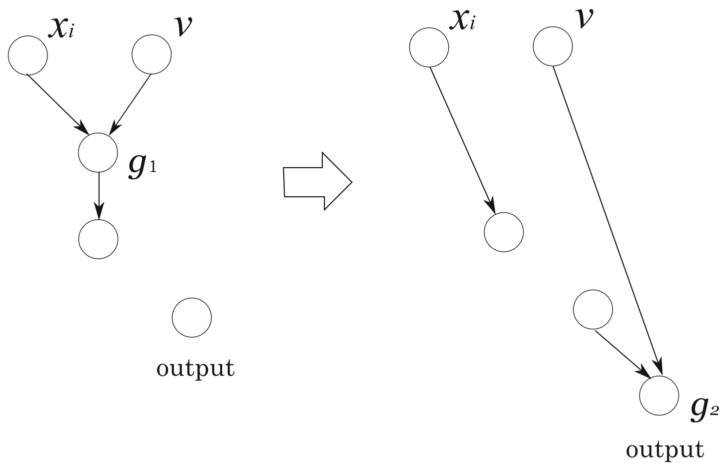


Fig. 3. Case 2

Proof. Let C' be a nondeterministic U_2-circuit computing Parity$_{n-1}$. For nondeterministic circuits, notice that negating the output gate in C' does not give a circuit computing $\neg$Parity$_{n-1}$. We negate an arbitrary actual input x_i, $1 \leq i \leq n$, in C' and obtain a nondeterministic U_2-circuit which computes $\neg$Parity$_{n-1}$ and has the same size to C', which can be done by relabeling each Boolean function of all gates which are connected from x_i. □

Therefore,

$$\begin{aligned} size^{\mathrm{ndc}}(\mathrm{Parity}_n) &\geq size^{\mathrm{ndc}}(\mathrm{Parity}_{n-1}) + 3 \\ &\vdots \\ &\geq 3(n-1). \end{aligned}$$

Thus, the theorem holds. □

4 Discussions

In this paper, we proved a $3(n-1)$ lower bound for the size of nondeterministic U_2-circuits computing the parity function. To the best of our knowledge, this is the first nontrivial lower bound for the size of nondeterministic circuits with unlimited nondeterminism for an explicit Boolean function. In this section, as one of motivations to prove lower bounds for the size of nondeterministic circuits, we discuss an approach to proving lower bounds for the size of deterministic circuits via lower bounds for the size of nondeterministic restricted circuits.

It is known that the Tseitin transformation [8] converts an arbitrary Boolean circuit to a CNF formula. We restate the Tseitin transformation as the form of the following theorem.

Theorem 3. *Any B_2-circuit of n inputs and size s can be converted to a nondeterministic 3CNF formula of n actual inputs, s guess inputs, and size $O(s)$.*

Proof. We prepare one guess input for the output of each gate in the B_2-circuit, and use the Tseitin transformation. □

Thus, if we hope to prove a nonlinear lower bound for the size of deterministic general circuits, (which is a major open problem in circuit complexity theory,) then it is enough to prove a nonlinear lower bound for the size of nondeterministic circuits in the theorem. The nondeterministic circuit in Theorem 3 is a constant depth circuit with depth two and some restrictions. In this paper, we saw that, for U_2-circuits, a known proof technique (i.e., the gate elimination method) for deterministic circuits is applicable to the nondeterministic case. It remains future work whether many known ideas or techniques for constant depth circuits are applicable to nondeterministic circuits.

The basic idea of the proof of Theorem 3 and the Tseitin transformation can be widely applied. We show another example in which guess inputs are prepared for a part of gates in the circuit.

Theorem 4. *Any B_2-circuit of n inputs, size $O(n)$ and depth $O(\log n)$ can be converted to a nondeterministic formula of n actual inputs, $O(n/\log\log n)$ guess inputs, and size $O(n^{1+\epsilon})$, where $\epsilon > 0$ is an arbitrary small constant.*

Proof. Let C be such a B_2-circuit. It is known that we can find $O(n/\log\log n)$ edges in C whose removal yields a circuit of depth at most $\epsilon \log n$ ([9], Section 14.4.3 of [1]). We prepare $O(n/\log\log n)$ guess inputs for the edges, and one guess input for the output of C. Consider that an assignment to actual inputs and guess inputs is given. It can be checked whether the value of each guess input corresponds to correct computation by $O(n/\log\log n)$ nondeterministic formulas of size n^ϵ, since the depth is at most $\epsilon \log n$. We construct a nondeterministic formula so that it outputs 1 iff all guess inputs are correct and the guess input which corresponds to the output of C is 1. □

For the case that the number of guess inputs is limited, there is a known lower bound for the size of nondeterministic formulas [5].

References

1. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009)
2. Blum, N.: A boolean function requiring 3n network size. Theor. Comput. Sci. **28**, 337–345 (1984)
3. Demenkov, E., Kulikov, A.S.: An elementary proof of a $3n-o(n)$ lower bound on the circuit complexity of affine dispersers. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 256–265. Springer, Heidelberg (2011)
4. Iwama, K., Morizumi, H.: An explicit lower bound of 5n - o(n) for boolean circuits. In: Proc. of MFCS, pp. 353–364 (2002)

5. Klauck, H.: Lower bounds for computation with limited nondeterminism. In: Proc. of CCC, pp. 141–152 (1998)
6. Lachish, O., Raz, R.: Explicit lower bound of 4.5n - o(n) for boolean circuits. In: Proc. of STOC, pp. 399–408 (2001)
7. Schnorr, C.: Zwei lineare untere schranken für die komplexität boolescher funktionen. Computing **13**(2), 155–171 (1974)
8. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic, pp. 115–125 (1968)
9. Valiant, L.G.: Graph-theoretic properties in computational complexity. J. Comput. Syst. Sci. **13**(3), 278–285 (1976)

Computing Minimum Dilation Spanning Trees in Geometric Graphs

Aléx F. Brandt, Miguel F.A. de M. Gaiowski, Pedro J. de Rezende[✉], and Cid C. de Souza

Institute of Computing, University of Campinas, Campinas, Brazil
rezende@ic.unicamp.br

Abstract. Let P be a set of points in the plane and $G(P)$ be the associated geometric graph. Let T be a spanning tree of $G(P)$. The dilation of a pair of points i and j of P in T is the ratio between the length of the path between i and j in T and their Euclidean distance. The dilation of T is the maximum dilation among all pairs of points in P. The *minimum dilation spanning tree problem* (MDSTP) asks for a tree with minimum dilation. So far, no exact algorithm has been proposed in the literature to compute optimal solutions to the MDSTP. This paper aims at filling this gap. To this end, we developed an algorithm that combines an integer programming model, a geometric preprocessing and an efficient heuristic for the MDSTP. We report on computational tests in which, for the first time, instances of up to 20 points have been solved to proven optimality.

Keywords: Minimum dilation · Stretch factor · Geometric Graphs

1 Introduction

In many practical applications, one is faced with problems requiring the construction of a low cost network connecting a set of locations that satisfy certain quality requirements. Depending on how the cost and the quality are measured, different optimization problems emerge.

In this paper, we consider one such case, known as the *minimum dilation spanning tree problem*. Let P be a set of n points in the plane. For every $u, v \in P$, d_{uv} denotes the Euclidean distance between u and v. The *geometric graph*, $G(P) = (P, E)$, is the weighted complete graph whose vertex-set is P and whose edge weights are given by the Euclidean distances between the corresponding endpoints. Consider a spanning subgraph H of $G(P)$. Let u, v be two vertices in H. The *distance* on H between u, v is defined as the length of a shortest path $\pi_H(u, v)$ connecting u to v in H, denoted $|\pi_H(u, v)|$, or infinite when v is unreachable from u. As usual, the length of a path is defined as the sum of the weights of its edges. The *dilation* (also known as *stretch factor* or *distortion* [1])

Research supported by grants: CNPq (302804/2010-2, 477692/2012-5, 311140/2014-9, 139107/2012-6), FAPESP (2015/08734-9, 2012/17965-6), and Faepex.

of H is defined as $\rho(H) = \max_{u,v \in P} |\pi_H(u,v)|/d_{uv}$, see [2]. Since $\rho(G(P)) = 1$, the interesting problems arise when a proper subgraph of $G(P)$ is sought.[1] An often considered situation is when we limit to some constant k the number of edges that can be used to build the network while seeking to obtain the minimum dilation. As this problem only makes sense when the resulting subgraph is connected, the smallest value for k is $n - 1$, restricting the plausible subgraphs to the spanning trees of $G(P)$. This gives rise to the *minimum dilation spanning tree problem* (MDSTP), which is known to be $\mathbb{NP}$-hard [3]. Figure 1 shows an example. Moreover, thus far, no $o(n)$ approximate algorithm for the MDSTP is known [3].

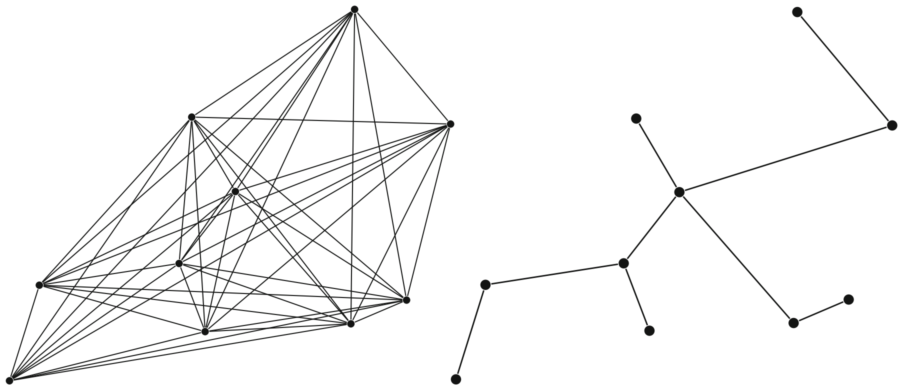


Fig. 1. A geometric graph $G(P, E)$ and its minimum dilation spanning tree

This paper focuses on the design of an exact algorithm for the MDSTP, which is effective in practice for applications that require actual optimal solutions.

Motivation. The investigation of minimum dilation problems is prompted by their applications to robotics, network topology design and many others (see [1,4]). While the literature does not yet include exact algorithms for the MDSTP, we believe that producing optimal solutions may enable us to identify geometric properties that can lead to algorithms that are more effective in practice. Studying the MDSTP may reveal relevant information for solving similar problem for other classes of graphs.

Related Work. Experimental works published on dilation problems attempt to find, for a given t, (sub)optimal t-spanners minimizing the total edge weight, vertex degrees, the number of edges for a fixed dilation, the spanner diameter or the number of crossings ([5,6]). In contrast to what is done here, none of these look for a subgraph that minimizes the dilation *within a given family.*

Some important bounds for the minimum dilation of spanning trees are the worst case lower bound $\Omega(n)$ achieved by the vertex set of a regular n-gon [1]

[1] The reader will find ample material on the related concept of t-spanners in [1,2].

and the upper bound $n - 1$ guaranteed by the minimum weight spanning tree [4]. Also, Cheong, Haverkort and Lee [3] (Klein and Kutz [7]) presented a set P of five (seven) points in the plane for which any MDST has self-intersections.

Our Contribution. This work presents the first exact algorithm for the MDSTP. The method employs a mixed integer linear programming (MILP) model that may be computed by state-of-the-art solvers. However, as the MILP formulation, can easily grow to an unmanageable size, we developed a preprocessing phase that exploits the geometry of the problem to significantly reduce the model. Moreover, while as the preprocessing requires the knowledge of good viable solutions, we designed a metaheuristic that yields high quality solutions. As a convenient byproduct, these fine primal bounds also allow the MILP solver to do early pruning of the enumeration tree during the branch-and-bound algorithm. As a result of this strategy, we were able to find provably optimal solutions for instances of the MDSTP with up to 20 points, which is unprecedented and, from our experience, extremely challenging as the largest instances took 30 minutes of CPU time. To enable future comparisons, we also make available our benchmark of thousands of instances, as well as scripts for generating pseudo-random point sets based on either fixed or image-based distributions.

Text Organization. In Section 2 a MILP formulation for the problem is given. Section 3 shows how any known primal bounds can be used in preprocessing routines that, *a priori*, retain or discard edges from participating in an optimal solution or from being in a path joining two vertices in such a solution. These routines strongly influence the performance of our method since, ultimately, they reduce the size of the MILP model. A GRASP based heuristic that we developed for the MDSTP is presented in Section 4. The computational tests are reported in Section 5 and Section 6 discusses future research directions.

2 A MILP Model for the MDSTP

The MILP formulation aims at building paths with limited dilation between all pairs of vertices in such a way that the union of these paths is a spanning tree. This dilation is then minimized in the objective function.

The characterization of the paths is achieved by means of a multicommodity flow network model in which there is a special unitary flow between each distinct pair of vertices. This makes it possible to obtain the length of a particular path and to bound its dilation. The flow model is defined on the directed graph $DG(P) = (P, A)$ obtained from $G(P)$ by replacing each of its edges by two directed arcs. A binary variable x_{ij}^{ab} is introduced corresponding to the flow of the commodity ab (the one associated to the path from a to b) through the directed arc (i, j). Naturally, x_{ij}^{ab} is set to one if and only if the arc (i, j) belongs to the path from a to b in an optimal solution. It is easy to see that a full characterization of a spanning tree can be determined from the subset of variables of the form x_{ij}^{ij} since, once one of these is set to 1, the (unique) path from i to j is comprised solely by the edge $\{i, j\}$. The last element of the model is a continuous variable ρ that represents the optimum. The formulation reads:

$$\min \quad \rho \tag{1}$$

$$\sum_{i \in V \setminus \{a\}} x_{ia}^{ab} - \sum_{i \in V \setminus \{a\}} x_{ai}^{ab} = -1 \qquad \forall a, b \in V \tag{2}$$

$$\sum_{i \in V \setminus \{j\}} x_{ij}^{ab} - \sum_{i \in V \setminus \{j\}} x_{ji}^{ab} = 0 \qquad \forall j \in V \setminus \{a, b\},\ \forall a, b \in V \tag{3}$$

$$\sum_{i \in V \setminus \{b\}} x_{ib}^{ab} - \sum_{i \in V \setminus \{b\}} x_{bi}^{ab} = 1 \qquad \forall a, b \in V \tag{4}$$

$$x_{ij}^{ab} + x_{ji}^{ab} - x_{ij}^{ij} \leq 0 \qquad \forall \{i, j\} \in E,\ \forall a, b \in V \tag{5}$$

$$\sum_{\{i,j\} \in E} x_{ij}^{ij} = |V| - 1 \tag{6}$$

$$\sum_{(i,j) \in A} \frac{d_{ij}}{d_{ab}} x_{ij}^{ab} - \rho \leq 0 \qquad \forall a, b \in V \tag{7}$$

$$x \in \mathbb{B}^{|A| \cdot |E|} \tag{8}$$

The objective function (1) simply asks for a solution of minimum dilation. For a given pair of points a and b, (2), (3) and (4) impose that the variables x^{ab} describe a path from a to b in the final solution. Once we have the description of the paths between all vertex pairs it remains to ensure that we have a connected structure with only $n - 1$ edges, implying that the resulting subgraph is acyclic. This is done by two sets of constraints: (5) which does the coupling of the arcs in the paths and the edges in the tree and (6) that restricts the number of edges in the solution. Finally, (7) constrains the dilation of every path to be no larger than ρ. As the objective function minimizes ρ, it is clear that, in the solution, this variable is equal to the maximum dilation among all pairs of vertices.

This formulation is based on the *extended multicommodity flow* model described in [8] for the spanning tree problem. An important feature of that model is that it gives a complete description of the convex hull of the integer solutions of the spanning tree problem, called the *spanning tree polytope*. Besides, the number of constraints and variables in this linear system is polynomial in the size of the input graph, i.e., the model is compact. Basically, in this formulation one chooses an arbitrary vertex as the root of the tree, in our case vertex zero, and flows a especial commodity to each other vertex. In reality, the original coupling constraints are slightly more complex than the one presented here as they also ensure that every pair of commodities flows in the same direction in each arc. This is done by enforcing the inequalities $x_{ij}^{0k} + x_{ji}^{0k'} - x_{ij}^{ij} \leq 0$, for all k, k' in $V \setminus \{0\}$ and $\{i, j\}$ in E. Our formulation relinquishes this full description in favor of a more compact and still correct model by using (5).

One may have noticed that the model (1)–(8) already has an enormous number of constraints and variables, both of which are $\Theta(n^4)$. The massive size of the model severely affects the performance of MILP solvers. It is worth noting that other options regarding the coupling constraints exist, some of which leading to tighter linear relaxations. However, these alternative formulations are even larger than the one given here. Besides, in early experiments where they were used, we observed little or no actual gain in the dual bounds – the professed benefit of using such relaxations – despite a noticeable increase in execution times.

In the next section, we discuss how a known primal (upper) bound can be used to accelerate the search for a proven optimal solution. Essentially, this is done by exploiting geometric properties that allow us to fix some of the variables of our MILP model. As we mentioned in the previous paragraph, the size of the formulation can be a bottleneck for the direct usage of our method. Therefore, the effectiveness of the preprocessing is a key step for the success of the approach.

3 Preprocessing

Suppose that a feasible solution for the MDSTP has been computed by some heuristic, whose dilation is ρ'. Now, based on geometric properties, we devise routines that allow the identification of edges of the geometric graph $G(P)$ that must or must not be in any tree with dilation less than or equal to ρ'. Alternatively, we may be able to reach the weaker but relevant conclusion that an edge may be required to or forbidden from being part of the path between a specific pair of vertices in any solution with dilation not exceeding ρ'. As said earlier, these findings allow us to reduce the size of the MILP model as discussed below.

Consider the first case, where we conclude that an edge $\{i, j\}$ is or is not part of a tree with dilation bounded from above by ρ'. The x_{ij}^{ij} variable is either set to one or to zero. Due to (5), the latter case also forces all the $O(n^2)$ path variables associated to arc (i, j) to be set to zero. On the other hand, if $\{i, j\}$ is forced to be in the optimal tree, all the $O(n^2)$ path variables x_{kl}^{ij} for $(k, l) \in A \setminus (i, j)$ are required to take value zero by (2)–(4). Of course, in this case, the $O(n)$ equations describing the path from i to j can also be removed from the model.

Let us focus now on the case where we conclude that a given arc (i, j) is or is not in the path from a to b, with $\{i, j\} \neq \{a, b\}$. When this arc is identified as a required part of the path, not only the variable x_{ij}^{ab} but also the corresponding edge variable x_{ij}^{ij} is set to one. As discussed earlier, the latter assignment propagates the setting of variables to arc variables (a, b). On the other hand, if (i, j) is not in the path from a to b and no other information is at hand, only the flow variable can be set to zero. Thus, in contrast to the previous analysis, there is no direct propagation of variable assignments. But, as we shall see, depending on the distribution of the points of P on the plane and on the primal bound, a large number of variables may be set, considerably reducing the model size.

Below, we describe three routines that trigger the setting of some variable. We assume that a, b, u and v are four distinct points in P, and ρ' is an upper bound for the minimum dilation of the spanning trees of $G(P)$.

The Ellipse-based Elimination Routine. Let T be a spanning tree and suppose u is a vertex on the path $\pi_T(a, b)$, i.e., $\pi_T(a, b) = \pi_T(a, u) \cup \pi_T(u, b)$. The dilation of the pair $\{a, b\}$ in T is given by $\rho_T(a, b) = \frac{|\pi_T(a,b)|}{d_{ab}}$. From the triangle inequality, we have $\pi_T(a,u) \geq d_{au}$ and $\pi_T(u,b) \geq d_{ub}$ and $\rho_T(a,b) \geq \frac{d_{au}+d_{ub}}{d_{ab}}$. Hence, if T has dilation at most ρ', the following constraint must hold:

$$\frac{d_{au}+d_{ub}}{d_{ab}} \leq \rho_T(a,b) \leq \rho' \ , \tag{9}$$

which means that u must lie on or inside the ellipse with foci a and b having a major axis of length $\rho' \cdot d_{ab}$. In other words, given an upper bound ρ' for the dilation and two vertices a and b, only the points in the ellipse defined by (9) can be on the path joining a and b in a tree with dilation not exceeding ρ'. A similar argument appears in [7], where the MDSTP is proven to be $\mathbb{NP}$-hard.

Furthermore, all $O(n)$ arcs incoming to or outgoing from any discarded vertex are precluded from being part of the path between a and b and may, therefore, be removed from the model, i.e., the corresponding variables may be set to zero.

The Long Arc Elimination Routine. Once the vertices inside the ellipse with foci a and b have been identified, it may still be possible to discard arcs joining pairs of them from the path from a to b. Consider two such vertices, say, u and v, with $\{u,v\} \neq \{a,b\}$. If the arc (u,v) is in a solution with dilation bounded by ρ', the triangle inequality requires that the following inequality holds:

$$\frac{d_{au}+d_{uv}+d_{vb}}{d_{ab}} \leq \rho_T(a,b) \leq \rho' \ . \tag{10}$$

Clearly, when the above inequality is not satisfied, the arc (u,v) cannot belong to the path joining a and b in a solution with dilation limited to ρ'. Since the flow is directed from a to b, using the arc (u,v) is not the same as including (v,u) in the path. So, (10) must be applied to every arc of A, i.e., both directions of the respective edge have to be considered.

This routine can be viewed as an extension of the previous one, in the sense that it evaluates subpaths with two intermediate vertices instead of just one. Longer subpaths could be analyzed in almost the same way. However, the complexity of such tasks would increase quickly rendering them impractical.

The Edge Fixing Routine. A very handy situation occurs when an ellipse with foci a and b contains no other points of P in its interior. We will refer to it as an "empty" ellipse. Of course, if the ellipse's major axis has length $\rho' \cdot d_{ab}$ the only way to connect a and b in a tree with dilation no greater than ρ' is through the edge ab. In other words, this edge is fixed (i.e., must belong to that tree).

As edges are ascertained to be part of the tree, the subgraph they induce is a forest containing connected components of two or more vertices. Two types of reductions of the MILP model can then occur. Firstly, any edge between two vertices on the same component that is not already fixed corresponds to a variable that can be set to zero; otherwise a cycle would be formed. The same holds for the path variables associated to the arcs corresponding to those edges. Similarly, every two vertices in a connected component induced by fixed edges have their paths already defined. Hence, the flow variables on this route can be set to one while, at the same time, all other flow variables associated to arcs on alternative paths can be set to zero.

4 A GRASP for the MDSTP

Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic widely applied to combinatorial optimization problems. Since it is reported to produce high quality solutions within a short computing time [9], we employ it to find good primal bounds for the MDSTP. We assume that the reader is familiar with the workings of a basic GRASP algorithm. So, to explain its use in the context of the MDSTP, just the construction and the local search phases are described.

Firstly, a solution is built by randomizing Prim's algorithm [10] for building a minimum spanning tree (MST) from the (complete) geometric graph $G(P)$. This randomization simply expands the purely greedy choice of a vertex to be added to a partially built MST to a uniformly distributed selection of a vertex from a *Restricted Candidate List* (RCL) of vertices. This list is constructed in the following way. Let d_m (D_m) denote the length of the shortest (longest) edge connecting the current (incomplete) MST to the vertices that are not yet part of this MST. Given an α in the interval $[0, 1]$ we take into the RCL each vertex of $G(P)$ whose distance to the current (incomplete) MST is within the range $[d_m, (1 - \alpha)d_m + \alpha D_m]$. Clearly, the smaller α is, the shorter the RCL will be, and the choice of $\alpha = 0$ leads to a purely greedy algorithm, while $\alpha = 1$ turns it into a randomized one. In our implementation, at each iteration of the GRASP's construction phase, a new value for α is chosen from the set $\{0.0, 0.1, \ldots, 1.0\}$ with equal probability, leading to a likely diverse sequence of viable solutions.

Secondly, in the local search phase, attempts are made to improve known solutions. For the sake of efficiency, the perturbations defining the neighborhood of a solution should be quickly computable. That is why, for the MDSTP the naive approach of merely replacing a tree edge for another is undesirable. After all, that might drastically alter the dilation between unpredictably many pairs of vertices in an arbitrary way and even the tree's. For this reason, we devised a controlled method for navigating within a neighborhood of a given solution, called *Triangle*. Let u and v two non-adjacent vertices in a spanning tree T and z be a vertex in the unique path from u to v in T. Suppose that a and b are the vertices adjacent to z in this path. Denote by T_A (T_B) the subtree containing vertex a (b) and obtained from T by removing the edge (a, z) ((b, z)). Replacing (a, z) by (a, b) changes T in such a way that the dilation of the pair $\{u, v\}$ decreases. On the other hand, by the triangle inequality, we know exactly which pairs of vertices may have their dilation worsened. To see this, define A and B as the set of vertices in T_A and T_B, respectively, and $Z = P - (A \cup B)$. Clearly, the pairs of vertices from $A \times Z$ are the only ones for which the dilation needs to be recalculated. Although there may still be $O(n^2)$ such pairs, in practice, significant reductions in computing times were observed when we applied this method. A homologous analysis holds if we replace (b, z) by (b, a), instead.

To overcome the drawback of the *Triangle* method, which generates only two neighbor solutions, we devised an extended version called *Path*. This new neighborhood iterates the *Triangle* local search through consecutive triples of vertices along a path joining two vertices with maximum dilation. *Path* achieves a compromise between neighborhood size and the time needed to explore it. The attentive reader will realize that, given a path of length k, the size of the *Path* neighborhood is $2k$, instead of only 2 for *Triangle*. Although exploring more solutions is, of course, more expensive, our tests showed that the slight increase in running time was compensated by the gain in quality of the yielded solutions.

Two strategies to halt local search are commonly applied: *first improvement* and *best improvement* (see [11]). Experimentation lead us to chose the former, since best improvement proved too time consuming for the minute benefit it generates after its inherently exhaustive search.

Our implementation also applies *path relinking* [11] to a pool of *elite solutions*. This pool is created throughout the iterations to store the best known solutions, cost wise, as well as those whose costs are within a certain threshold of the best primal bound. Moreover, diversity, measured according to the number of elements in the symmetric difference between two solutions, is also favored. To describe the path relinking process employed here, given two elite solutions S and S', we need to outline how to iteratively perturb S into S', giving us a path in the search space within which a new improved solution is likely to be found [11]. Suppose that S and S' are the starting and target trees, respectively. The path relinking S and S' is traversed as follows. An edge $e \in S' - S$ is added to S creating a cycle. The removal of another edge in $S - S'$ from this cycle leads to a new tree which is a step closer to S'. Clearly, k such iterations create k new trees among which there may be one with lower dilation than both S and S'.

Lastly, notice that the size of the elite set influences the overall performance of GRASP. Too many solutions makes the algorithm run too slow, whereas too few solutions may thwart its ability to generate any improvement. In our tests, we found that a good strategy is to store all the solutions found during the GRASP iterations that resulted in an update of the best known solution up to that point. In our implementation, *path relinking* is executed after 1000 iterations of the loop construction phase/local search phase have been completed.

We conclude this section reiterating that, in this paper, GRASP must be viewed solely as a tool to help in attaining optimality. An in depth investigation on the theme of heuristics, in its own right, for the MDSTP is deserving of future attention.

5 Computational Results

This section discusses the experiments we carried out. We begin by describing the characteristics of our benchmark and of the computational environment. We continue the section with the presentation and analyses of the results obtained.

Instances. We generated a benchmark consisting of instances comprised of uniformly distributed points on a 10×10 square. The coordinates of the points and the distances between two points were both rounded to six decimal places in order to avoid arithmetic pitfalls and to facilitate future comparisons. To promote the fulfillment of the latter goal, the entire benchmark is made available for download at `www.ic.unicamp.br/~cid/Problem-instances/Dilation`. The set consists of 30 instances for each number of points from the sequence $10, 12, 14, 16, 18, 20$.

Computing Environment. The results reported here were obtained on identical machines featuring: Intel® Xeon® CPU E3-1230 V2 @ 3.30GHz (4 cores and 8 threads) processors; 32GB of RAM; running OS Ubuntu 12.04; using a g++ 4.6.3 compiler and the MILP solver IBM® ILOG® CPLEX® Optimization Studio 12.5.1. The solver was allowed to use all 8 threads simultaneously and to run for at most 30 minutes. Most CPLEX parameters were left at default values, although some changes are noteworthy: *(i)* the use of *Traditional Branch & Cut* search method instead of *Dynamic Search*; *(ii)* higher branching priorities were enforced on the edge variables in detriment of the arc ones; and *(iii)* the relative and absolute gaps were set to 10^{-7}.

Results. We now discuss the relevant tests we ran to assess the efficiency of our algorithm. Since the application of our approach depends on the computation of good primal bounds, our first analysis focus on the performance of the GRASP metaheuristic. To evaluate that, we present in Table 1 statistics relative to the execution of the heuristic on instances for which the optimum was found. The first column of this table displays the number of points per instance. Two groups of four columns follow reporting minimum, average, standard deviation and maximum values, respectively for the gap relative to the optima and for the running times (in milliseconds).

As seen from these results, GRASP solutions are of very high quality, at least for the instance sizes considered in our tests. Despite some fluctuation in the gaps, one can perceive a slight loss in quality as the sizes of the instances increase. Since the main purpose of this work is to compute optimal solutions and, as we will see, we are still unable to prove optimality for instances of more than 20 points, we decided not to invest more time in improving the GRASP heuristic at this time. Clearly, since the MDSTP is $\mathbb{NP}$-hard, pursuing this goal would be an interesting investigation in its own right. But, for now, let us just remark that the CPU times spent by the heuristic, as is, are insignificant on our benchmark.

Next, we analyze the roles played by the primal information and by the preprocessing in the computation of optimal solutions. To accomplish this task, five different variants of the MILP solver were tested: *(i) None*: corresponds to the execution of CPLEX with default options and the complete model given in Section 2; *(ii) Sol*: same as in *(i)*, but with CPLEX also fed with the upper bound as well as an optimal solution; *(iii) PreP*: same as in *(i)* except that, in this case, the model is reduced through the preprocessing discussed in Section 3; *(iv) UB+PreP*: same as in *(iii)* but, besides the preprocessing, the optimum is also given as an upper bound to CPLEX; and, finally, *(v) Sol+PreP*: same as in *(iv)* but adding as the part of the input an optimal solution.

At first glance, the usage of optimal solutions and bounds in these initial tests may sound strange. However, this information was given instead of GRASP outcomes because, as said before, the heuristic results present some oscillation even when equal sized instances are compared. Besides, the quality of the heuristic solution was also seen to deteriorate slightly as the instance size increases. By using optimal information, we intend to minimize the effects of these phenomena which, otherwise, could lead to biased conclusions. The influence of the degradation of the quality of the primal information in the computation of optimal solutions is evaluated later in this section.

The strategy to assess the contributions of the present work should now be clear from the choice of these variants. An obvious way to speed up the running time of an enumeration algorithm is to provide as input a primal bound and a solution with cost equal to this bound. A comparison between the performances of *None* and *Sol* answers the question on how much is gained by applying this standard technique to the MDSTP.

Notice that state-of-the art solvers like CPLEX, are equipped with powerful algebraic preprocessing routines that, when fed with primal information, can dramatically reduce the MILP formulation. Hence, to evaluate the importance of *our* geometric preprocessing, we tested the three remaining variants, all of which include this preprocessing. In *PreP* we constrain ourselves to the model

Table 1. Statistics for the GRASP metaheuristic

Size	Dilation Gap to Opt. Sols (%)				Grasp Runtime (ms)			
	Min	Average	Std Dev.	Max	Min	Average	Std Dev.	Max
10	0	0,07	0,41	2,23	22	44,3	8,8	60
12	0	0,14	0,76	4,15	48	65,8	10,9	90
14	0	0,05	0,22	1,12	65	93,7	17,7	140
16	0	0,01	0,06	0,31	100	125,5	14,4	157
18	0	0,38	1,47	7,38	113	148,2	23,3	198
20	0	0,47	1,62	7,37	147	186,9	16,6	230

Table 2. Statistics for five variations of the MILP solver

Size	Method	# Opt Sols	Avg Exec Time (s)	# Feas Sols	% Avg Gap	Wins	Avg Fxd Edges	Avg free vars (%)
10	*Sol+PreP*	30	0.2	0	–	23	4.0	32.3
	UB+PreP	30	0.2	0	–	7	4.0	32.3
	PreP	30	0.3	0	–	0	4.0	32.3
	Sol	30	1.5	0	–	0	0.0	100.0
	None	30	5.0	0	–	0	0.0	100.0
12	*Sol+PreP*	30	1.5	0	–	29	4.5	35.4
	UB+PreP	30	1.8	0	–	1	4.5	35.4
	PreP	30	1.9	0	–	0	4.5	35.4
	Sol	30	22.5	0	–	0	0.0	100.0
	None	30	83.1	0	–	0	0.0	100.0
14	*Sol+PreP*	30	4.9	0	–	27	4.4	39.6
	UB+PreP	30	7.6	0	–	3	4.4	39.6
	PreP	30	7.7	0	–	0	4.4	39.6
	Sol	30	121.0	0	–	0	0.0	100.0
	None	27	588.0	3	4.3	0	0.0	100.0
16	*Sol+PreP*	30	42.3	0	–	24	4.1	45.8
	UB+PreP	30	54.6	0	–	4	4.1	45.8
	PreP	30	64.3	0	–	2	4.1	45.8
	Sol	20	387.0	10	25.6	0	0.0	100.0
	None	3	1661.0	27	39.7	0	0.0	100.0
18	*Sol+PreP*	27	222.1	3	43.5	19	4.7	46.6
	UB+PreP	26	211.1	2	27.4	5	4.7	46.6
	PreP	26	208.4	3	38.8	3	4.7	46.6
	Sol	12	647.4	18	37.8	0	0.0	100.0
	None	0	–	5	72.1	0	0.0	100.0
20	*Sol+PreP*	21	261.5	9	44.7	12	4.1	49.1
	UB+PreP	23	466.7	0	–	5	4.1	49.1
	PreP	22	400.3	4	46.5	6	4.1	49.1
	Sol	4	1328.6	26	41.3	0	0.0	100.0
	None	0	–	13	95.2	0	0.0	100.0

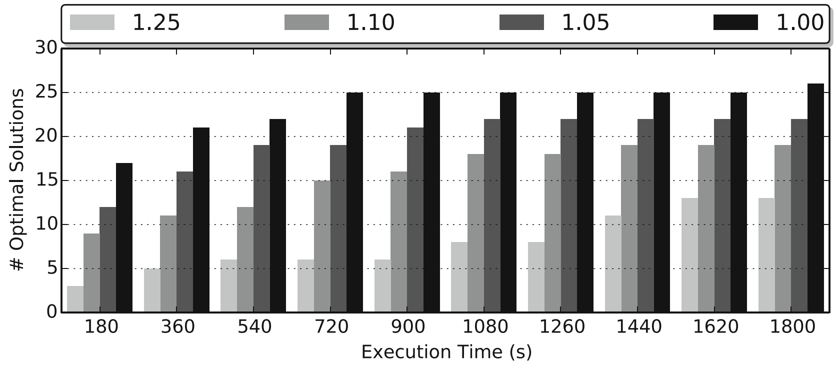


Fig. 2. Variation of the number of optima found as the primal bound deteriorates

reduction, leaving out the primal information. To measure the effect of adding the latter, piece by piece, we first provide only the upper bound in *UB+PreP* and, finally, the complete information in *Sol+PreP*.

The results obtained by the MILP variants are summarized in Table 2. For each instance size displayed in the first column, there are five rows, each corresponding to one variant, as shown in the second column. The third column exhibits the number of optima obtained and the next column shows the average execution time achieved by the corresponding variant, over the instances for which it reached a proven optimum. The fifth column shows the number of instances for which optimality was not proved but that the solver found a feasible solution. In the following column, the value corresponds to the average gaps of the latter solutions. The seventh column gives the total number of instances where the variant outperformed the remaining ones. The eighth and ninth columns allow an assessment of the model reductions resulting from the geometric preprocessing. They include, first, reports of the average number of variables selected by the edge fixing routine, followed by the average percentage of variables that remain in the model after all preprocessing routines have been executed. Percentages are calculated in terms of the total number of variables in the complete MILP model.

Consider the 90 instances of the three largest sizes. Variant *None* could only solve 3 of them to optimality, suggesting that the MILP model, alone, is not very useful. A substantial gain was obtained when the primal information was made available in *Sol* as 36 instances were now solved to optimality. Finally, the contribution of the preprocessing can be fully appreciated when we compare these numbers to those of the three last variants. One can see that any of them solved at least twice as many instances as the other variants where no geometric preprocessing was performed.

Once we have established the relevance of our geometric preprocessing, the 3 variants that use it should still be compared. The numbers of proven optima were 48, 49 and 48 for *Sol+PreP*, *UB+PreP* and *PreP*, respectively. This might suggest a small advantage for *UB+PreP*. However, a closer analysis of the data shows that the number of feasible solutions (including optimal ones) obtained by these variants were 180, 171 and 175, respectively, leaving the *UB+PreP* variant behind the other two and *Sol+PreP* in a better position. The advantage of *Sol+PreP* becomes even more apparent when we turn our attention to the overall performance of the algorithm, including the evaluation of the running times. As can be seen on column *"Wins"*, *Sol+PreP* has a total of 134 instances, compared to 25 of *UB+PreP* and 11 of *PreP*. Concerning the execution times (fourth column), the benefit of employing the routines in Section 3 speaks for itself: just contrast *None* vs. *PreP* and *Sol* vs. *Sol+PreP*. Therefore, one concludes that the geometric preprocessing is crucial for the success of the algorithm.

The last two columns of Table 2 confirm that the preprocessing is indeed a powerful tool for model reduction. In this context, the primal information is fundamental since, without an upper bound, no preprocessing could have been done, in the first place. Moreover, the knowledge of a solution with cost equal to this bound accelerates the convergence of the algorithm, emphasizing the importance of obtaining good primal solutions, as done by our GRASP.

Since GRASP is a heuristic, we cannot guarantee the quality of the primal information yielded by the procedure. Therefore, another relevant issue to consider is how much the overall algorithm's performance deteriorates as the upper bound used for preprocessing and given as input to the solver worsens. This analysis can be carried out based on the bar graph shown in Fig. 2. The data refer to the executions of the MILP variant *UB+PreP* over the 30 instances of size 18 with the upper bounds given by the dilation of the optimal solution multiplied by 1.00, 1.05, 1.10 and 1.25. The bars reflect the cumulative number of optimal solutions found for each of these multiplying factors and measured at every 180 seconds for up to half an hour. For example, after 720 seconds, the number of instances solved was 25, 19, 15 and 6 for each of the aforementioned multiplying factors. From this graph, it is quite clear that the algorithm's performance declines rapidly as the upper bound decreases. From the previous analyses, this effect is likely to be a repercussion of the loss of efficiency of the preprocessing.

6 Future Directions

Some issues are currently being investigated to improve the method presented here, including: *(i)* the customization of the choice of the branch variable according to geometric properties; *(ii)* the use of GRASP and geometric preprocessing in all nodes of the enumeration tree; and, *(iii)* the use of other MILP formulations.

References

1. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, New York (2007)
2. Peleg, D., Schäfer, A.A.: Graph spanners. Journal of Graph Theory **13**(1), 99–116 (1989)
3. Cheong, O., Haverkort, H., Lee, M.: Computing a minimum-dilation spanning tree is NP-hard. Comput. Geom. Theory Appl. **41**(3), 188–205 (2008)
4. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Smid, M., Vigneron, A.: Sparse geometric graphs with small dilation. Computational Geometry **40**(3), 207–219 (2008)

5. Sigurd, M., Zachariasen, M.: Construction of minimum-weight spanners. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 797–808. Springer, Heidelberg (2004)
6. Farshi, M., Gudmundsson, J.: Experimental study of geometric t-spanners. J. Exp. Algorithmics **14**, 3:1.3–3:1.39 (2009)
7. Klein, R., Kutz, M.: Computing geometric minimum-dilation graphs Is NP-hard. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 196–207. Springer, Heidelberg (2007)
8. Magnanti, T.: Wolsey: Optimal trees. CORE discussion paper. Center for Operations Research and Econometrics (1994)
9. Resende, M.G.C., Ribeiro, C.C.: Greedy randomized adaptive search procedures: advances, hybridizations, and applications. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, vol. 57. International Series in Operations Research and Management Science. second edn., pp. 219–249. Springer (2009)
10. Prim, R.C.: Shortest connection networks and some generalizations. The Bell Systems Technical Journal **36**(6), 1389–1401 (1957)
11. Ribeiro, C.C., Resende, M.G.C.: Path-relinking intensification methods for stochastic local search algorithms. Computers and Operations Research **37**, 498–508 (2010)

Speedy Colorful Subtrees

W. Timothy J. White[1(✉)], Stephan Beyer[2], Kai Dührkop[1], Markus Chimani[2], and Sebastian Böcker[1]

[1] Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany
{tim.white,kai.duehrkop,sebastian.boecker}@uni-jena.de

[2] Institute of Computer Science, University of Osnabrück, Osnabrück, Germany
{stephan.beyer,markus.chimani}@uni-osnabrueck.de

Abstract. Fragmentation trees are a technique for identifying molecular formulas and deriving some chemical properties of metabolites—small organic molecules—solely from mass spectral data. Computing these trees involves finding exact solutions to the NP-hard Maximum Colorful Subtree problem. Existing solvers struggle to solve the large instances involved fast enough to keep up with instrument throughput, and their performance remains a hindrance to adoption in practice.

We attack this problem on two fronts: by combining fast and effective reduction algorithms with a strong integer linear program (ILP) formulation of the problem, we achieve overall speedups of 9.4 fold and 8.8 fold on two sets of real-world problems—without sacrificing optimality. Both approaches are, to our knowledge, the first of their kind for this problem. We also evaluate the strategy of solving *global* problem instances, instead of first subdividing them into many *candidate* instances as has been done in the past. Software (C++ source for our reduction program and our CPLEX/Gurobi driver program) available under LGPL at https://github.com/wtwhite/speedy_colorful_subtrees/.

1 Introduction

Metabolites—small molecules involved in cellular reactions—provide a direct functional signature of cellular state. Untargeted metabolomics aims to identify all such compounds present in a biological or environmental sample, and the predominant technology in use is mass spectrometry (MS). This remains a challenging problem, in particular for the many compounds that cannot be found in any spectral library [17,18]. Here we consider tandem mass spectra (MS^2), which measure the masses and abundances of fragments of an isolated compound.

A first step toward full structural elucidation of a compound is the identification of its molecular formula. While it is possible to derive the molecular formula for a given exact mass, measurement inaccuracies have to be considered. Even for high-accuracy instruments, when using an appropriate error range for the mass measurement there may be thousands of possible molecular formulas for a given mass [7]. Approaches for identifying the correct formula include isotope pattern analysis [3], fragmentation pattern analysis [2], or a combination of both [8,9,11,12,15].

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 310–322, 2015.
DOI: 10.1007/978-3-319-21398-9_25

Computation of fragmentation trees [12] is a highly powerful method for fragmentation pattern analysis: In the 2013 CASMI (Critical Assessment of Small Molecule Identification) Challenge for identifying molecular formulas, a combination of fragmentation tree and isotope pattern analysis was selected "best automated tool" [6,10]. In addition, fragmentation tree structure can help to derive information about an unknown compound's structure [13,16]. Peaks in the spectrum are annotated with molecular formulas by looking for *consistent explanations*, using knowledge of possible fragmentation events and their probabilities. This translates into finding exact solutions to the NP-hard Maximum Colorful Subtree (MCS) problem, described later. Unfortunately the problem instances generated can contain over 100,000 edges, and the performance of existing approaches cannot keep up with the throughput of the MS instruments, sometimes limiting the method's appeal in practice. Heuristics often fail to find the optimal solution, and a simple integer linear program (ILP) has been identified as the fastest exact method [14].

We attack this problem on two fronts: by combining fast and effective reduction algorithms with facet-defining inequalities for the ILP formulation of the problem, we achieve overall speedups of 9.4 fold and 8.8 fold on two sets of real-world problems—without sacrificing optimality. Both approaches are, to our knowledge, the first of their kind for this problem. We also evaluate the strategy of solving *global* problem instances, instead of first subdividing them into many *candidate* instances as has been done in the past. Here, we will not evaluate the quality of solutions, as these are identical for any exact method; also, we will assume the edge weights of the MCS problem to be given [5].

1.1 Fragmentation Trees Are Maximum Colorful Subtrees

Consider an MS^2 spectrum containing k peaks $p_1, \ldots, p_k$, having mass-to-charge (m/z) ratios m_i and peak intensities q_i for $1 \leq i \leq k$, listed in decreasing m/z order. Following Böcker and Rasche [2], we use the Round-Robin algorithm [1] to find all possible *explanations* of the parent peak—that is, all candidate molecular formulas having m/z approximately equal to m_1. Each such formula becomes the 1-colored root vertex in a separate MCS instance graph. Within each MCS instance, i-colored vertices are added for each possible explanation of peak p_i, for all $2 \leq i \leq k$. Whenever the molecular formula of v is a subformula of the formula of u, indicating that v could possibly be generated by fragmenting u, we add a directed edge (u, v) and assign an edge weight (which may be positive, negative or zero) according to a probabilistic model of fragmentation. Intuitively, a rooted colorful subtree T in one of these graphs maps each peak to at most one molecular formula in such a way that all formulas in T are consistent with fragmentation of the candidate formula at the root, with the tree of highest total weight corresponding to the best such explanation. By calculating the weights of these optimal trees for all MCS instances and ranking them, the best candidate formula for the spectrum can be determined. Fig. 1 shows an example.

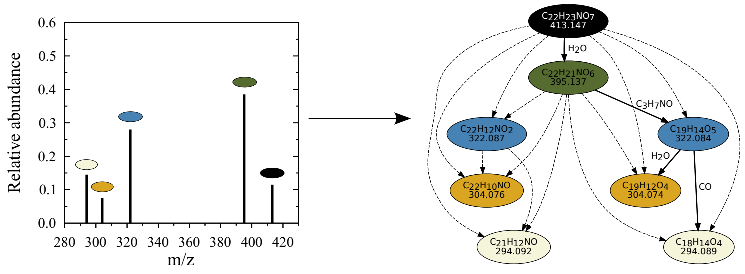

Relative abundance
m/z
0.0
0.1
0.2
0.3
0.4
0.5
0.6
280 300 320 340 360 380 400 420
$C_{22}H_{23}NO_7$ 413.147
H_2O
$C_{22}H_{21}NO_6$ 395.137
C_3H_7NO
$C_{22}H_{12}NO_2$ 322.087
$C_{19}H_{14}O_5$ 322.084
H_2O
$C_{22}H_{10}NO$ 304.076
$C_{19}H_{12}O_4$ 304.074
CO
$C_{21}H_{12}NO$ 294.092
$C_{18}H_{14}O_4$ 294.089

Fig. 1. Example MS^2 spectrum and resulting MCS instance. Nodes (peak explanations) show their molecular formulas and weights. Edges (fragmentation events) in the optimal subtree are solid and labeled with their neutral losses. Edge weights not shown.

The full version of this paper discusses a technique for solving a single *global* MCS instance representing the entire problem, instead of multiple *candidate* instances.

Formally, an instance of the MCS problem is given by (V, E, C, w, c, r) where V is the set of vertices, $E \subsetneq V^2$ is the set of directed edges, C is the set of colors, $w \colon E \to \mathbb{R}$ is the weight function on edges, $c \colon V \to C$ is the function defining colors for each vertex, and $r \in V$ is a distinguished vertex called the *root*. The graph (V, E) is acyclic, and there is a path from r to every $v \in V$.

A subgraph $G' \subseteq G$ is *colorful* iff all vertices in G' have different colors. The weight of an edge $e = (u, v) \in E$ is given by $w(u, v)$, and we define $w(u, v) = -\infty$ when $(u, v) \notin E$. We further extend this function to operate on any subgraph G' in the usual way, by summing over all edges in G'. A subgraph $X \subseteq G$ *dominates* a subgraph $Y \subseteq G$ iff $w(X) \geq w(Y)$.

We would like to assign weights to both edges and vertices: the former to reflect the likelihood of the specific neutral loss in question; the latter to capture peak-specific or explanation-specific information such as peak intensity, mass deviation between measurement and prediction, and estimates of formula plausibility. In order to represent a weight function $w' \colon V \cup E \to \mathbb{R}$ on both vertices and edges using a weight function $w \colon E \to \mathbb{R}$ on edges only, we can simply set $w(u, v) := w'(u, v) + w'(v)$ for each $(u, v) \in E$, since every valid subtree containing v must contain exactly one incoming edge (u, v).

Our goal is to find a maximum colorful r-rooted subtree T of G: that is, among all subtrees rooted at r and in which at most one vertex of any given color appears, a subtree having maximum total weight. This problem is NP-hard. It remains NP-hard even if G is a tree with unit edge weights [2], or if color constraints are dropped [14].[1]

We say that a subgraph G' is *below* a vertex u iff there is a path of zero or more edges from u to every vertex in G'. We denote by G_u the unique maximal

[1] When edge weights are constrained to be nonnegative and color constraints are dropped, all leaves will appear in some optimal solution and the problem reduces to the polynomial-time-solvable maximum spanning tree problem.

subgraph of G below u. A color i is below u iff there exists a path from u to a vertex of color i. Furthermore a color i is below a color i' iff there exists an i'-colored vertex u such that i is below u. A subtree T of a graph G is *full in* G iff it is rooted at some vertex u of G, and every edge in G below u is in T. We use the term "cost" to describe a (typically negative) quantity that is to be *added* to a weight to produce another weight. We also declare a vertex to be an ancestor of itself, and use the adjective *strict* to denote non-self ancestors.

Let $n := |V|$, $m := |E|$, and $k := |C|$. Let $\delta^-(U) := \{(v, u) \mid u \in U, v \notin U\}$, $\delta^+(U) := \{(u, v) \mid u \in U, v \notin U\}$, and $\delta(U) := \delta^-(U) \cup \delta^+(U)$. When $U = \{u\}$ we dispense with the braces. We also define $V_i := \{v \in V \mid c(v) = i\}$.

2 Data Reduction

Our data reduction rules seek to shrink an MCS instance X to a smaller instance X' by deleting edges that are provably unnecessary—that is, edges that are simultaneously absent from some optimal solution to X. Here we outline our rules and their computationally efficient implementations.

Vertex Upper Bounds. The following sections describe upper bounds $U(\cdot)$ on the maximum-weight subtree rooted at some vertex u. Particular upper bounds are named by subscripting U; when just U with no subscript appears, it means that any arbitrary upper bound can be substituted. Trivially we have that $U(u) \geq 0$ for all u, since the 0-weight tree containing just u and no edges is a subtree rooted at u. A computationally useful property of all vertex upper bounds is that they remain valid in the face of edge deletions, enabling reductions to safely delete multiple edges in between bound updates.

Child Upper Bound. A simple upper bound $U_\chi(\cdot)$ for a given vertex u can be obtained by considering the upper bounds of u's children and the edges leading from u to them. Specifically we may choose, among all u's outgoing edges to i-colored children $(u, v) \in \delta^+(u) \cap \delta^-(V_i)$, either the edge (u, v) that maximises $w(u, v) + U(v)$ or no edge if this expression is negative. Summing across colors i yields equation (1). This bound tends to become very loose for vertices near the top of the graph, since high-weight edges near the bottom of the graph will usually be visited by large numbers of paths. Nevertheless it is capable of eliminating many edges near the bottom of the graph when applied to the vertex upper bound reduction rule. It can be considerably strengthened by incorporating other vertex upper bounds, such as the Colorful Forest upper bound.

$$U_\chi(u) = \sum_{i \in \{c(v) \mid (u,v) \in E\}} \max\Big\{0, \max_{\substack{(u,v) \in E,\\ c(v)=i}} \big(w(u, v) + U(v)\big)\Big\} \tag{1}$$

We calculate this bound in $O(m \log k)$ time and $O(n+k)$ space using dynamic programming.

Colorful Forest Upper Bound. We next describe an upper bound $U_\lambda(\cdot)$ obtained by relaxing the subtree constraint. Consider a vertex u, and the subgraph G_u below u. Suppose that for each color i below u we choose either no edge, or some edge $(v, v') \in E(G_u)$ such that $c(v') = i$. All colorful forests in G_u may be generated by choosing incoming edges in this way, and this set of subgraphs contains the set of all colorful subtrees rooted at u, so the problem of finding a maximum-weight colorful forest in G_u is a relaxation of the u-rooted MCS problem. The optimal solution to the relaxed problem is easily found by choosing, for each color i, the maximum-weight incoming edge when this is positive and no edge otherwise, yielding an upper bound on the weight of a colorful subtree rooted at u. This is given in equation (2).

$$U_\lambda(u) = \sum_{i \in C} \max\Big\{0, \max_{\substack{(v,v') \in E(G_u),\\ c(v')=i}} w(v, v')\Big\} \tag{2}$$

Dynamic programming permits calculation in $O(km)$ time and $O(kn)$ space.

Strengthening the Colorful Forest Bound. The Colorful Forest bound can be strengthened by noticing that whenever the forest that it constructs fails to be a tree, we can determine an upper bound on the cost that must be incurred to transform it into one. This upper bound can be added to the weight of the forest to produce a new, stronger vertex upper bound $U_\Lambda(\cdot)$. Here we merely mention that careful implementation allows this stronger bound to be computed in the same time complexity as the original; for a full description, see the full version of this paper.

Anchor Lower Bound. Given that a vertex u is in the solution T, what is a lower bound $L_a(u, v)$ on the cost of forcing in a given vertex v? Here we assume that T does not already contain a $c(v)$-colored vertex, and only consider attaching v to a vertex in T by a single edge.

If v is a child of u, then clearly $w(u, v)$ is a possibility. Regardless, it may still be possible to attach v to a strict ancestor of u. Specifically, since the "anchor" vertex u is in T by assumption, either $u = r$ or one of the parents of u is also in T. Therefore to form a lower bound, we have the option of attaching v to u if this is possible, or to the worst of u's parents, recursively:

$$L_a(u, v) = \begin{cases} \max\left\{w(u, v), \min_{(p,u) \in E} L_a(p, v)\right\}, & u \neq r \\ w(r, v), & u = r \end{cases}$$

recalling that we define $w(u, v) = -\infty$ whenever $(u, v) \notin E$. ($L_a(u, v)$ will produce $-\infty$ iff there is some path from r to u that contains no vertex with an edge to v.)

$L_a(\cdot, \cdot)$ can be computed via dynamic programming in $O(n^2)$ time and space.

It is also helpful to define $L_{a'}(u, v) := \min_{(p,u) \in E} L_a(p, v)$. This variant of $L_a(\cdot, \cdot)$ excludes any direct edge from u to v from consideration.

Slide Lower Bound. Suppose we have a solution T which contains a vertex u. We want to calculate a lower bound $L_s(u, v)$ on the cost of changing T into a new solution T' by replacing u with another given vertex v of the same color as u. We call this the *Slide lower bound* because in the usual representation of fragmentation graphs, vertices of the same color occupy the same row, so forcing v into and u out of T is akin to horizontally sliding the endpoint of an edge in T from u to v. Such a modification may in general completely change the vertices and edges in the tree below u, subject to the important restriction that it respects color usage: that is, it only ever transforms a subtree T_u into a subtree T'_v such that $c(T'_v) \subseteq c(T_u)$. This reflects the fact that we cannot safely insert vertices of new colors, because these colors may already be in use by other parts of the solution. The full version of this paper describes how to compute $L_s(u, v)$ by dynamic programming in $O(mn_k)$ time and space, where n_k is the maximum number of vertices of any color.

Vertex Upper Bound Rule. If for some edge (u, v) we have that $w(u, v) + U(v) \leq 0$, then clearly any solution containing (u, v) is dominated by a solution in which (u, v) and any subtree below it have been deleted, implying that (u, v) can be safely deleted. Applying this rule before other rules removes certain uninteresting special cases from consideration.

Slide Rule. Whenever two edges exist from a vertex u to distinct vertices v and v' of the same color, there is an opportunity to apply the Slide reduction rule. If

$$w(u, v') - w(u, v) + L_s(v, v') > 0 \tag{3}$$

holds then any solution T containing (u, v) can be improved by sliding (u, v) to (u, v'). This rule can be strengthened by replacing the first term with $L_a(u, v')$, which affords us the chance to connect v' to an ancestor of u. We may then usefully allow $v' = v$ to eliminate edges (u, v) that can always be replaced with a better edge (a, v), where a is a strict ancestor of u.[2]

Dominating Path Rule. The idea behind the Slide rule can be taken further: instead of trying to replace an edge (u, v) with another single edge from an ancestor of u to a vertex of the same color as v, we can replace it with a chain of d edges connecting vertices $v_1, \ldots, v_{d+1}$, with the starting point v_1 an ancestor of u and the endpoint v_{d+1} obeying $c(v_{d+1}) = c(v)$ as before. However we must now pay a price for forcing in each internal vertex v_j for all $2 \leq j \leq d$ in this chain, because the solution may already contain some different vertex of color $c(v_j)$ that needs to be dealt with. This can be done for each such internal vertex by using the Slide lower bound. Suppose the path we wish to force in contains some i-colored vertex x, but the solution already contains a conflicting

[2] The full version of this paper discusses a subtlety regarding floating-point arithmetic and comparisons for equality.

vertex—an i-colored vertex $y \neq x$. The solution can be patched up by deleting the incoming edge to y and sliding any subtree below y so that it appears below x for a total cost of $L_s(y,x) - w(p,y)$, where p is y's parent in the solution. Since we do not know, for any color i, which i-colored vertex (if any) is already in the solution, we must take the worst case over all i-colored vertices and all their possible incoming edges:

$$L_{\text{force}}(x) = \min_{y \in V_{c(x)}} \Big(L_s(y,x) - \max_{(p,y) \in E} w(p,y) \Big) \tag{4}$$

It is now possible to state a recursion to calculate an upper bound on the cost to force in a given vertex x, assuming that a vertex u is already in the solution:

$$f(u,x) = \min \Big\{ 0, \alpha, \alpha + L_{\text{force}}(x) \Big\} \tag{5}$$

$$\alpha = \max \Big\{ L_a(u,x), \max_{p,\,(p,x) \in E} \big(f(u,p) + w(p,x) \big) \Big\} \tag{6}$$

We now have that an edge (u,v) can be deleted if there exists an edge (x,z) such that $c(z) = c(v)$ and $w(x,z) + f(u,x) + L_s(v,z) > w(u,v)$. $f(u,x)$ can be calculated in $\mathrm{O}(n)$ space because its first argument never varies during recursion.

Two further reduction rules, the Implied Edge rule and the Color Combining rule, are described in the full version of this paper.

3 Integer Linear Programming

Rauf *et al.* [14] surveyed different methods to obtain optimal solutions of the MCS problem, including an integer linear program (ILP). We extend this to obtain a strictly stronger LP relaxation, and solve the resulting ILP using the cutting plane method.

The ILP of Rauf *et al.* [14] is equivalent to

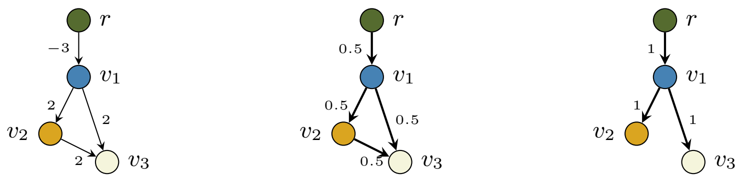
$$\max \sum_{(u,v) \in E} w(u,v)\, x_e \tag{7a}$$

$$\text{s.t.} \sum_{e \in \delta^-(V_i)} x_e \leq 1 \qquad \forall i \in C, \tag{7b}$$

$$\sum_{e \in \delta^-(v)} x_e \geq x_{(v,u)} \qquad \forall (v,u) \in E, v \neq r \tag{7c}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{7d}$$

where x_e is assigned 1 iff the directed edge e is included in the solution. For each $v \in V \setminus \{r\}$ their formulation also includes a constraint $\sum_{e \in \delta^-(v)} x_e \leq 1$, but these constraints are redundant due to the *colorful forest constraints* (7b), which ensure that every color is contained in the solution at most once and that there is at most one incoming directed edge for each vertex. The *connectivity*



(a) the input graph, each vertex has a different color

(b) optimal solution of LP relaxation of (7) with objective value 1.5

(c) optimal solution of LP relaxation of (7)∧(8) with objective value 1

Fig. 2. An example showing that the LP relaxation of (7) including (8) is strictly stronger than without (8)

constraints (7c) say that for each non-root vertex of V, there may only be outgoing directed edges if there is an incoming directed edge. Note that the ILP has a linear number of constraints and variables, so its linear relaxation can be solved as-is without separation.

In this paper, we add the constraints

$$\sum_{e\in\delta^-(V_i)} x_e \leq \sum_{f\in\delta^-(S)} x_f \quad \forall i \in C \quad \forall S \subseteq V, V_i \subseteq S \tag{8}$$

that prohibit splits and joins of fractional values. The constraints are valid for the ILP since they only forbid the case where the left-hand side is 1 and the right-hand side is 0, which could only happen if the result is not connected. However, the constraints make the LP relaxation strictly stronger, as can be seen in Fig. 2: in Fig. 2(b) the incoming value of v_1 is 0.5 and the incoming value of v_3 is 1 which is forbidden by (8) for $S = \{v_1, v_2, v_3\}$ and $i = c(v_3)$.

We first solve the LP for a subset of the constraints. Then, we solve the *separation problem*: we search (8) for one or more violated constraints, add them to the LP, and re-solve, iterating the process until there are no further violated constraints. Here, the separation problem can be answered by finding, for each $i \in C$, a minimum r-V_i-cut in the solution network (V, E, x) and testing if it is less than $\sum_{e\in\delta^-(V_i)} x_e$. Although (8) contains an exponential number of constraints, the separation problem can be solved in polynomial time using a Maximum Flow algorithm, and only a small number of iterations are typically needed to find a feasible LP solution.

Theorem 1. (7b) *and* (8) *provide facet-defining inequalities of the problem polytope and are the only necessary ones.*

The proof is given in the full version of this paper. Although just these inequalities suffice for correctness, we also keep (7c) for evaluation in practice because they do not need to be separated.

4 Results and Discussion

We tested the performance of our reductions and ILP improvements on a spectral dataset containing 1232 compounds that appear in the KEGG http://www.kegg.jp metabolite database. From this we selected hard instances where the "classic" ILP from Rauf *et al.* [14]—previously being the fastest exact method for the MCS problem—showed poor running times. We computed fragmentation graphs for each compound and built two datasets for evaluation:

– graphs100: A set containing the 10 hardest candidate instances as well as a random sample of a further 90 hard candidate instances. We use this dataset to measure the performance of our reductions and ILP improvements.
– fmm1: A set of 20 hard global instances, comprising 86358 candidate instances in total. We use this dataset to compare the heretofore typical strategy of solving all candidate MCS instances separately, to solving a single global instance. Results for this dataset are given in the full version of this paper.

Rauf *et al.* [14] found that 95 % of MCS instances could be solved by ILP in under 5 seconds, while some took up to 5.6 minutes. To this end, it is sufficient to consider the hard instances in our comparison. The full version of this paper describes both the datasets and our results in more detail.

We implemented our reductions in `ft_reduce`, a C++ program that understands a simple language for describing the sequence of reductions to perform, affording flexibility in testing different orders and combinations of reductions. We selected three representative reduction scripts to analyse:

– R1 computes vertex upper bounds using both the Child bound and the Colorful Forest bound, and then applies the Vertex Upper Bound rule.
– R2 does the same, but uses the strengthened Colorful Forest bound.
– R3 applies R2 and then all remaining reduction rules.

Each script iterates until no more edges can be removed. The full version of this paper gives the complete scripts.

We implemented our new ILP formulation using a C++ driver program linked with CPLEX 12.6.0 (http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/). Our new facet-defining cuts can be turned on or off using a command-line argument. In the remainder, we call the solver with these cuts turned on "CPLEX+Cuts", and the solver with them turned off "CPLEX" or "stock CPLEX". For the separation of the split-and-join constraints, we use the Maximum Flow code by Cherkassky and Goldberg [4]. We also performed tests using Gurobi 5.5.0 (http://www.gurobi.com/), although we were not able to implement the cuts efficiently using its callback framework.

All computational experiments were performed on a cluster of four 12-CPU 2.4GHz E5645 Linux machines with 48 GB RAM each. All reductions and all ILP solver runs for the graphs100 dataset ran to completion with a RAM limit of 4 GB and a time limit of 2 hours in place. For the fmm1 dataset, the memory limit was increased from 4 GB to 12 GB, but some instances failed to run to completion in the 2 hour limit. Our reduction program is single-threaded, and ILP solvers were operated in single-threaded mode. All time measurements are in elapsed (wallclock) seconds, and exclude time spent on I/O.

4.1 Results for graphs100 Dataset

Fig. 3 (left) shows the effectiveness of our reductions in shrinking the graphs100 problem instances. Every R1 or R2 reduction removed at least 11.6 % of the edges, and every R3 reduction removed at least 35.4 %, with the average reductions being 62.4 %, 64.3 % and 70.4 % for R1, R2 and R3, respectively. Many instances produced much larger reductions, and it is clear from Fig. 3 (left) that reduced instance size is only very weakly correlated with original instance size.

Fig. 3 (right) compares the performance of various combinations of reduction scripts and ILP solvers. Two effects are immediately apparent: using the strengthened ILP formulation improves average solution times for CPLEX by at least a factor of 4; and applying either the R1 or R2 reduction script produces anywhere from a 30.9 % decrease (from unreduced to R2 on stock CPLEX) to a 57.6 % decrease (from unreduced to R1 on CPLEX+Cuts). We note with particular interest that applying both techniques is substantially *more* effective than would be expected by performing each separately: assuming their effects on running time to be independent, we would expect that both performing an R1 reduction and changing from stock CPLEX to CPLEX+Cuts would result in instances taking on average $0.69109 * 0.25065 = 0.173$ times as long to solve, but

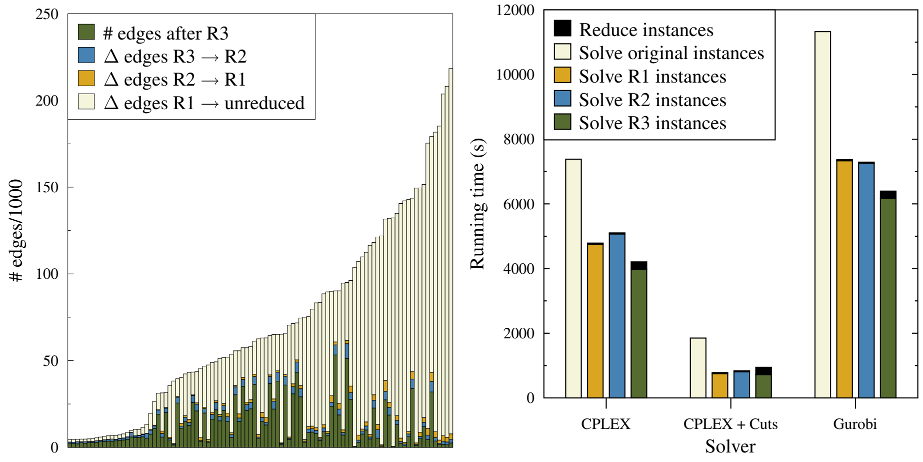


Fig. 3. Left: Comparing unreduced and reduced instance sizes for each graphs100 instance. The bottom bar in each stack gives the number of edges after R3 reduction; higher bars correspond to weaker reductions, with the entire stack indicating the unreduced instance size. Right: Running time evaluation for all graphs100 instances. Each column shows the total elapsed time needed to solve all 100 instances, with reduction time broken out as a black bar at the top.

in fact we find that they take only 0.106 times as long—a relative improvement of 38.7 %, representing a 9.42-fold overall reduction in execution time.

In the other direction, we observe that both stock CPLEX and CPLEX+Cuts take slightly longer to solve the R2 instances than the R1 instances, despite the fact that every R2 instance's edge set is a strict subset of the corresponding R1 instance's edge set, having on average 5.1 % fewer edges. We can only surmise that the additional edges removed by using the strengthened Colorful Forest bound destroyed some structure sought by CPLEX's various heuristics.

The more expensive R3 reductions are a net improvement for the stock CPLEX and Gurobi solvers, but result in an overall slowdown for CPLEX+Cuts.

The highest memory usage on any unreduced instance was 1166MB, 1199MB and 956MB for stock CPLEX, CPLEX+Cuts and Gurobi, respectively. On reduced instances these figures dropped to 853MB, 498MB and 640MB. The highest memory usage by our `ft_reduce` program was 15MB, 15MB and 31MB for R1, R2 and R3 reductions, respectively.

5 Conclusion

We have presented two highly effective techniques for accelerating the optimal solution of MCS instances, thereby bringing practical *de novo* identification of metabolite molecular formulas a step closer to reality. The two methods complement each other admirably: applying both yields a *larger* speedup than the product of the speedups obtained by applying each separately. Based on our experiments with two real-world datasets, we find that it is essentially always advantageous to use our strengthened ILP formulation and to apply our simple reductions, and frequently advantageous to apply our more complex ones.

The lion's share of the improvement in running times comes from our new, facet-defining cutting planes for ILP solvers. ILP solvers have demonstrated effectiveness across a wide range of hard optimization problems, and we anticipate that they will remain the dominant approach to solving MCS problems. At the same time, the problem reductions we present offer immediately-available speedups (and, often, memory usage reductions) not only for ILP formulations but for any exact or heuristic solution method, such as the "brute force" algorithm of Böcker and Rasche [2] or the Tree Completion heuristic of Rauf *et al.* [14].

We noted above that the use of fragmentation trees goes beyond the determination of molecular formulas [13]: see for instance Shen *et al.* [16] where fragmentation trees are used in conjunction with machine learning to search a molecular structure database using fragmentation spectra. In this analysis pipeline, computing fragmentation trees accounts for more than 90 % of the total running time. To this end, faster methods for this task are highly sought.

References

1. Böcker, S., Lipták, Z.: A fast and simple algorithm for the Money Changing Problem. Algorithmica **48**(4), 413–432 (2007)
2. Böcker, S., Rasche, F.: Towards de novo identification of metabolites by analyzing tandem mass spectra. Bioinformatics **24**, I49–I55 (2008). Proc. of European Conference on Computational Biology (ECCB 2008)
3. Böcker, S., Letzel, M., Lipták, Z., Pervukhin, A.: SIRIUS: Decomposing isotope patterns for metabolite identification. Bioinformatics **25**(2), 218–224 (2009)
4. Cherkassky, B., Goldberg, A.: On implementing push-relabel method for the maximum flow problem. Algorithmica **19**, 390–410 (1997)
5. Dührkop, K., Böcker, S.: Fragmentation trees reloaded. In: Przytycka, T.M. (ed.) RECOMB 2015. LNCS, vol. 9029, pp. 65–79. Springer, Heidelberg (2015)
6. Dührkop, K., Hufsky, F., Böcker, S.: Molecular formula identification using isotope pattern analysis and calculation of fragmentation trees. Mass Spectrom **3**(special issue 2), S0037 (2014)
7. Kind, T., Fiehn, O.: Metabolomic database annotations via query of elemental compositions: Mass accuracy is insufficient even at less than 1 ppm. BMC Bioinformatics **7**(1), 234 (2006)
8. Menikarachchi, L.C., Cawley, S., Hill, D.W., Hall, L.M., Hall, L., Lai, S., Wilder, J., Grant, D.F.: MolFind: A software package enabling HPLC/MS-based identification of unknown chemical structures. Anal Chem **84**(21), 9388–9394 (2012)
9. Meringer, M., Reinker, S., Zhang, J., Muller, A.: MS/MS data improves automated determination of molecular formulas by mass spectrometry. MATCH-Commun Math Co **65**, 259–290 (2011)
10. Nishioka, T., Kasama, T., Kinumi, T., Makabe, H., Matsuda, F., Miura, D., Miyashita, M., Nakamura, T., Tanaka, K., Yamamoto, A.: Winners of CASMI2013: Automated tools and challenge data. Mass Spectrom **3**(special issue 2), S0039 (2014)
11. Pluskal, T., Uehara, T., Yanagida, M.: Highly accurate chemical formula prediction tool utilizing high-resolution mass spectra, MS/MS fragmentation, heuristic rules, and isotope pattern matching. Anal Chem **84**(10), 4396–4403 (2012)
12. Rasche, F., Svatoš, A., Maddula, R.K., Böttcher, C., Böcker, S.: Computing fragmentation trees from tandem mass spectrometry data. Anal Chem **83**(4), 1243–1251 (2011)
13. Rasche, F., Scheubert, K., Hufsky, F., Zichner, T., Kai, M., Svatoš, A., Böcker, S.: Identifying the unknowns by aligning fragmentation trees. Anal Chem **84**(7), 3417–3426 (2012)
14. Rauf, I., Rasche, F., Nicolas, F., Böcker, S.: Finding maximum colorful subtrees in practice. J Comput Biol **20**(4), 1–11 (2013)
15. Rojas-Chertó, M., Kasper, P.T., Willighagen, E.L., Vreeken, R.J., Hankemeier, T., Reijmers, T.H.: Elemental composition determination based on MS^n. Bioinformatics **27**, 2376–2383 (2011)
16. Shen, H., Dührkop, K., Böcker, S., Rousu, J.: Metabolite identification through multiple kernel learning on fragmentation trees. Bioinformatics **30**(12), 157–164 (2014). Proc. of Intelligent Systems for Molecular Biology (ISMB 2014)

17. Tautenhahn, R., Cho, K., Uritboonthai, W., Zhu, Z., Patti, G.J., Siuzdak, G.: An accelerated workflow for untargeted metabolomics using the METLIN database. Nat Biotechnol **30**(9), 826–828 (2012)
18. Wishart, D.S., Knox, C., Guo, A.C., Eisner, R., Young, N., Gautam, B., Hau, D.D., Psychogios, N., Dong, E., Bouatra, S., Mandal, R., Sinelnikov, I., Xia, J., Jia, L., Cruz, J.A., Lim, E., Sobsey, C.A., Shrivastava, S., Huang, P., Liu, P., Fang, L., Peng, J., Fradette, R., Cheng, D., Tzur, D., Clements, M., Lewis, A., Souza, A.D., Zuniga, A., Dawe, M., Xiong, Y., Clive, D., Greiner, R., Nazyrova, A., Shaykhutdinov, R., Li, L., Vogel, H.J., Forsythe, I.: HMDB: A knowledgebase for the human metabolome. Nucleic Acids Res **37**, D603–D610 (2009)

Graph Algorithms II

Algorithmic Aspects of Disjunctive Domination in Graphs

B.S. Panda[1], Arti Pandey[1](✉), and S. Paul[2]

[1] Department of Mathematics, Indian Institute of Technology Delhi Hauz Khas, New Delhi 110016, India
{bspanda,artipandey}@maths.iitd.ac.in

[2] Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata 700108, India
paulsubhabrata@gmail.com

Abstract. For a graph $G = (V, E)$, a set $D \subseteq V$ is called a *disjunctive dominating set* of G if for every vertex $v \in V \setminus D$, v is either adjacent to a vertex of D or has at least two vertices in D at distance 2 from it. The cardinality of a minimum disjunctive dominating set of G is called the *disjunctive domination number* of graph G, and is denoted by $\gamma_2^d(G)$. The Minimum Disjunctive Domination Problem (MDDP) is to find a disjunctive dominating set of cardinality $\gamma_2^d(G)$. Given a positive integer k and a graph G, the Disjunctive Domination Decision Problem (DDDP) is to decide whether G has a disjunctive dominating set of cardinality at most k. In this article, we first propose a polynomial time algorithm for MDDP in proper interval graphs. Next we tighten the NP-completeness of DDDP by showing that it remains NP-complete even in chordal graphs. We also propose a $(\ln(\Delta^2 + \Delta + 2) + 1)$-approximation algorithm for MDDP, where Δ is the maximum degree of input graph $G = (V, E)$ and prove that MDDP can not be approximated within $(1 - \epsilon)\ln(|V|)$ for any $\epsilon > 0$ unless NP $\subseteq$ DTIME$(|V|^{O(\log\log|V|)})$. Finally, we show that MDDP is APX-complete for bipartite graphs with maximum degree 3.

Keywords: Domination · Chordal graph · Graph algorithm · Approximation algorithm · Np-complete · Apx-complete

1 Introduction

Let $G = (V, E)$ be a graph. For a vertex $v \in V$, let $N_G(v) = \{u \in V | uv \in E\}$ and $N_G[v] = N_G(v) \cup \{v\}$ denote the *open neighborhood* and the *closed neighborhood* of v, respectively. For two distinct vertices $u, v \in V$, the distance $dist_G(u, v)$ between u and v is the length of a shortest path between u and v. A vertex u *dominates* v if either $u = v$ or u is adjacent to v. A set $D \subseteq V$ is called a *dominating set* of $G = (V, E)$ if each $v \in V$ is dominated by a vertex in D, that is, $|N_G[v] \cap D| \geq 1$ for all $v \in V$. The *domination number* of a graph G, denoted by $\gamma(G)$, is the minimum cardinality of a dominating set of G. For a

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 325–336, 2015.
DOI: 10.1007/978-3-319-21398-9_26

graph G, the Minimum Domination problem is to find a dominating set of cardinality $\gamma(G)$. Domination in graphs is one of the classical problems in graph theory and it has been well studied from theoretical as well as algorithmic point of view [9,10]. Over the years, many variants of domination problem have been studied in the literature due to its application in different fields. The concept of *disjunctive domination* is a recent and an interesting variation of domination [8].

In domination problem, our goal is to place minimum number of sentinels at some vertices of the graph so that all the remaining vertices are adjacent to at least one sentinel. In practice, depending upon the monitoring power, we can have different types of sentinels. To secure the graph with different types of sentinels, we need concept of different variants of domination. Efforts made in this direction have given rise to different types of domination, such as, distance domination, exponential domination, secondary domination. In some cases, it might happen that the monitoring power of a sentinel is inversely proportional to the distance, that is, the domination power of a vertex reduces as the distance increases. Motivated by this idea, Goddard et al. [8] have introduced the concept of *disjunctive domination* which captures the notion of decay in domination with increasing distance. A set $D_d \subseteq V$ is called a *b-disjunctive dominating set* of G if every vertex $v \in V \setminus D_d$ is either adjacent to a vertex in D_d or there are at least b vertices of D_d within a distance of two from v. The minimum cardinality of a b-disjunctive dominating set of G is called the *b-disjunctive domination number* of G and it is denoted by $\gamma_b^d(G)$. A vertex v is said to be *b-disjunctively dominated* by $D_d \subseteq V$ if either $v \in D_d$ or v is adjacent to a vertex of D_d or has at least b vertices in D_d at distance 2 from it. Note that disjunctive domination is more general concept than distance two domination, since the parameter $\gamma_1^d(G)$ is the distance two domination number. For simplicity, 2-disjunctive domination is called disjunctive domination. The disjunctive domination problem and its decision version are defined as follows:

Minimum Disjunctive Domination Problem (MDDP)

Instance: A graph $G = (V, E)$.
Solution: A disjunctive dominating set D_d of G.
Measure: Cardinality of the set D_d.

Disjunctive Domination Decision Problem (DDDP)

Instance: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Does there exist a disjunctive dominating set D_d of G such that $|D_d| \leq k$?

The concept of disjunctive domination has been introduced recently in 2014 [8] and further studied in [11]. In [8], Goddard et al. have proven bounds on disjunctive domination number for specially regular graphs and claw-free graphs. They have shown that the decision version of b-disjunctive domination is NP-complete for planar and bipartite graphs and also designed a dynamic programming based linear time algorithm to find a minimum b-disjunctive dominating set in a tree. In [11], Henning et al. have studied the relation between domination number and disjunctive domination number of a tree T and proved that $\gamma(T) \leq 2\gamma_2^d(T) - 1$. They have also given a constructive characterization of the trees achieving equality in this bound. On the other hand, a variation of disjunctive domination is also studied in the literature (see [12]).

In this paper, our focus is on algorithmic study of disjunctive domination problem. The rest of the paper is organized as follows. In Section 2, we give some pertinent definitions and notations that would be used in the rest of the paper. In this section, we also observe some graph classes where domination problem is NP-complete but disjunctive domination can be easily solved and vice versa. This motivates us to study the status of the problem in other graph classes. In Section 3, we design a polynomial time algorithm for disjunctive domination problem in proper interval graphs, an important subclass of chordal graphs. In Section 4, we prove that DDDP remains NP-complete for chordal graphs. In Section 5, we design a polynomial time approximation algorithm for MDDP for general graph G with approximation ratio $\ln(\Delta^2+\Delta+2)+1$, where Δ is the maximum degree of G. In this section, we also prove that MDDP can not be approximated within $(1-\epsilon)\ln(|V|)$ for any $\epsilon > 0$ unless NP $\subseteq$ DTIME$(|V|^{O(\log\log|V|)})$. In addition, for bipartite graphs with maximum degree 3, MDDP is shown to be APX-complete in this section. Finally, Section 6 concludes the paper.

2 Preliminaries

2.1 Notations

Let $G = (V, E)$ be a graph. Let $N_G^2(v)$ denote the set of vertices which are at distance 2 from the vertex v in graph G. Let $G[S]$, $S \subseteq V$ denote the induced subgraph of G on the vertex set S. The *degree* of a vertex $v \in V$, denoted by $d_G(v)$, is the number of neighbors of v, that is, $d_G(v) = |N_G(v)|$. The *minimum degree* and *maximum degree* of a graph G is defined by $\delta(G) = \min_{v\in V} d_G(v)$ and $\Delta(G) = \max_{v\in V} d_G(v)$, respectively. A set $S \subseteq V$ is called an *independent set* of G if $uv \notin E$ for all $u, v \in S$. A set $K \subseteq V$ is called a *clique* of G if $uv \in E$ for all $u, v \in K$. A set $C \subseteq V$ is called a *vertex cover* of G if for each edge $ab \in E$, either $a \in C$ or $b \in C$. Let n and m denote the number of vertices and number of edges of G, respectively. In this paper, we only consider connected graphs with at least two vertices.

2.2 Graph Classes

A graph G is said to be a *chordal graph* if every cycle in G of length at least four has a *chord*, that is, an edge joining two non-consecutive vertices of the cycle. Let $\mathscr{F}$ be a family of sets. The *intersection graph* of $\mathscr{F}$ is obtained by taking each set in $\mathscr{F}$ as a vertex and joining two sets in $\mathscr{F}$ if and only if they have a non-empty intersection. A graph G is an *interval graph* if G is the intersection graph of a family $\mathscr{F}$ of intervals on the real line. A graph G is called a *proper* *interval* graph if it is the intersection graph of a family $\mathscr{F}$ of intervals on the real line such that no interval in $\mathscr{F}$ contains another interval in $\mathscr{F}$ set-theoretically. A vertex $v \in V(G)$ is a *simplicial* vertex of G if $N_G[v]$ is a clique of G. An ordering $\alpha = (v_1, v_2, ..., v_n)$ is a *perfect elimination ordering* (PEO) of G if v_i is a simplicial vertex of $G_i = G[\{v_i, v_{i+1}, ..., v_n\}]$ for all i, $1 \leq i \leq n$. A graph G has a PEO if and only if G is chordal [7]. A PEO $\alpha = (v_1, v_2, \ldots, v_n)$ of a chordal graph is a *bi-compatible elimination ordering* (BCO) if $\alpha^{-1} = (v_n, v_{n-1}, \ldots, v_1)$, that is, the reverse of α, is also a PEO of G. A graph G has a BCO if and only if G is a proper interval graph [14].

2.3 Domination vs Disjunctive Domination

In this subsection, we make some observations on complexity difference of domination and disjunctive domination problem. It is known that domination problem is NP-complete for split graphs [4] and for graphs with diameter two [2]. But disjunctive domination problem can be easily solved in these graph classes. Because, disjunctive domination number is at most 2 in these classes and $\gamma_2^d(G) = 1$ if and only if G contains a vertex of degree $n - 1$. Next, we define a graph class, called *GC graph*, for which domination problem is easily solvable, but disjunctive domination problem is NP-complete.

Definition 1 (GC graph). *A graph $G' = (V', E')$ is said to be a* GC graph *if it can be constructed from a general graph $G = (V, E)$ by adding a pendant vertex to every vertex of G. Formally, $V' = V \cup \{w_i \mid 1 \leq i \leq n\}$ and $E' = E \cup \{v_i w_i \mid 1 \leq i \leq n\}$.*

Note that, every vertex of a GC graph G' is either a pendant vertex or adjacent to a unique pendant vertex and hence, $\gamma(G') = n$. In Section 4, we show that DDDP is NP-complete for the class of GC graphs.

3 Polynomial Time Algorithm for Proper Interval Graphs

In this section, we present a polynomial time algorithm to find a minimum cardinality disjunctive dominating set in proper interval graphs.

Let $\alpha = (v_1, v_2, \ldots, v_n)$ be a BCO of the proper interval graph G. Let $MaxN_G(v_i)$ denote the maximum index neighbor of v_i with respect to the ordering α. We start with an empty set D. At each iteration i of the algorithm, we update the set D in such a way that the vertex v_i and all the vertices which appear before v_i in the BCO α, are disjunctively dominated by the set D. At the end of n^{th} iteration, D disjunctively dominate all the vertices of graph G. The algorithm DISJUNCTIVE-PIG for finding a minimum cardinality disjunctive dominating set in a proper interval graph is given below.

Next we give the proof of correctness of the algorithm. Let $\alpha = (v_1, v_2, \ldots, v_n)$ be the BCO of a proper interval graph G. Define the set $V_i = \{v_1, v_2, \ldots, v_i\}$, $1 \leq i \leq n$, and $V_0 = \emptyset$. Also suppose that D_i denotes the set D obtained after processing vertex v_i, $1 \leq i \leq n$, and $D_0 = \emptyset$. We will prove that D_n is a minimum cardinality disjunctive dominating set of G.

Algorithm 1. DISJUNCTIVE-PIG($G, \alpha = (v_1, v_2, \ldots, v_n)$)

```
Initialize $D = \emptyset$;
for $i = 1 : n$ do
    Compute $N_G[v_i] \cap D$ and $N_G^2(v_i) \cap D$;
    Case 1: Either $N_G[v_i] \cap D \neq \emptyset$, or $|N_G^2(v_i) \cap D| \geq 2$
        No update in $D$ is done;
    Case 2: $N_G[v_i] \cap D == \emptyset$ and $N_G^2(v_i) \cap D == \emptyset$
        Update $D$ as $D = D \cup \{MaxN_G(v_i)\}$;
    Case 3: $N_G[v_i] \cap D == \emptyset$ and $|N_G^2(v_i) \cap D| == 1$
        Find $v_r \in N_G^2(v_i) \cap D$;
        $v_j = MaxN_G[v_i]$; $v_k = MaxN_G[v_j]$;
        $S = \{v_{i+1}, v_{i+2}, \ldots, v_{j-1}\}$;
        Subcase 3.1: For every $v \in S$, either $vv_k \in E$ or $d(v, v_r) = 2$
            Update $D$ as $D = D \cup \{v_k\}$;
        Subcase 3.2: $v_s$ is the least index vertex in $S$ such that
        $d(v_s, v_k) = 2$ and $d(v_s, v_r) > 2$
            Update $D$ as $D = D \cup \{MaxN_G(v_s)\}$;
return $D$;
```

Theorem 1. *For each i, $0 \leq i \leq n$, the following statements are true:*

(a) *D_i disjunctively dominates the set V_i.*
(b) *There exists a minimum cardinality disjunctive dominating set D_d^* such that D_i is contained in D_d^*.*

Proof. We prove the theorem by induction on i. The basis step is trivial as $D_0 = \emptyset$. Next assume that the theorem is true for $i-1$. So, (a) D_{i-1} disjunctively dominates the set V_{i-1}, (b) there exists a minimum cardinality disjunctive dominating set D_d^* such that D_{i-1} is contained in D_d^*.

Next we prove the theorem for i. According to our algorithm, we need to discuss the following three cases.

Case 1: Either $N_G[v_i] \cap D_{i-1} \neq \emptyset$, or $|N_G^2(v_i) \cap D_{i-1}| \geq 2$.

Here $D_i = D_{i-1}$. It is easy to notice that all the conditions of the theorem are satisfied.

Case 2: $N_G[v_i] \cap D_{i-1} = \emptyset$ and $N_G^2(v_i) \cap D_{i-1} = \emptyset$.

Here $D_i = D_{i-1} \cup \{v_j\}$ where $v_j = MaxN_G(v_i)$. Hence, condition (a) of the theorem is trivially satisfied. If $v_j \in D_d^*$, then $D_i \subseteq D_d^*$. Hence both the conditions of the theorem are satisfied, and D_d^* is the required minimum cardinality disjunctive dominating set of G. If $v_j \notin D_d^*$, then there are two possibilities:

(I) There exists a vertex $v_p \in N_G[v_i] \cap D_d^*$.
Define the set $D_d^{**} = (D_d^* \setminus \{v_p\}) \cup \{v_j\}$. Note that $D_i \subseteq D_d^{**}$, and $|D_d^*| = |D_d^{**}|$. Now, to prove condition (b) of the theorem, it is enough to show that D_d^{**} is a disjunctive dominating set of G. Note that $D_{i-1} \cup \{v_j\} \subseteq D_d^{**}$. Now consider an arbitrary vertex v_a of G. If $a < i$, then the vertex v_a is disjunctively dominated

by the set D_{i-1}, and hence by D_d^{**}. If $a \geq i$, and $v_p \in N_G[a]$, then $v_j \in N_G[v_a]$. If $a \geq i$, and $v_p \in N_G^2(v_a)$, then $v_j \in N_G[v_a]$ or $v_j \in N_G^2(v_a)$. This proves that D_d^{**} is a disjunctive dominating set of G.

(II) For $q < s$, vertices $v_q, v_s \in N_G^2(v_i) \cap D_d^*$.
Let $MaxN_G(v_i) = v_j$ and $MaxN_G(v_j) = v_k$. Then $q < s \leq k$. Let $v_t = MaxN_G(v_s)$ and $v_r = MaxN_G(v_t)$. We again consider three possibilities:

(i) $q < s < i$
Here $r \leq j$. Now consider an arbitrary vertex v_a of G. If $a < i$, then the vertex v_a is disjunctively dominated by the set D_{i-1}. If $a \geq i$, and $v_s \in N_G^2(v_a)$ or $v_q, v_s \in N_G^2(v_i)$, then $v_j \in N_G[v_a]$. Hence $(D_d^* \setminus \{v_q, v_s\}) \cup \{v_j\}$ is a disjunctive dominating set of G of cardinality less than $|D_d^*|$, which is a contradiction, as D_d^* is a minimum disjunctive dominating set of G. Therefore, this situation will never arise.
(ii) $q < i < s$
Consider an arbitrary vertex v_a of G. If $a < i$, then the vertex v_a is disjunctively dominated by the set D_{i-1}. If $a \geq i$, and $v_q \in N_G^2(v_a)$, then $v_j \in N_G[v_a]$. If $a \geq i$, and $v_q \notin N_G^2(v_a)$, and either $v_s \in N_G[v_a]$ or $v_s \in N_G^2(v_a)$, then either $v_j \in N_G[v_a]$ or $v_t \in N_G[v_a]$. Hence, if we define $D_d^{**} = (D_d^* \setminus \{v_q, v_s\}) \cup \{v_j, v_t\}$, then D_d^{**} is a minimum cardinality disjunctive dominating set of G and $D_i \subseteq D_d^{**}$. This proves the condition (b) of the theorem.
(iii) $i < q < s$
Here $s \leq k$. Consider an arbitrary vertex v_a of G. If $a < i$, then the vertex v_a is disjunctively dominated by the set D_{i-1}. If $a \geq i$, and $v_q \in N_G[v_a]$ or $v_s \in N_G[v_a]$ or $v_q, v_s \in N_G^2(v_a)$ or $v_s \in N_G^2(v_a)$, then either $v_j \in N_G[v_a]$ or $v_t \in N_G[v_a]$. Hence, if we define $D_d^{**} = (D_d^* \setminus \{v_q, v_s\}) \cup \{v_j, v_t\}$, then D_d^{**} is a minimum cardinality disjunctive dominating set of G and $D_i \subseteq D_d^{**}$. This proves the condition (b) of the theorem.

Case 3: $N_G[v_i] \cap D_{i-1} = \emptyset$ and $|N_G^2(v_i) \cap D_{i-1}| = 1$.
In this case as well, condition (a) and condition (b) of the theorem are satisfied. Due to space constraints, the proof is omitted.
Hence our theorem is proved. □

In view of the above theorem, the set D computed by the algorithm DISJUNCTIVE-PIG is a minimum cardinality disjunctive dominating set of G. Now, we show that the algorithm DISJUNCTIVE-PIG can be implemented in polynomial time. We use the adjacency list representation of the graph. We maintain an array D_{set} for the set D such that $D_{set}[j] = 1$ if $v_j \in D$. We maintain the all pair distance matrix $Dist[1..n, 1..n]$ such that $Dist[i, j]$ is the distance between v_i and v_j. This can be done in $O(n^3)$ time. Now $N_G[v_i] \cap D$ can be computed in $O(n)$ time by looking up $Dist$ matrix and array D_{set}. Similarly, $N_G^2(v_i) \cap D$ can be computed in $O(n)$ time. Also $MaxN_G(v_i)$ can be computed in $O(n)$ time. Hence, in any iteration, all the operations can be done in $O(n^2)$ time. Therefore overall time is $O(n^3)$, as number of iterations are n. Since, BCO of a proper interval graph can be computed in $O(n + m)$ time [15], and all the computations in the algorithm DISJUNCTIVE-PIG can be done in $O(n^3)$ time, we have the following theorem.

Theorem 2. *MDDP can be solved in $O(n^3)$ time in proper interval graphs.*

However, the algorithm DISJUNCTIVE-PIG can be implemented in $O(n + m)$ time using additional data structures. The details are omitted due to space constraints.

4 NP-completeness

In this section, we prove that DDDP is NP-complete for chordal graphs. For that, we first show that DDDP is NP-complete for GC graphs. To prove this NP-completeness result, we use a reduction from another variant of domination problem, namely 2-*domination problem*. For a graph $G = (V, E)$, a set $D_2 \subseteq V$ is called 2-*dominating set* if every vertex $v \in V \setminus D_2$ has at least two neighbors in D_2. Given a positive integer k and a graph $G = (V, E)$, the 2-DOMINATION DECISION PROBLEM (2DDP) is to decide whether G has a 2-dominating set of cardinality at most k. It is known that 2DDP is NP-complete for chordal graphs [13]. The following lemma shows that DDDP is NP-complete for GC graphs.

Lemma 1. *DDDP is NP-complete for GC graphs.*

Proof. Clearly, DDDP is in NP for GC graphs. To prove the NP-hardness, we give a polynomial transformation from 2DDP for general graphs. Let $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$, and a positive integer k be an instance of 2DDP. Given G, we construct the graph $G' = (V', E')$ in the following way: $V' = V \cup \{w_i \mid 1 \leq i \leq n\}$ and $E' = E \cup \{v_i w_i \mid 1 \leq i \leq n\}$. Clearly G' is a GC graph and it can be constructed from G in polynomial time.

The following claim is enough to complete the proof of the theorem.

Claim 1 G has a 2-dominating set of cardinality at most k if and only if G' has a disjunctive dominating set of cardinality at most k.

Proof. (Proof of the claim) Let D_2 be a 2-dominating set of G of cardinality at most k. Clearly D_2 is a disjunctive dominating set of G'. Because every $v_i \in V'$ either is in D_2 or dominated by at least two vertices of D_2 and every $w_i \in V'$ is either dominated by $v_i \in D_2$ or contains at least two vertices from D_2 at a distance of two. Hence, G' has a disjunctive dominating set of cardinality at most k.

Conversely, suppose that D_d is a disjunctive dominating set of G' of cardinality at most k. Note that, every vertex of G' is either a pendant vertex or a support vertex. Also, the vertex set of graph G is exactly the set of all support vertices of G'. Let P be the set of pendant vertices of graph G', i.e., $P = \{w_i \mid 1 \leq i \leq n\}$. If a pendant vertex $w_i \in D_d$, then the set $D'_d = (D_d \setminus \{w_i\}) \cup \{v_i\}$ still remains a disjunctive dominating set of G' of cardinality at most k. So, without loss of generality we assume that $D_d \cap P = \emptyset$. Now for every vertex $v_i \in V$, either $v_i \in D_d$ or $|N_G(v_i) \cap D_d| \geq 2$. If not, let there is a vertex $v_i \in V \setminus D_d$ such that

$|N_G(v_i) \cap D_d| \leq 1$. This implies that the vertex $w_i \in V'$ is neither dominated nor has at least two vertices from D_d at a distance of two, contradicting the fact that D_d is a disjunctive dominating set of G'. Hence, D_d is a 2-dominating set of G of cardinality at most k. □

Hence, it is proved that DDDP is NP-complete for GC graphs. □

It is easy to observe that, if the graph G is chordal, then the constructed graph G' in Lemma 1 is also chordal. Hence, we have the following theorem.

Theorem 3. *DDDP is NP-complete for chordal graphs.*

5 Approximation Results

5.1 Approximation Algorithm

In this subsection, we propose a $(\ln(\Delta^2 + \Delta + 2) + 1)$-approximation algorithm for MDDP. Our algorithm is based on the reduction from MDDP to the Constrained Multiset Multicover (CMSMC) problem. We first recall the definition of the Constrained Multiset Multicover problem.

Let X be a set and $\mathcal{F}$ be a collection of subsets of X. The Set Cover problem is to find a smallest sub-collection, say $\mathcal{C}$ of $\mathcal{F}$, such that $\mathcal{C}$ covers all the elements of X, that is, $\cup_{S \in \mathcal{C}} S = X$. The Constrained Multiset Multicover problem is a generalization of the Set Cover problem. In this problem, $\mathcal{F}$ is the collection of multisets of X, that is, each element $x \in X$ occurs in a multiset $S \in \mathcal{F}$ with arbitrary multiplicity, and each element $x \in X$ has an integer coverage requirement r_x which specifies how many times x has to be covered. Note that each set $S \in \mathcal{F}$ is chosen at most once. So, for a given set X, a collection $\mathcal{F}$ of multisets of X, and integer requirement r_x for each $x \in X$, the CMSMC problem is to find a smallest collection $\mathcal{C} \subseteq \mathcal{F}$, such that $\mathcal{C}$ covers each element x in X at least r_x times. In the case, when r_x is constant for each $x \in X$, then $\mathcal{C}$ is called a r_x-cover of X, and the CMSMC problem is to find a minimum cardinality r_x-cover of X.

Theorem 4. *The* Minimum Disjunctive Domination Problem *for a graph $G = (V, E)$ with maximum degree Δ can be approximated with an approximation ratio of* $\ln(\Delta^2 + \Delta + 2) + 1$.

Proof. Let us show the transformation from MDDP to the CMSMC problem. **Construction :** Let $G = (V, E)$ be a graph with n vertices and m edges where $V = \{v_1, v_2, \ldots, v_n\}$ (an instance of MDDP). Now we construct an instance of the CMSMC problem, that is, a set X, a family $\mathcal{F}$ of multisets of X, and a vector $R = (r_x)_{x \in X}$ (r_x is a non-negative integer for each $x \in X$) in the following way:

$X = V$, $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$, where for each i, $1 \leq i \leq n$, F_i is a multiset which contains two copies of each element in $N_G[v_i]$ and one copy of the set of elements which are at distance 2 from the vertex v_i in graph G, $r_x = 2$ for each $x \in X$.

Now we first need to prove the following correspondence.

Claim 2 The set $D = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ is a disjunctive dominating set of G if and only if $\mathcal{C} = \{F_{i_1}, F_{i_2}, \ldots, F_{i_k}\}$ is a 2-cover of X.

Proof. The proof is omitted due to space constraints. □

By the above claim, if D_d^* is a minimum cardinality disjunctive dominating set of G and $\mathcal{C}^*$ is an optimal 2-cover of X, then $|D_d^*| = |\mathcal{C}^*|$. In [16], S. Rajgopalan and V. V. Vazirani gave a greedy approximation algorithm for the CMSMC problem, which achieves an approximation ratio of $\ln(|F_M|) + 1$, where F_M is the maximum cardinality multiset in $\mathcal{F}$. Let $\mathcal{C}^*$ be an optimal 2-cover and $\mathcal{C}'$ be a 2-cover obtained by greedy approximation algorithm, then $|\mathcal{C}'| \le (\ln(|F_M|) + 1) \cdot |\mathcal{C}^*|$. Given a 2-cover of X, we can also obtain a disjunctive dominating set of graph G of same cardinality. Suppose that D'_d is a disjunctive dominating set of G obtained from 2-cover $\mathcal{C}'$ of X. Then $|D'_d| \le (\ln(|F_M|) + 1) \cdot |D_d^*|$. If the maximum degree of the graph G is Δ, then the cardinality of a set in family $\mathcal{C}$ will be at most $2(\Delta + 1) + \Delta(\Delta - 1)$, which is equal to $\Delta^2 + \Delta + 2$. Hence $|D'_d| \le (\ln(\Delta^2 + \Delta + 2) + 1) \cdot |D_d^*|$. This completes the proof of the theorem. □

5.2 Lower Bound on Approximation Ratio

To obtain the lower bound, we give an approximation preserving reduction from the Minimum Domination problem. The following approximation hardness result for the Minimum Domination problem is already known.

Theorem 5. *[5] For a graph $G = (V, E)$, the* Minimum Domination *problem can not be approximated within $(1 - \epsilon) \ln |V|$ for any $\epsilon > 0$ unless NP $\subseteq$ DTIME $(|V|^{O(\log \log |V|)})$.*

Theorem 6. *For a graph $G = (V, E)$, MDDP can not be approximated within $(1 - \epsilon) \ln |V|$ for any $\epsilon > 0$ unless NP $\subseteq$ DTIME$(|V|^{O(\log \log |V|)})$.*

Proof. Let us describe the reduction from the Minimum Domination problem to MDDP. Let $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ be an instance of the Minimum Domination problem. Now, we construct a graph $H = (V_H, E_H)$ an instance of MDDP in the following way: $V_H = V \cup \{w_i, z_i \mid 1 \le i \le n\} \cup \{p, q\}$, $E_H = E \cup \{v_i w_i, w_i z_i, z_i p \mid 1 \le i \le n\} \cup \{pq\}$.

Fig. 1 illustrates the construction of the graph H from a given graph G. Note that $|V_H| = 3|V| + 2$.

If D^* is a minimum cardinality dominating set of G, then $D^* \cup \{p\}$ is a disjunctive dominating set of H. Hence for a minimum cardinality disjunctive dominating set D_d^* of H, $|D_d^*| \le |D^*| + 1$.

On the other hand, let D_d be a disjunctive dominating set of H. Consider the vertex w_i. Since w_i is disjunctively dominated by the set D_d, one of the following possibilities may occur:
(i) $v_i \in D_d$, *(ii)* $w_i \in D_d$ or $z_i \in D_d$, *(iii)* $|N_H^2(w_i) \cap D_d| \ge 2$, that is, $N_G(v_i) \cap D_d \ne \emptyset$.

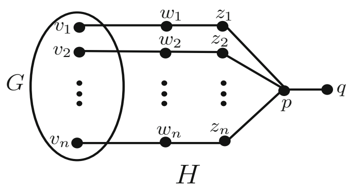


Fig. 1. An illustration to the construction of H from G

If (ii) occurs, then define $D_d = (D_d \setminus \{w_i, z_i\}) \cup \{v_i\}$. Do it for all i, $1 \leq i \leq n$. Note that the set $D = D_d \cap V$ dominates all the vertices of G, and $|D| \leq |D_d|$.

Now suppose that MDDP can be approximated with an approximation ratio of α, where $\alpha = (1 - \epsilon)\ln(|V_H|)$ for some fixed $\epsilon > 0$, by a polynomial time approximation algorithm APPROX-DISJUNCTIVE. Let l be a fixed positive integer. Consider the following algorithm to compute a dominating set of a given graph G.

Algorithm 2. APPROX-DOMINATION(G)

```
Input: A graph $G = (V, E)$.
Output: A dominating set $D$ of graph $G$.
begin
    if there exists a minimum dominating set $D'$ of cardinality $\leq l$ then
        $D = D'$;
    else
        Construct the graph $H$;
        Compute a disjunctive dominating set $D_d$ of $H$ using the
        algorithm APPROX-DISJUNCTIVE;
        for $i = 1 : m$ do
            if $w_i \in D_d$ or $z_i \in D_d$ then
                $D_d = (D_d \setminus \{w_i, z_i\}) \cup \{v_i\}$;
        $D = D_d \cap V$;
    return $D$;
```

Clearly, the algorithm APPROX-DOMINATION outputs a dominating set of G in polynomial time. If the cardinality of a minimum dominating set of G is at most l, then it can be computed in polynomial time. So, we consider the case, when the cardinality of a minimum dominating set of G is greater than l. Let D^* denotes a minimum cardinality dominating set of G, and D_d^* denotes a minimum cardinality disjunctive dominating set of H. Note that $|D^*| > l$.

Let D be the dominating set of G computed by the algorithm APPROX-DOMINATION, then $|D| \leq |D_d| \leq \alpha|D_d^*| \leq \alpha(|D^*| + 1) = \alpha(1 + \frac{1}{|D^*|})|D^*| < \alpha(1 + \frac{1}{l})|D^*|$.

Since ϵ is fixed, there exists a positive integer l such that $\frac{1}{l} < \epsilon$. So, $|D| < \alpha(1 + \epsilon)|D^*| = (1 - \epsilon)(1 + \epsilon)\ln(|V_H|)|D^*| = (1 - \epsilon')\ln(|V_H|)|D^*|$.

Since $|V_H| = 3|V| + 1$, and $|V|$ is very large, $\ln(|V_H|) \approx \ln(|V|)$. Hence $|D| < (1 - \epsilon') \ln(|V|)|D^*|$. Hence, the dominating set D computed by the algorithm APPROX-DOMINATION achieves an approximation ratio of $(1-\epsilon') \ln(|V|)$ for some $\epsilon' > 0$.

By Theorem 5, if the Minimum Domination problem can be approximated within a ratio of $(1-\epsilon') \ln(|V|)$, then NP $\subseteq$ DTIME$(|V|^{O(\log \log |V|)})$. This proves that for a graph $H = (V_H, E_H)$, MDDP can not be approximated within a ratio of $(1 - \epsilon) \ln(|V_H|)$ unless NP $\subseteq$ DTIME$(|V_H|^{O(\log \log |V_H|)})$. □

5.3 APX-completeness

In this subsection, we prove that MDDP is APX-complete for bounded degree graphs. To prove this, we need the concept of L-reduction, which is defined as follows.

Definition 2. *Given two NP optimization problems F and G and a polynomial time transformation f from instances of F to instances of G, we say that f is an L-reduction if there are positive constants α and β such that for every instance x of F*

1. *$opt_G(f(x)) \leq \alpha \cdot opt_F(x)$.*
2. *for every feasible solution y of $f(x)$ with objective value $m_G(f(x), y) = c_2$ we can in polynomial time find a solution y' of x with $m_F(x, y') = c_1$ such that $|opt_F(x) - c_1| \leq \beta|opt_G(f(x)) - c_2|$.*

To show the APX-completeness of a problem $\Pi \in$APX, it is enough to show that there is an L-reduction from some APX-complete problem to Π [3].

By Theorem 4, it is clear that MDDP can be approximated within a constant factor for bounded degree graphs. Thus the problem is in APX for bounded degree graphs. To show the APX-hardness of MDDP, we give an L-reduction from the Minimum Vertex Cover Problem (MVCP) for 3-regular graphs which is known to be APX-complete [1].

Theorem 7. *The* Minimum Disjunctive Domination Problem *is APX-complete for bipartite graphs with maximum degree* 3.

Proof. The proof is omitted due to space constraints.

6 Conclusion

In this article, we have proposed a linear time algorithm for MDDP in proper interval graphs. We have also tightened the NP-completeness of DDDP by showing that it remains NP-complete even in chordal graphs. From approximation point of view, we have proposed an approximation algorithm for MDDP in general graphs and have shown that this problem is APX-complete for bipartite graphs with maximum degree 3. It would be interesting to study the complexity of this problem in other graph classes and also the relation between disjunctive domination number and other domination parameters.

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoret. Comput. Sci. **237**(1–2), 123–134 (2000)
2. Ambalath, A.M., Balasundaram, R., Rao H., C., Koppula, V., Misra, N., Philip, G., Ramanujan, M.S.: On the kernelization complexity of colorful motifs. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 14–25. Springer, Heidelberg (2010)
3. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation. Springer, Berlin (1999)
4. Bertossi, A.A.: Dominating sets for split and bipartite graphs. Inf. Process. Lett. **19**(1), 37–40 (1984)
5. Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems in bounded degree graphs. Inform. and Comput. **206**, 1264–1275 (2008)
6. Dankelmann, P., Day, D., Erwin, D., Mukwembi, S., Swart, H.: Domination with exponential decay. Discrete Math. **309**, 5877–5883 (2009)
7. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific J. Math. **15**, 835–855 (1965)
8. Goddard, W., Henning, M.A., McPillan, C.A.: The disjunctive domination number of a graph. Quaestiones Math. **37**(4), 547–561 (2014)
9. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of domination in graphs. Marcel Dekker Inc., New York (1998)
10. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs, Advanced Topics. Marcel Dekker Inc., New York (1998)
11. Henning, M.A., Marcon, S.A.: Domination versus disjunctive domination in trees. Discrete Appl. Math. (2014)
12. Henning, M.A., Naicker, V.: Disjunctive total domination in graphs. J. Comb. Optim. (2014). doi:10.1007/s10878-014-9811-4
13. Jacobson, M.S., Peters, K.: Complexity questions for n-domination and related parameters. In: Eighteenth Manitoba Conference on Numerical Mathematics and Computing, Winnipeg, MB (1988), Congr. Numer. **68**, 722 (1989)
14. Jamison, R.E., Laskar, R.: Elimination orderings of chordal graphs. In: Combinatorics and applications. ISI, Calcutta, pp. 192–200 (1982, 1984)
15. Panda, B.S., Das, S.K.: A linear time recognition algorithm for proper interval graphs. Inform. Process. Lett. **87**(3), 153–161 (2003)
16. Rajgopalan, S., Vazirani, V.V.: Primal-dual RNC approximation algorithms for set cover and covering integer programs. SIAM J. Comput. **28**, 526–541 (1999)

Algorithmic Aspect of Minus Domination on Small-Degree Graphs

Jin-Yong Lin[1], Ching-Hao Liu[1](✉), and Sheung-Hung Poon[2]

[1] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
yongdottw@hotmail.com, chinghao.liu@gmail.com

[2] School of Computing and Informatics, Institut Teknologi Brunei, Gadong, Brunei Darussalam
sheung.hung.poon@gmail.com

Abstract. Let $G = (V, E)$ be an undirected graph. A *minus dominating function* for G is a function $f : V \rightarrow \{-1, 0, +1\}$ such that for each vertex $v \in V$, the sum of the function values over the closed neighborhood of v is positive. The *weight* of a minus dominating function f for G, denoted by $w(f(V))$, is $\sum f(v)$ over all vertices $v \in V$. The *minus domination (MD) number* of G is the minimum weight for any minus dominating function for G. The *minus domination (MD) problem* asks for the minus dominating function which contributes the MD number. In this paper, we first show that the MD problem is $W[2]$-hard for general graphs. Then we show that the MD problem is NP-complete for subcubic bipartite planar graphs. We further show that the MD problem is APX-hard for graphs of maximum degree seven. Lastly, we present the first fixed-parameter algorithm for the MD problem on subcubic graphs, which runs in $O^*(2.3761^{5k})$ time, where k is the MD number of the graph.

1 Introduction

Let $G = (V, E)$ be an undirected graph. A *minus dominating function* for G is a function $f : V \rightarrow \{-1, 0, +1\}$ such that for each vertex $v \in V$, the sum of the function values over the closed neighborhood of v is positive, where the *closed neighborhood* of v is the set contains v and all neighbors of v. The *weight* of a minus dominating function f for G, denoted by $w(f(V))$, is $\sum f(v)$ over all vertices $v \in V$. The *minus domination (MD) number* of G, denoted by $\gamma^-(G)$, is the minimum weight for any minus dominating function for G. The *minus domination (MD) problem* asks for the minus dominating function which contributes the MD number.

According to [9], we can see that the function values $+1$ and -1 of the signed domination problem can be formulated as yes-no decisions for social networks. In a similar fashion, the function values $+1$, 0, and -1 of the MD problem can be formulated as yes-uncertain-no decisions for in social networks. In the following, we first mention related work on complexities. Dunbar et al. [4,5] first showed that the MD problem is NP-complete for bipartite graphs and for chordal graphs, and can be solved in linear time for trees. Then, Damaschke [2]

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 337–348, 2015.
DOI: 10.1007/978-3-319-21398-9_27

showed that the MD problem is NP-complete for planar graphs of maximum degree 4. They [2] also showed that for every fixed k there is a polynomial-time algorithm, which runs in time $O(3^{8k} \cdot n^{8k})$, for deciding whether a given graph G of maximum degree 4 satisfies $\gamma^{-}(G) \leq k$. Recently, Faria et al. [6] showed that the MD problem is NP-complete for splitgraphs, and is polynomial for graphs of bounded rankwidth and for strongly chordal graphs.

Next, we survey the related work on parameterized complexities. It is known that Downey et al. [3] showed that the domination problem is $W[2]$-complete. Then, Faria et al. [6] showed that the MD problem has no fixed-parameter algorithm, i.e., not in $W[0]$, unless $P = NP$. They also show that the MD problem is fixed-parameter tractable for planar graphs, when parameterized by the *size* of f, the number of vertices $x \in V$ with $f(x) = 1$, and for d-degenerate graphs, when parameterized by the size of f and by d.

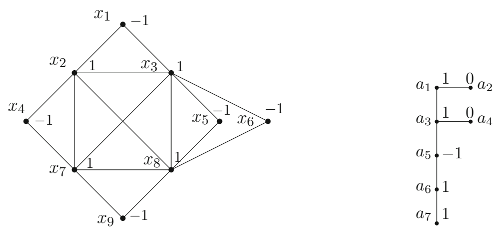
Lastly, we survey the related work on approximation complexities for graphs of bounded degree. First, Alimonti and Kann [1] showed that the domination problem is APX-hard on subcubic graphs. Then, Damaschke [2] showed that the MD number cannot be approximated in polynomial time within a factor $1 + \epsilon$, for some $\epsilon > 0$, for graphs of maximum degree 4, unless $P = NP$.

Outline. We organize the rest of this paper as follows. In Section 2, we show that the MD problem is $W[2]$-hard for general graphs. In Section 3, we show that the MD problem is NP-complete for subcubic bipartite planar graphs. In Section 4, we show the MD problem is APX-hard for graphs of maximum degree 7. In Section 5, with a very involved analysis, we obtain the first FPT-algorithm for the MD problem on subcubic graphs G, which runs in time $O^{*}(2.3761^{5k})$, where k is the MD number of G.

2 $W[2]$-hardness for General Graphs

In this section, by reducing the domination problem to the MD problem, we show that the MD problem on general connected graphs is $W[2]$-hard. Thus we introduce domination problem as follows. A *dominating set* of a graph $G = (V, E)$ is a vertex set $D \subseteq V$ such that for each vertex $v \in V$, there exists at least one vertex in the closed neighborhood $N_G[v]$ belonging to D. In other words, we can label the vertices in V with $\{0, +1\}$ such that the closed neighborhood of each vertex is positive. The *domination number* of G, denoted by $\gamma(G)$ is the cardinality of the minimum dominating set of G. The *domination problem* asks for a dominating set which contributes the domination number $\gamma(G)$. Downey et al. [3] showed that the domination problem on general connected graphs is $W[2]$-complete.

In our reduction, we need to make use of a graph H as shown in Figure 1(a). If we label the vertices of H as shown in Figure 1(a), where there are four vertices labeled with $+1$ and five vertices labeled with -1, then the weight for such a minus domination function is -1. In the following lemma, we show that -1 is in fact the minimum weight which a legitimate minus dominating function for graph H can provide.



(a) (b)

Fig. 1. (a) Graph H with weight -1. (b) A key with minimum weight 3.

Lemma 1. *The MD number of graph H shown in Figure 1(a) is at least -1.*

Proof. Suppose to the contrary that $\gamma^-(H) \leq -2$. Let f be a minus dominating function of H whose weight is not more than -2. Let P be the set of vertices labeled with $+1$ in H. If $\gamma^-(H) \leq -2$, then we have $|P| \leq 3$. It is clear that $|P| > 1$. If $|P| = 2$, it is easy to see that the rest vertices should label with 0 in H. Thus we have $|P| \neq 2$. If $|P| = 3$, then there are at most three vertices labeled with -1 in H. Thus we also have $|P| \neq 3$. With such a contradiction, we thus have $\gamma^-(H) \geq -1$. □

With this lemma, now we show the $W[2]$-hardness for the MD problem.

Theorem 1. *The MD problem is $W[2]$-hard for general connected graphs.*

Proof. We reduce the domination problem to our problem as follows. Given a graph $G = (V, E)$, we construct a new graph G' as described below. Initially we set G' to be G. Then we add each vertex v of G' a set of $\deg_G(v) + 1$ keys as shown in Figure 1(b), and connect v to a_7 of each keys in this set, where $\deg_G(v)$ is the degree of vertex v in G. Note that at most one of vertex in each key can be labeled with -1, and thus the weight for each key is greater than or equal to three. We let F be the set of newly added vertices. Next, we add $6m + 2n$ copies of graph H into graph G' and denote them by $H_1, H_2, \ldots, H_{6m+2n}$, respectively. Let vertex set $Y = V(H_1) \cup \ldots \cup V(H_{6m+2n})$. Then we take any previously added key, say ρ. Now, we connect the vertex a_1 of ρ to the corresponding vertex x_2 of each copy $H_i; i = \{1, \ldots, 6m + 2n\}$ of graph H. This completes the construction of the graph G'. Let V' be the set of vertices in G'. As G is a connected graph, graph G' is also a connected graph by our construction. In the following, we show that there is a domination set of size at most k in G if and only if there is a minus dominating function with weight at most k in G'.

Assume that there is a dominating set D of size at most k in G. Then we will show that there is a minus dominating function f with weight at most k of G'. We construct a function f, which labels vertices in F as Figure 1(b), vertices in D with value 0, vertices in $V \setminus D$ with value -1. We also label the corresponding vertices x_2, x_3, x_7, x_8 with value $+1$ and x_1, x_4, x_5, x_6, x_9 with value -1 for each

$H_i; i = \{1, \ldots, 6m + 2n\}$. Now, we claim that f is a minus dominating function of G' with weight at most k. Clearly, the sum weight of the closed neighborhood of any vertex in $F \cup Y$ is positive. Then we consider vertices in $V' \setminus (F \cup Y)$. For any vertex v in $V' \setminus (F \cup Y)$, since D is a dominating set, there is at least one vertex in $N_{G'}[v] \cap D$. Thus more than half of the vertices in $N_{G'}[v]$ are labeled with $+1$ in function f. Thus the weight for induced subgraph $G[N'_G[v]]$ is positive. Hence, f is a legitimate minus dominating function of G'. Since the weight of each H_i is at least -1 by Lemma 1, we obtain that

$$w(f(V')) \leq -(n-k) + (-1)(6m + 2n) + 3(2m + n) = k.$$

Conversely, we can also show that if f is a minus dominating function of G' with weight at most k, then there is a dominating set size at most k in G. The proof is omitted here. Hence, we complete the proof. □

3 NP-completeness for Subcubic Bipartite Planar Graphs

A graph is called *cubic* if its vertex degrees are degree three, and *subcubic* if its vertex degrees are at most three. It has been shown in [2] that the MD problem on planar graphs of maximum degree four is NP-complete. In this section, we show that the MD problem is NP-complete even for subcubic bipartite planar graphs. We leave open the question that whether the MD problem is NP-complete for cubic bipartite planar graphs. For showing our NP-completeness results, we make use of a lemma presented by Damaschke in [2].

Lemma 2 (Lemma 3 of [2]). *In any graph, we consider a vertex x of degree 1 and the unique neighbor w of x. Then there is an optimal minus dominating function such that $f(x) = 0$ and $f(w) = 1$.*

Then we show the main NP-completeness theorem in the following.

Theorem 2. *The MD problem is NP-complete for subcubic bipartite planar graphs.*

Proof. Clearly, the problem is in NP. We reduce the planar 3SAT problem [7] to this problem. The input instance for the planar 3SAT problem is a set $\{x_1, x_2, \ldots, x_n\}$ of n variables and a Boolean expression with conjunctive normal form $\Phi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ of m clauses, where each clause consists of exactly three literals, such that the variable clause graph of the input instance is planar. The *planar 3SAT problem* asks for whether there exists a truth assignment to the variables so that the Boolean expression Φ is satisfied. We then describe our polynomial-time reduction as follows.

Variable Gadget. First, we construct the variable gadget V_i for a variable x_i. The variable gadget V_i for x_i is a circular linkage as shown in Figure 2(a). We connect $4m + 2$ keys (of Figure 1(b)) together as shown in Figure 2(a), where $2m + 1$ keys are connected as a *chain of keys* for the upper part of V_i, and the other $2m + 1$ keys are connected as a chain of keys for the lower part of V_i .

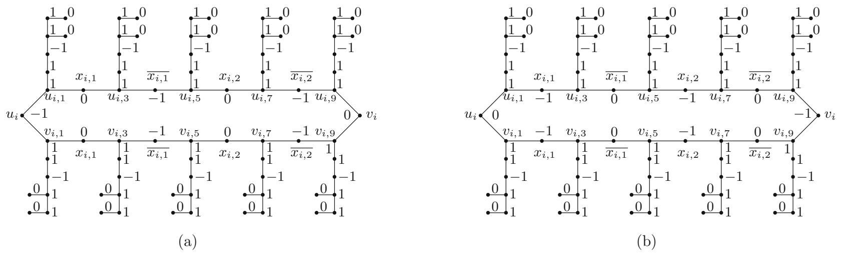


Fig. 2. Gadget V_i for variable x_i: (a) x_i = TRUE; (b) x_i = FALSE. Note that vertices $x_{i,j}$ and $\overline{x_{i,j}}$ represents the positive and negative literals for x_i, respectively. They are used to connect to a clause gadget C_j.

The connection of the keys is performed by the introduction of a cycle of length $8m + 4$. See Figure 2(a). We call such a circular linkage a *cycle of keys*. See Figure 2(a). We let u_i and v_i be the leftmost and the rightmost endpoints of V_i. Then we let $u_i, u_{i,1}, u_{i,2}, \ldots, u_{i,4m+1}, v_i$ be the upper chain starting from u_i to v_i. Similarly, we let $u_i, v_{i,1}, v_{i,2}, \ldots, v_{i,4m+1}, v_i$ be the lower chain starting from u_i to v_i. The vertices $u_{i,2}, u_{i,4}, \ldots, u_{i,4m}$ and $v_{i,2}, v_{i,4}, \ldots, v_{i,4m}$ are used for connecting with clause gadgets. That is, $u_{i,2}, u_{i,6}, \ldots, u_{i,4m-2}$ and $v_{i,2}, v_{i,6}, \ldots, v_{i,4m-2}$ are parts for positive literals $x_{i,j}$, and $u_{i,4}, u_{i,8}, \ldots, u_{i,4m}$ and $v_{i,4}, v_{i,8}, \ldots, v_{i,4m}$ are parts for negative literals $\overline{x_{i,j}}$. The interior of the whole variable gadget can be duplicated to make a longer gadget so that there are enough ports on the variable gadget for connecting to the corresponding literal gadgets of the related clauses in the later context.

Next, we first describe the truth assignment of the optimal minus dominating function for a key. The labeling method in Figure 1(b) is optimal, whose weight is 3. There is another way of optimal labeling such that the lowest vertex of the key is labeled with value 0. Since the lowest vertex of a key is connected to the main body of variable gadget, it is always advantageous to use the labeling method shown in Figure 1(b).

Then we describe the truth assignment of the optimal minus dominating function for the whole variable gadget. To attain the assignment of minimum weight, the internal cycle of variable gadget V_i may be labeled as either the way in Figure 2(a) or the way in Figure 2(b). In either way, the sum of the weights $f(x)$ for $x \in V_i$ is $8m$ and such $f(x)$ is the minimum minus dominating function. We use the domination way in Figure 2(a) to represent that x_i = TRUE, and the other domination way in Figure 2(b) to represent that x_i = FALSE.

Clause Gadget. We use clause gadgets to connect to the variable gadgets directly and there is no link gadget. Now we are prepared to construct the clause gadget C_j for clause $c_j = x_i \vee x_k \vee x_\ell$ which contains 28 vertices, that is, $p_1, \ldots, p_8, q_1, \ldots, q_8, r_1, \ldots, r_8, s_1, s_2, s_3, t$, and 33 edges as shown in Figure 3(a). The vertices p_0, q_0 and r_0 lie in variable gadgets V_i, V_k and V_ℓ, respectively. If x_i is TRUE or FALSE, then p_0 connects to a vertex in V_i which represents $x_{i,j}$ or $\overline{x_{i,j}}$, respectively. Similarly, if x_k is TRUE or FALSE, then q_0 connects to a vertex in V_k which represents $x_{k,j}$ or $\overline{x_{k,j}}$, respectively. If x_ℓ is TRUE, then r_0 connects

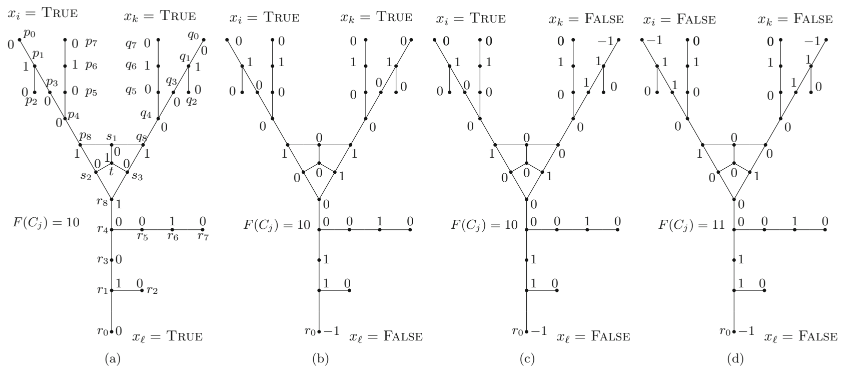


Fig. 3. Gadget C_j for clause c_j: (a) $x_i = x_k = x_\ell =$ TRUE; (b) $x_i = x_k =$ TRUE and $x_\ell =$ FALSE; (c) $x_i =$ TRUE and $x_k = x_\ell =$ FALSE; (d) $x_i = x_k = x_\ell =$ FALSE

to a vertex in V_ℓ which represents $x_{\ell,j}$; otherwise, if x_ℓ is FALSE, then r_0 connects to a vertex in V_ℓ which represents $\overline{x_{\ell,j}}$. This completes the construction of the clause gadget.

We denote the minimum weight for the clause gadget by

$$F(C_j) = \min_f \{ \sum_x f(x) \mid x \in \{p_1, \ldots, p_8, q_1, \ldots, q_8, r_1, \ldots, r_8, s_1, s_2, s_3, t\} \}.$$

By Lemma 2, we assign $f(p_1) = f(q_1) = f(r_1) = 1$, $f(p_2) = f(q_2) = f(r_2) = 0$, $f(p_6) = f(q_6) = f(r_6) = 1$ and $f(p_7) = f(q_7) = f(r_7) = 0$. Then since $\sum_{x\in N[p_4]} f(x) \geq 1$, $\sum_{x\in N[q_4]} f(x) \geq 1$, $\sum_{x\in N[r_4]} f(x) \geq 1$ and $\sum_{x\in N[s_1]} f(x) \geq 1$, thus $F(C_j) \geq 10$. Here we claim that if a clause is TRUE, then $F(C_j)$ meets the lower bound, that is, $F(C_j) = 10$; otherwise, if a clause is FALSE, then $F(C_j) = 11$.

To prove $F(C_j) = 10$ for a TRUE clause gadget, we show that there exists an assignment of $f(x)$ such that $\sum_{x\in N[p_4]} f(x) = 1$, $\sum_{x\in N[q_4]} f(x) = 1$, $\sum_{x\in N[r_4]} f(x) = 1$, and $\sum_{x\in N[t]} f(x) = 1$. To prove $F(C_j) = 11$ for a FALSE clause gadget, we show that there exists an assignment of $f(x)$ such that one of $\sum_{x\in N[p_4]} f(x)$, $\sum_{x\in N[q_4]} f(x)$, $\sum_{x\in N[r_4]} f(x)$, and $\sum_{x\in N[t]} f(x)$ is two, and the rest of them are one.

Analysis of Truth Assignment. We need to analyze totally four cases for the truth assignments for the clause gadget mentioned above. We discuss the case that all three literals are FALSE in the following paragraph since it is the most important case. As for the other three cases, we omit the analysis.

Suppose that all three literals x_i, x_k and x_ℓ are FALSE as shown in Figure 3(d). We prove that $F(C_j) = 10$ is impossible as follows. Suppose that $F(C_j) = 10$. Then $\sum_{x\in N[y]} f(x) = 1$ for $y \in \{p_4, q_4, r_4, t\}$. Since x_i, x_k and x_ℓ are FALSE, we assign $f(x)$ for $x \in N[p_4]$ by setting $f(p_3) = 1$ and $f(p_4) = f(p_5) = 0$, and we perform the similar labeling for $x \in N[q_4]$ and for $x \in N[r_4]$. On the other hand, it is easy to check that the minimum sum of weight of vertices in $\{p_8, q_8, r_8, s_1, s_2, s_3, t\}$ is not less than 2. Hence, $F(C_j) \geq 11$.

In a true assignment of clause c_j, the minimum weight of $F(C_j)$ for the optimal minus dominating function for gadget C_j is 10, which is omitted due to lack of space. Hence, the Boolean expression Φ is satisfied if and only if the constructed graph has a minus dominating function of weight $n(8m+4)+10m$. $\square$

4 APX-hardness for Graphs of Maximum Degree 7

In this section, we show that the MD problem for graphs of maximum degree 7 is APX-hard. Alimonti and Kann [1] show that the domination problem is APX-hard on subcubic graphs. Here we use a well-known technique called *L-reduction* to show the APX-hardness for the MD problem. See [8] for more details on L-reduction. We show that our problem satisfies the two main properties of L-reduction.

We perform an L-reduction f from an instance of the domination problem on subcubic graphs to the corresponding instance of the MD problem on graphs of maximum degree 7.

Given a subcubic graph $G=(V,E)$, we construct a graph $G'=(V',E')$ as follows. For each vertex v in V, we add $\deg_G(v)+1$ keys as Figure 1(b) and connect a_7 of each key to v. This completes the construction of G'. For each vertex $v \in V$, we have just added one more edge connecting to the vertex v for each edge adjacent to v in graph G. Since graph G is subcubic, we have that G' is of maximum degree 7. Next, we obtain the following lemma.

Lemma 3. *Let $G=(V,E)$ be a subcubic graph and let the corresponding $G'=(V',E')$ constructed as described above. If D^* is a minimum dominating set of G, and f^* is a minimum minus dominating function of G'. Then $w(f^*(V'))=|D^*|+6m+2n$, where $n=|V|$ and $m=|E|$.*

Proof. We construct a function f by assigning the vertices $\{a_1,a_3,a_6,a_7\}$ in each keys with value $+1$, we label D^* and the vertices $\{a_2,a_4\}$ in each keys with value 0, and label other vertices with value -1. Then we verify whether function f is a valid minus dominating function of G', that is, we verify whether $N_G[x]>0$ for each vertex x in G'.

It is clear that the sum of function values of the neighborhood of each vertex in keys is greater than 0. Now we claim that the same holds for the remaining vertices in V'. For a vertex v labeled with 0, the vertex has at most $\deg_G(v)$ neighbors labeled with -1, since there are $\deg_G(v)+1$ keys connecting to v, we obtain that $N_G[x]>0$. Moreover, we consider au vertex labeled with -1 in V'. Since the corresponding vertex of u in G is not in D^*, there must be at least a neighbor of u in G' labeled with value 0. Hence, the sum of the function values of the closed neighborhood of u is positive. Then we calculate the weight for minus dominating function f, $w(f)=3(2m+n)-(n-|D^*|)=6m+2n+|D^*|$. Thus we have $w(f^*)\leq w(f)=3(2m+n)-(n-k)=6m+2n+|D^*|$.

Now let f^* be the minimum minus dominating function of G', let vertex set D in G collect those vertices labeled with 0 and $+1$ in f^*. Then we claim that D is a dominating set of G, in other words, we claim that for each vertex v in

$G \setminus D$, there is at least a neighbor in D. Let v' be a vertex labeled with -1 in G'. Note that there are $2\deg_G(v)+2$ vertices in $N_{G'}[v']$. It is clear that the vertices in $N_{G'}[v'] \cap F \setminus \{v\}$ must be labeled with $+1$. Since f^* is a minus dominating function, the sum of $N_{G'}[v']$ must be positive. Hence, there must exist at least one neighbor of $v \notin F$ is labeled with 0 or $+1$. That is, D is a dominating set of G. Then we calculate the size of set D. $|D| \leq w(f^*) - 3(2m+n) + n = w(f^*) - 6m - 2n$. Thus we have $|D^*| \leq |D| \leq w(f^*) - 6m - 2n$. Hence we have $w(f^*(V')) = |D^*| + 6m + 2n$. This completes the proof. □

Since G is a subcubic graph, we have $m \leq \frac{3n}{2}$. Moreover, according to Lemma 4, $n \leq 5|D^*|$. Hence, $w(f^*) = |D^*| + 6m + 2n \leq |D^*| + 9n + 2n \leq |D^*| + 55|D^*| = 56|D^*|$. Since $w(f^*) \leq 56|D^*|$, then $\alpha = 56$.

Now we consider a minus dominating function f of G', we can construct a dominating set D for G by the above algorithm, then we have $|D| = w(f) - 6m - 2n$. Thus we obtain that $|D| - |D^*| = w(f) - 6m - 2n - (w(f^*) - 6m - 2n) = w(f) - w(f^*)$ Thus $\beta = 1$. Hence, we have proved that f is an L-reduction with $\alpha = 56$ and $\beta = 1$. Finally, we obtain the following theorem.

Theorem 3. *The MD problem is APX-hard for graphs of maximum degree* 7.

5 An FPT-algorithm for Subcubic Graphs

In Section 2, we have shown that the MD problem for subcubic bipartite planar graphs is NP-complete. It thus follows that the MD problem for subcubic graphs is NP-complete. Then it is interesting to study FPT-algorithms for the MD problem on subcubic graphs parameterized by the MD number. Th the best of our knowledge, there is no such FPT-algorithm in the literature.

In this section, we thus present the first FPT-algorithm for the MD problem on subcubic graphs G parameterized by the MD number k. In the following lemma, we begin with showing that our problem has a kernel of size $5k$. Then with an involved analysis, we come up with an FPT-algorithm which runs in time $O^*(2.3761^{5k})$.

Lemma 4. *There is a kernel of size* $5k$ *for the MD problem on subcubic graphs, and the bound is tight, where* k *is the weight for the minus dominating function of graph* G.

Proof. Let $G = (V, E)$ be a subcubic graph. For any minus dominating function of G with weight k, we claim that $|V| \leq 5k$. We divide weight k into two parts,

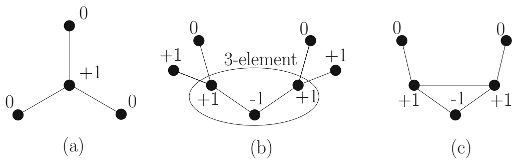


Fig. 4. (a)(b) The two conditions for weight +1. (c) A minus dominating function of five vertices with weight 1.

such that one part contains any vertex labeled with $+1$, whose neighbors are all not labeled with -1, and the other part contains any vertex labeled with $+1$, which has a neighbor labeled with -1. For the first part, If every single value one of the weight k is from this part, each value one contains at most four vertices see Figure 4(a), let V_1 be a set contains these vertices, we have $n \leq 4k$. Second, let v be a vertex labeled with value -1, it is clear that v has at least two neighbors labeled with value $+1$, and the distance to any other vertex labeled with -1 is at least three. Thus we obtain a 3-*element*, which contains v labeled with value -1 and its two neighbors see Figure 4(b), which labeled with $+1$. Moreover, for any pair of such subsets, their intersection is empty. Let u be the vertices labeled with $+1$, which is neighboring 3-element, it is clear that u belongs to a 3-element or to V_1. Next we consider the vertices labeled with 0, which connect to 3-element. We can observe that the vertices which labeled with 0 and connect to vertex labeled with -1 in 3-element, the vertices either the vertex connect to $+1$ in 3-element or the vertex labeled 0 in V_1. So the second part contains at most five vertices. If every single value of the weight k is from the second part, we have $n \leq 5k$. Finally, we can observe that all the vertices are concerned in the above two part. Hence, we can obtain a kernel of size $5k$ as an upper bound for the MD problem on subcubic graphs. Furthermore, For a graph with five vertices labeled as in Figure 4(c) is $k = 1$ and $n = 5$, so the bound of kernel is tight. □

According to this lemma, a naïve FPT-algorithm can be easily obtained via the brute-force method which runs in $O^*(3^{5k}) = O^*(243^k)$ time. Contrary to the brute-force algorithm, we claim that our FPT-algorithm runs in time $O^*(2.3761^{5k}) = O^*(75.7397^k)$, which is a great improvement. Now, our FPT-algorithm is presented in the following theorem. We first give the detailed algorithm and its correctness proof, and then analyze its time complexity.

Theorem 4. *The MD problem for subcubic graphs G can be solved in $O^*(2.3761^{5k})$ time, where k is the MD number of G.*

Proof. Due to Lemma 4, we only need to consider the given subcubic graph G with kernel size of $5k$. Since disconnected components of a graph can be handled separately, we assume that the given graph G is connected in the following context. We also use $N[\cdot]$ to represent $N_G[\cdot]$ for simplicity.

The details of our algorithm are as follows. In our algorithm, we grow a potential optimal minus dominating set D incrementally, where the *minus dominating set D* is a subset of vertices in V labeled with values $+1, 0$ or -1. The *label* of a vertex is the value of vertex assigned by a specific minus dominating function of G. A vertex is called *labeled* if it has been assigned a value; otherwise, it is called *unlabeled*. The weight for the closed neighborhood $N[v]$ of a labeled vertex v is called *valid* (resp. *invalid*) if all the vertices in $N[v]$ are labeled, and the sum of weights of the vertices in $N[v]$ is positive (resp. non-positive). In the process, we maintain a list L of unlabeled vertices which are the neighboring vertices of the currently labeled vertices in D. Initially, L is set to contain one degree-3 vertex of the input graph G, and $D = \emptyset$. During each iteration of our algorithm, we select an arbitrary unlabeled vertex y from list L as the focus vertex, and we assume that x is a labeled vertex adjacent to y in $D \cap N[y]$. We set $\Delta = N[N[y]] \setminus D$. Then our algorithm makes execution branches on all different ways of assigning values to vertices in Δ of new unlabeled vertices in $N[N[y]] \setminus D$, we performing detailed case analysis from *Case 1* to *Case 15*. In the case analysis, our algorithm makes subsequent recursions only on those feasible ways of value assignment in the corresponding cases. For each of such subsequent recursions, the vertices of Δ, which have been labeled, are added into D, resulting in a larger labeled dominating set $D + \Delta$. Then for each vertex v in $D + \Delta$, if all the vertices in $N[v]$ are labeled, we then check whether the weight for $N[v]$ is valid. If we reach any weight for closed neighborhood of a vertex, which is invalid, then the current execution branch is aborted; otherwise, we proceed to update D and L for the next round of execution. We update D by setting $D = D + \Delta$, and then we update list L accordingly by visiting the neighboring vertices of the neighbors of vertices in Δ. More precisely, the vertices in Δ are removed from L, and the unlabeled vertices in the neighborhoods of vertices in Δ are added into L. It is clear that such an update takes only $O(1)$ time. We then proceed to the next execution round with the updated D and L as parameters. We repeat such a selection step until all vertices in G are labeled, that is, L becomes empty. Thus we obtain a candidate minus dominating set D.

In the selection process, we enumerate and store all possible candidates of D according to the above recursive procedure. When all branches of the selection process finished, we obtain a set of candidate minus dominating sets D for the input graph G. We choose the one with minimum weight among all these candidates. This completes our algorithm.

Case analysis for selection step. We divide the analysis into two parts: the initial step and the general selection step.

The initial step. We choose one degree-3 vertex v is placed in D. Then add one neighbor u of v is subsequently into D. In these two beginning steps, there are 8 choices to labeled vertices u and v, since u and v cannot both be of value -1. In any subsequent step, we focus on an unlabeled vertex y, which has a neighbor x in D. Now the degree of x and y can be one, two or three. However, it is easy to see that the worst case happens when both degree of x and y are three. We only need to perform the detailed case analysis for such a case. Thus in the following, we assume that the degree of both x and y are three. Due to the above initial step, we know that x must have another neighbor x_1 in D. Thus we have in total 15 cases to analyze by considering which vertices of $N[N[y]]$ lie in d. See Figure 5.

The selection step. We analyze all possible labeling ways to find the optimal minus dominating function. Due to lack of space, we only provide the analysis of *Case 1* (Figure 5(a)) in the following. The analysis of *Cases 2* to *15* are omitted.

Case 1. Let x_2 be the third neighbor of x, let y_1 and y_2 be the other two neighbors of y, and let z_1, z_2 be the neighbors of y_1, z_3, z_4 be the neighbors of y_2. Since y_1 and y_2 are symmetric, we need to consider 15 cases depending on the content of $\varDelta$, the set of unlabeled vertices in the subgraph of G. See Figures 5(a) to (o). $\varDelta = \{y, x_2, y_1, y_2, z_1, z_2, z_3, z_4\}$. That is, $\{x, x_1\} \subset D$. See Figure 5(a).

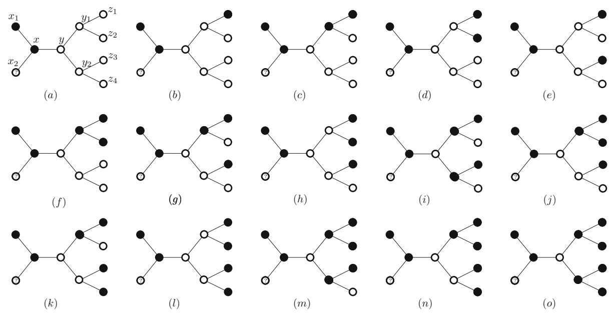


Fig. 5. This figure shows the 15 cases, where the labeled vertices in D are drawn as black disks, the unlabeled vertices in $\varDelta$ as circles, and the question mark in circle is where we analyze whether the vertex is labeled or unlabeled in this case

In order to reduce the number of cases we need to consider under a specific case, we only need to consider the worst-case scenarios in each of the cases. For such a purpose, we make the following three assumptions. We remark that the three assumptions will be applied to all the subsequent cases in this proof.

(i) If there is an edge not in the subgraph, let u, v be the endpoints. Then we can add two vertices u' and v', where we connect u to u' and connect v to v'. Thus we can label the subgraph with constrain : the value of u and v' is same, so as v and u'. We can observe that the number of feasible ways of labeling the vertices with constrain is less than the vertices without constrains, which we will consider in other cases. Hence, we do not need to consider adding some edges in subgraphs.

(ii) We observe that a vertex labeled with -1 need to have at least two vertices labeled with $+1$ as its neighbors, and a vertex labeled with $+1$ can have vertices labeled with $+1, 0$ or -1 as its neighbors. Thus for a specific case, the worst-case scenario for the number of feasible ways to label the vertices in $\varDelta$ is when the labeled vertices for the specific case (for instance, vertices x and y in *Case 1*) are all assigned value $+1$.

(iii) We observe that, for a graph G, if there is a connected graph H is delete a vertex v from G, the number choices to labeling vertices in H are less than or equal to number of choices of graph to labeling G with v assigned value $+1$. Hence, we do not need to consider the subgraph which delete some vertices.

For *Case 1*, according to assumption (i), we suppose that there is no edge connecting any pair of vertices in the subgraph, and according to assumption (ii), we suppose that x and x_1 are labeled with $+1$. We have two situations depending on whether x_2 is labeled. In *Case 1*(a), we first consider x_2 is unlabeled, thus x_2 has at most 3 choices for its labeling, say with value -1, 0 or $+1$. Now we focus on vertex y_1. First we consider to label y_1 with value -1, then $\{z_1, z_2\}$ has at most 3 choices for assigning, and $\{y_2, z_3, z_4$ has at most 14 choices. Thus there are at most 42 choices if y_1 is labeled with -1. Second, we consider to label y_1 with value 0, then $\{z_1, z_2\}$ has at most 6 choices for assigning the values, and $\{y_2, z_3, z_4\}$ has at most 17 choices; hence, there are at most 102 choice if y_1 is labeled 0. Lastly, we consider to label y_1 with value $+1$, then $\{z_1, z_2\}$ has at most 8 choices for assigning the values, and $\{y_2, z_3, z_4\}$ has at most 17 choices. Thus there are at most 136 choice if y_1 is labeled $+1$. After all of $x_2, y_1, y_2, z_1, z_2, z_3, z_4$ are labeled, it is clear that we can label vertex y with an unique minimum value, such that the weights for neighborhoods of x, y, y_1 and y_2 are positive, respectively. By multiplying with the three choices for the value of x_2. Thus there are at most $3 \times (42+102+136) = 840$ feasible ways in total to label the eight vertices in Δ for *Case 1*(a).

For *Case 1*(b), where x_2 is labeled, we use similar argument as the analysis for *Case 1*(a). But we do not need to multiply three choices for the value of x_2. Thus there are at most 240 feasible ways in total to label the seven vertices in Δ for *Case 1*(b). This finishes the analysis of *Case 1*.

In the above detailed analysis, we obtain the recurrence relation $T(n) \leq 840(n-8) + O(1)$ for the worst-case running time of *Case 1*(a). By analyzing all 15 cases in Figure 5 and solving the corresponding recurrence relations, we thus have the worst-case running time for the whole algorithm, which occurs at *Case 5*(a) (see Figure 5(e)). Hence we obtain that the total running time of our algorithm is $T(n) = O^*(2.3761^n) = O^*(2.3761^{5k})$. □

References

1. Alimonti, P., Kann, V.: Hardness of approximating problems on cubic graphs. In: Proc. 3rd Italian Conference on Algorithms and Complexity (CIAC), pp. 288–298 (1997)
2. Damaschke, P.: Minus domination in small-degree graphs. Discrete Applied Mathematics **108**(1–2), 53–64 (2001)
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity, 3rd edn. Springer (1999)
4. Dunbar, J., Goddard, W., Hedetniemi, S., McRae, A., Henning, M.A.: The algorithmic complexity of minus domination in graphs. Discrete Applied Mathematics **68**(1–2), 73–84 (1996)
5. Dunbar, J., Hedetniemi, S., McRae, A., Henning, M.A.: Minus domination in graphs. Discrete Applied Mathematics **199**(1–3), 35–47 (1999)
6. Faria, L., Hon, W.-K., Kloks, T., Liu, H.-H., Wang, T.-M., Wang, Y.-L.: On complexities of minus domination. In: Widmayer, P., Xu, Y., Zhu, B. (eds.) COCOA 2013. LNCS, vol. 8287, pp. 178–189. Springer, Heidelberg (2013)
7. Lichtenstein, D.: Planar formulae and their uses. SIAM Journal on Computing **11**(2), 329–343 (1982)
8. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and System Sciences **43**(3), 425–440 (1991)
9. Zheng, Y., Wang, J., Feng, Q., Chen, J.: FPT results for signed domination. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 572–583. Springer, Heidelberg (2012)

Time-Space Tradeoffs for Dynamic Programming Algorithms in Trees and Bounded Treewidth Graphs

Niranka Banerjee[1], Sankardeep Chakraborty[1], Venkatesh Raman[1](✉), Sasanka Roy[2], and Saket Saurabh[1]

[1] The Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai 600 113, India
{nirankab,sankardeep,vraman,saket}@imsc.res.in
[2] Chennai Mathematical Institute, H1, SIPCOT IT Park, Siruseri, Chennai 603 103, India
sasanka@cmi.ac.in

Abstract. The well-known Courcelle's theorem states that many graph properties (that are expressible in monadic second order logic) can be solved in linear time on graphs of bounded treewidth. Logspace versions of this using automata theoretic framework are also known. In this paper, we develop an alternate methodology using the standard table-based dynamic programming approach to give a space efficient version of Courcelle's theorem. We assume that the given graph and its tree decomposition are given in a read-only memory. Our algorithms use the recently developed stack-compression machinery and the classical framework of Borie *et al.* to develop time-space tradeoffs for dynamic programming algorithms that use $\mathcal{O}(p \log_p n)$ variables where $2 \leq p \leq n$ is a parameter. En route we also generalize the stack compression framework to a broader class of algorithms, which we believe can be of independent interest.

1 Introduction

The aim of this paper is to demonstrate the power of two class of algorithms – one classical [16] and one recent [10] – and to show that a combination of them provides simple space efficient algorithms for graphs of bounded treewidth even when the tree of the decomposition is given in a read-only memory.

It is well-known [14] that many problems that are NP-hard on general graphs can be solved in linear time, using dynamic programming, on trees and graphs of bounded treewidth. This is captured by the most general theorem due to Courcelle [19] which states that properties that can be expressed in Counting Monadic Second Order Logic (CMSO) can be tested in linear time on graphs of bounded treewidth. Aspvall et al. [9] considered the space required by such dynamic programming algorithms, and showed that the entire computation through a simple post-order traversal requires only $\mathcal{O}(w)$ tables (each of size $f(t)$ for some function

DOI: 10.1007/978-3-319-21398-9_28

f of the treewidth t) where w is the pathwidth of the tree of the tree decomposition. In particular, as the pathwidth of a tree on n nodes is $\mathcal{O}(\log n)$, this implies that $\mathcal{O}(\log n)$ tables are sufficient. Bodlaender and Telle [15] later show that a set that realizes the optimum solution can be determined in $\mathcal{O}(n \log_s n)$ time using an additional $\mathcal{O}(s)$ words of space where $s \leq \sqrt{cn/2}$ is a parameter where c is an upper bound on the degree of a vertex in the tree decomposition (see Definition 1 in Section 4).

Our main objective in this paper is to implement these standard dynamic programming algorithms in the read-only memory model where the input tree is given in a read-only input. Apart from adjacency of the graph, we can only access the leftmost child, right sibling and the bags of each node in constant time (in particular, we do not assume parent pointers which poses a challenge when doing a bottom-up traversal). We show that the above bounds can be achieved if we use an additional stack that could grow to linear size. Our next step is then to use the recent method of Barba et al.[10] to reduce the space to $\mathcal{O}(\log n)$ words, without too much degradation in time. Towards this end, we first generalize the stack-compression method to a broader class of stack algorithms.

Related Work. Elberfeld et al.[24] showed that Bodlaender's [12] linear time algorithm (to determine whether a given graph has treewidth at most k for a fixed k) and Courcelle's [19] linear time algorithm (to determine the satisfiability of a CMSO expressible property on a bounded treewidth graph) can be implemented in $\mathcal{O}(1)$ words, i.e. using logarithmic bits of space. However, the algorithm of Elberfeld et al. [24] is reasonably complicated, and is based on the automata theoretic framework of Courcelle's theorem, while we use the natural table based approach of the dynamic programming algorithms.

Recently there has been a spate of work in space efficient graph and geometric algorithms [17,23,7,5,2,27,22,11,6,3] due to their theoretical interest and practical motivation for implementation in small hand held devices [4]. For example, for the Maximum Independent Set problem covered by our main result in this paper, Bhattacharya et al. [11] gave an $\mathcal{O}(n)$ algorithm using $\mathcal{O}(h)$ word space; here h is the height of the tree that could be as high as n. In contrast, our algorithm gives the optimum value in $\mathcal{O}(n)$ time and $\mathcal{O}(n^\epsilon)$ words of space even for weighted trees and bounded treewidth graphs. It can also output a set realizing the optimum value in the same time and space for unweighted trees. But for weighted trees and bounded treewidth graphs, reporting a solution set takes $\mathcal{O}(n^2)$ time using the same space. We also have algorithms that take $\mathcal{O}(\log n)$ words of space and $\mathcal{O}(n^{1+\epsilon})$ time to output the optimum value. Here ϵ is any fixed positive constant less than 1.

Our results, apart from adding to the growing body of literature on space efficient graph algorithms, bring to light the recent technique to reduce the space requirement of algorithms that use only a stack (that could grow to linear size) and a constant number of auxiliary words.

Organization of the Paper. In Section 2, we first explain the general working scheme of stack based algorithms and also state the main lemma from [10].

Then we provide our generalization of the stack based algorithms, and prove our stack compression lemma for the generalization. In Section 3 we provide algorithms for specific problems on trees using the stack compression machinery, as a warm up for our general result. In Section 4 we develop our main space efficient algorithm for problems expressible in CMSO. We also consider the weighted versions and the modifications necessary to output the solution set. Section 5 contains some concluding remarks.

Model. Our algorithms work in read-only memory model where the input elements cannot be modified or moved. We believe that read-only memory model is a natural model to study algorithms on graphs. Our algorithms work with the standard left-most child, right-sibling based representation for rooted trees [18] (this is slightly weaker in contrast to the doubly connected edge list representation used in [8,11]). I.e. we assume that the tree (of the input or of the tree-decomposition is given in a representation where the children of a node are organized in a linked list. I.e. given a node label, we can find the bag associated with it, its left-most child and its right sibling in constant time. In particular, we do not have parent pointers associated with the nodes, unless stated otherwise.

For the weighted versions of the algorithms and for graphs of bounded treewidth, we assume that weights or the bag sets of the vertices are given in a separate array indexed by the labels of the vertices of the tree (which we assume are in $\{1, 2, \ldots n\}$). For bounded treewidth graphs, apart from the tree decomposition, we also need the graph in read-only memory, represented in a way that adjacency can be checked in constant time. As for output, the algorithms will output values during the execution in a write once array, which cannot be accessed by the algorithm after reporting. We also denote the set of natural numbers by $\mathbb{N}$. When measuring extra space, we count the number of variables or words used by the algorithm, in addition to the input in read-only memory. We assume that standard RAM computations (including additions, multiplications and logical operations) can be done in constant time.

2 Generalized Stack Framework

In what follows, we explain the general scheme of stack based algorithms defined in [10]. Let $\mathcal{A}$ be a class of deterministic algorithms which uses a stack and optionally other auxiliary data structures of constant size. The operations that can be supported are push, pop and accessing the k topmost elements of the stack for a constant k.

Let $X \in \mathcal{A}$ be any algorithm and $a_1, a_2, \cdots, a_n \in I$ be the values of the input, in the order in which they are treated by $\mathcal{A}$. We only assume that given a_i, we can access the next input element in constant time. We will call X a stack based algorithm if, given a set of ordered input I, X processes $a_i \in I$ one by one in order and based on a_i, the top k elements of the stack and the auxiliary data structures' current configuration, it decides to pop some elements off the stack first then push i (or some function of i) along with some constant words of information into the stack. Then the computation moves forward to the next element, until all the elements are exhausted. The final result which is stored in the stack, is then output by popping them one by one until the stack is empty. Any algorithm following this structure is called a *stack based algorithm*. We provide the pseudocode below for completeness [10].

Algorithm 1. Basic scheme of a stack based algorithm

```
1: Initialize stack and auxiliary data structure DS with $\mathcal{O}(1)$ elements from $I$
2: for all subsequent input $a \in I$ do
3:     while some-condition(a, DS, STACK.TOP(1),$\cdots$, STACK.TOP($k$)) do
4:         STACK.POP
5:     end while
6:     if another-condition(a, DS, STACK.TOP(1),$\cdots$, STACK.TOP($k$)) then
7:         STACK.PUSH(a)
8:     end if
9: end for
10: Report(STACK)
```

For such stack based algorithms, in [10] they prove the following result.

Lemma 1 (Theorem 2, [10]). *Any stack algorithm which takes $\mathcal{O}(n)$ time and $\Theta(n)$ space can be adapted so that, for any parameter $2 \le p \le n$, it solves the problem in $\mathcal{O}(n^{1+(1/\log p)})$ time using $\mathcal{O}(p \log_p n)$ variables.*

The main idea of their proof is to divide the input into blocks in the order in which the input is processed. The size of the block depends on the allowed work space. During the execution of the stack algorithm, we store only the first and the last elements of the block which was pushed into the stack (except up to two top blocks that are stored in full). While simulating the algorithm, if we ever need any value not stored in stack, we invoke a reconstruction operation to generate the block containing that element and proceed. For efficiently supporting the reconstruction operation, we also store the content of the auxiliary data structure just after the first element of each block is pushed into the stack.

Our first observation is that we can generalize the above stack based algorithms to a broader class of algorithms and still prove a version of Lemma 1. More specifically,

- while we continue to assume that there can be at most one push while processing an element, we can let it precede and succeed with a sequence of pops (the original framework only allowed it to be preceded with pops.)
- We allow the number of auxiliary variables to be a parameter t, which need not be a constant, and any of the auxiliary variables can be accessed in a constant time. This is possible if we simply assume that the auxiliary variables form an array of t elements. We then capture the time and space also as a function of the number of auxiliary variables.

– We assume that the next element to be processed could depend on the pushes and pops done and hence could take more than a fixed constant time to determine. In fact, as the input is in read-only memory, the algorithm may make several scans over the input (along with the stack and auxiliary storage) and take, say some $g(n,t)$ time (as k is a constant, we ignore its dependence in g) to find the next element to process.
– We will also assume that each element is processed at most once.
– Finally, we allow the condition that decides what to push into the stack or whether to pop the stack element to take time an arbitrary function $h(n,t)$ of n and t and space $s(n)$ words.

Hence each push and pop can take $h(n,t)$ time, and hence the entire algorithm takes $\mathcal{O}(nh(n,t) + ng(n,t))$ time using $s(n) + t$ words apart from a stack. We also allow the algorithm to output elements to a write-once output array along the way. We call such a stack algorithm, a generalized stack based algorithm. We give the pseudocode below.

Algorithm 2. Basic scheme of a generalized stack based algorithm

```
1: Initialize the stack and auxiliary data structure DS of size $t$
2: Initialize the first input element $a$ to be processed.
3: repeat
4:     while some-condition(a, input, DS, STACK.TOP(1),$\cdots$, STACK.TOP($k$)) do
5:         STACK.POP
6:     end while
7:     if some-condition(a, input, DS, STACK.TOP(1),$\cdots$, STACK.TOP($k$)) then
8:         STACK.PUSH(a)
9:     end if
10:    while some-condition(a, input, DS, STACK.TOP(1),$\cdots$, STACK.TOP($k$)) do
11:        STACK.POP
12:    end while
13:    $a \leftarrow next(a)$ ($next(a)$ is computed based on $a$, all the pops and push done in
       this step as well as the input set)
14: until $a$ is NULL
```

For such generalized stack based algorithms, we can prove the following theorem:

Theorem 1. *Any generalized stack algorithm that takes $\mathcal{O}(n(h(n,t) + g(n,t)))$ time and $\mathcal{O}(n + t + s(n))$ space can be adapted so that, for any parameter $2 \leq p \leq n$, it solves the problem in $\mathcal{O}((h(n,t) + g(n,t))n^{1+(1/\log p)})$ time using $\mathcal{O}(pt \log_p n + s(n))$ variables, where $\mathcal{O}(n)$ space is for the explicit stack and $\mathcal{O}(t)$ space is for auxiliary data structures, $s(n)$ space and $h(n,t)$ time to check the conditions for pushes and pops, and $g(n,t)$ time to find the next element to process.*

Proof. (*Sketch*) The space for auxiliary data structures plays a role in Lemma 1 when we compress a block in the stack, and store the 'context' of the auxiliary data structures with the first element of the block. In our case, we can store the values of the t variables into the stack, thereby increasing the storage space by a factor of t.

Also the reconstruction steps in Lemma 1 simulate the original stack algorithm repeatedly (sometimes with recursive calls), and hence the ability to compute the next element of an element after processing that element (which could take more than a constant time) does not affect the analysis in any way. At the base case of the recursive step of reconstruction, a block of size p is reconstructed with full stack space of size p which takes $\mathcal{O}(p(g(n,t)+h(n,t)))$ time (as only the elements of that block are pushed and popped), so $T(p) = \mathcal{O}(p(g(n,t)+h(n,t)))$. The general step remains as before resulting in the earlier recurrence for the total running time as $T(n) = 2pT(\frac{n}{p}) + \mathcal{O}(p)$ which solves to the claimed runtime bound. □

3 Algorithms on Rooted Trees

In the next section we give a generic algorithm for problems expressible in CMSO over graphs of bounded treewidth. In this section we explain our algorithms for specific problems on rooted trees. Apart from serving as a warm up to the general result, this serves to illustrate the main idea that to obtain a space efficient algorithm, all we need to do is to turn a natural dynamic programming based algorithm into a stack based algorithm and then use Theorem 1 to obtain the desired trade-off between space and time. Thus, these algorithms are useful in practice. We exemplify our approach by giving algorithms for Minimum Vertex Cover, Maximum Independent Set and Minimum Weighted Dominating Set (MWDS).

Theorem 2. Minimum Vertex Cover *and* Maximum Independent Set *can be solved on a rooted tree on n vertices in $\mathcal{O}(n^{1+(1/\log p)})$ time using $\mathcal{O}(p\log_p n)$ variables of extra space where $2 \le p \le n$. We can also output the corresponding set for each of the above problems in the same time and space. The output is generated through an output array (which cannot be seen/used by the algorithm once the output has been generated) which will list out the vertices.*

Proof. We explore the tree in a depth first fashion by pushing elements into the stack as we traverse until we find the leaf. After we compute the values at the leaf, we pop the stack, and we transfer its value to its parent (which is at the top of the stack). Thus we get around the lack of parent pointer by pushing elements into the stack as we traverse (and so the parent is at the top of the stack when an element is popped), but then this lets the stack grow to size n. Then we apply the stack compression of Theorem 1 to reduce the space. First notice that this algorithm as we described fits into our generalized stack framework (with t, $g(n,t)$, $h(n,t)$ and $s(n)$ as constants) and hence Theorem 1 can be applied. Note that we require the generalized stack framework as the next vertex to be processed in the depth first search is a function of the pushes and pops and hence may take more than a constant time. Also in the depth first search, sometimes a push is followed by the pops (which wasn't allowed in the original stack framework). Below we give specific details of the algorithm for the problems claimed in the theorem.

Minimum Vertex Cover: A standard algorithm to find a minimum vertex cover in a tree is based on the observation that if v is a vertex with degree 1, then there exists a minimum vertex cover that contains v's unique neighbor u.

Hence the algorithm repeatedly includes the neighbor of leaf nodes into the solution and deletes them (and their leaf neighbors) from the tree, until the tree becomes empty. This algorithm can be implemented using a stack as follows: we do a depth first traversal of the tree starting from the root, including every vertex into the stack along with a bit 0 associated with the vertex (to indicate our initial guess to exclude that vertex from the solution) as we visit. The first popping happens when we visit a leaf. While popping out a vertex (after visiting its entire subtree), if its value remains as 0, we make its parent (which would be at the top of the stack) 1. Once a node becomes 1, it stays as 1 during its lifetime in the stack. While popping a node if its associated bit is 1, we output it in the output array.

Clearly this implements the above algorithm as we pop only elements that are leaves or have become leaves after some deletion of vertices and before throwing when a leaf has value 0, we make its parent 1.

Depth first order can be implemented by accessing a node's leftmost child to push at every step. Once we know that a node has no children, then we find its right sibling (the next child of its parent to visit in the depth first order) to add to stack before popping the node. This completes the algorithm, and it is easy to see that this algorithm essentially implements the algorithm outlined above using a stack and a constant number of variables. When a vertex assigned 0 is popped it implies that it is a leaf and hence its parent is assigned 1 which is correct by the observation above. Maximum Independent Set can be solved similarly (by switching the role of 1 and 0 in the above algorithm). □

A Weighted Case. The algorithms we described for Minimum Vertex Cover use the observation that for any leaf, its unique neighbor can be picked up in any minimum vertex cover. This is no longer true in the weighted variants of these problems (see Appendix for definitions). Thus, in this case we resort to a modification of standard dynamic programming algorithm. We exemplify the approach via Minimum Weighted Dominating Set (MWDS). However, this comes at a cost: now we cannot output the desired set in the same running time, though it is possible to output the set with a linear factor increase in the runtime.

Theorem 3. MWDS *can be solved, on a rooted tree with n vertices, in time $\mathcal{O}(n^{1+(1/\log p)})$ using $\mathcal{O}(p \log_p n)$ variables of extra space where $2 \leq p \leq n$.*

Proof. For MWDS the standard dynamic programming algorithm (see [26] for an algorithm for MWDS in bounded treewidth graphs) needs the values at each child of a node before computing the value of the node, which may result in some non-trivial storage at each node of the tree. A dynamic programming algorithm works as follows. It computes four quantities $I(v), D(v), DE(v), E(v)$ for each vertex v, which are defined as below.

1. $E(v) =$ The size of a minimum dominating set of the subtree rooted at v, that does not contain the vertex v.
2. $DE(v) =$ The size of a minimum dominating set of the subtree rooted at v that dominates all vertices of the subtree except possibly v.
3. $I(v) =$ The size of a minimum dominating set of the subtree rooted at v that contains vertex v.
4. $D(v) =$ The size of a minimum dominating set of the subtree rooted at v.

Clearly $D(v) = \min\{I(v), E(v)\}$. It is easy to find these values for leaf nodes, and once we have computed these quantities for all the children of a node v, they can be computed for v as follows: let $u_1, u_2, \dots u_d$ be the children of v. Let $H(v) = \sum_{i=1}^{d} D(u_i)$. Then
$I(v) = w(v) + \sum_{i=1}^{d} DE(u_i)$, $DE(v) = \min\{I(v), H(v)\}$,
$E(v) = \min_{1 \le i \le d}\{I(u_i) + \sum_{k=1, k \ne i}^{d} D(u_k)\}$

Now, all these quantities can be computed in a depth first order as follows. We initialize $DE(u), H(u)$ and $E(u)$ to 0, and $I(u)$ to $w(u)$, as we push a vertex u into the stack. Once we pop a vertex u, we would have visited its entire subtree, and hence computed these values for that vertex. We can then easily compute $D(u)$ for that vertex. Then we update its parent (which is at the top of the stack) with its contribution to all of these quantities of its parent. More precisely, for the parent vertex v, we make $I(v) \leftarrow I(v) + DE(u); H(v) \leftarrow H(v) + D(u)$.

Updating $E(v)$ is a bit tricky, but it becomes easier, if we rewrite the quantity $E(v)$ in the above equation as,

$$E(v) = \left\{ \sum_{i=1}^{d} D(u_i) + \min_{1 \le i \le d} (I(u_i) - D(u_i)) \right\} = H(v) + \min_{1 \le i \le d} (I(u_i) - D(u_i)).$$

To compute $H(v)$, we simply keep track of $\min\{I(u) - D(u)\}$ over its children u of v and also the vertex u that realizes the minimum. When we pop a vertex u, we update the DE value of its parent v as $H(v) = H(v) + D(u)$ and update the $\min\{I(u) - D(u)\}$ at its parent by $(I(u) - D(u))$ if the $(I(u) - D(u))$ quantity is smaller. When all the children are popped, $E(v)$ is set to $E(v) = H(v) + (I(u) - D(u))$. It is easy to see that the above expression correctly computes the four quantities in a postorder traversal using just a stack and the final answer is given by $D(root)$. As the algorithm visits the tree in postorder traversal and uses just a stack, by similar argument to that of Theorem 2, the result follows using Theorem 1. □

4 Algorithms for Graphs of Bounded Treewidth

In this section we design space efficient version of optimization variant of Courcelle's Theorem. We follow the proof of Borie *et al.* [16] and use the machinery of stack compression to obtain the desired theorem. We start with the notations and the language in which we will be working with.

Treewidth. For a rooted tree T and a non-root node $t \in V(T)$, by parent(t) we denote the parent of t in the tree T. For two nodes $u, t \in T$, we say that u is a *descendant* of t, denoted $u \preceq t$, if t lies on the unique path connecting u to the root. Note that every node is thus its own descendant.

Definition 1 (tree decomposition). *A* tree decomposition *of a graph G is a pair (T, β), where T is a rooted tree and $\beta : V(T) \to 2^{V(G)}$ is a mapping such that:*

- *for each node $v \in V(G)$ the set $\{t \in V(G) | v \in \beta(t)\}$ induces a nonempty and connected subtree of T,*
- *for each edge $e \in E(G)$ there exists $t \in V(T)$ such that $e \subseteq \beta(t)$.*

The set $\beta(t)$ is called the *bag at* t, while sets $\beta(u) \cap \beta(v)$ for $uv \in E(T)$ are called *adhesions*. Following the notation from [25], for a tree decomposition (T, β) of a graph G we define auxiliary mappings $\sigma, \gamma : V(T) \to 2^{V(G)}$ as

$$\sigma(t) = \begin{cases} \emptyset & \text{if t is the root of } T \\ \beta(t) \cap \beta(\text{parent}(t)) & \text{otherwise} \end{cases}$$

$$\gamma(t) = \bigcup_{u \preceq t} \beta(u)$$

We now define a class of graph optimization problems, called MIN/MAX-CMSO$[\psi]$, with one problem for each CMSO sentence ψ on graphs, where ψ has a free vertex (edge) set variable S. We refer to [1,19,20] and the book of Courcelle and Engelfriet [21] for a detailed introduction to CMSO. In [21], CMSO is referred to as CMS_2. The MIN-CMSO problem defined by ψ is denoted by MIN-CMSO$[\psi]$ and defined as follows.

MIN-CMSO$[\psi]$
Input: A graph $G = (V, E)$.
Question: Find the cardinality of a minimum sized subset $S \subseteq V$ ($S \subseteq E$) (if exists) such that $(G, S) \models \psi$.

The definition of MAX-CMSO$[\psi]$ problem is analogous to the MIN-CMSO$[\psi]$ problem. The only difference is that now we try to find a maximum sized subset $S \subseteq V$. Here, we only give an algorithm for MIN/MAX-CMSO$[\psi]$ problems when S is a vertex subset. All of our results can be extended to edge setting in a straightforward way. In particular, an edge set problem on graph $G = (V, E)$ can be transformed to a vertex subset problem on the edge-vertex incidence graph $I(G)$ of G, which is a bipartite graph with vertex bipartition's V and E with edges between vertices $v \in V$ and $e \in E$ if and only if v is incident with e in G. It is well-known that the treewidth of G and $I(G)$ only differ by a factor of 2. To make the translation work in the proof, it is sufficient to use the fact that the property of being an incidence graph of a graph G is expressible in CMSO. From now on *we only concentrate on* MIN/MAX-CMSO$[\psi]$ *problems defined over vertex subsets.*

Now we give a couple of examples of problems, that can be encoded using MIN/MAX-CMSO$[\psi]$. To express MAXIMUM INDEPENDENT SET we do as follows. Given a graph G and a vertex subset X, a simple constant length formula $\mathbf{indp}(X)$ that verifies that X is an independent set of G is: $\forall_{x \in X} \forall_{y \in X} \neg \mathbf{adj}(x, y)$. Thus we can express MAXIMUM INDEPENDENT SET using the above logical sentence. Let us consider another example, namely, MINIMUM DOMINATING SET. Given a graph G and a vertex subset X, a simple constant length formula $\mathbf{dom}(X)$ that verifies that X is a dominating set of G is: $\forall_{x \in V(G)}[x \in X \vee \exists_{y \in X} \mathbf{adj}(x, y)]$.

Dynamic Programming Algorithms over Tree Decompositions. The standard dynamic programming algorithms on bounded treewidth graphs for standard optimization problems, see [13,1,14] proceed by doing a bottom-up traversal of the tree computing tables at every node starting from the leaves. Aspvall et al., in [9] identify the space requirement of the algorithms and argue that $\mathcal{O}(\log n)$ 'open tables' are sufficient in the bottom-up implementation. Bodlaender and Telle [15], building on the work of Aspvall et al., claim that any property expressible in monadic second order logic can be implemented using $O(\log n)$ tables in the bottom-up traversal.

However, implementing this in the read-only memory model without parent pointers, provide challenges in terms of space, and that is our task in this section.

In particular, we provide an alternate $\mathcal{O}(\log n)$ word space algorithm that precludes the need for parent pointers in the input representation, and uses the stack compression machinery recently developed by Barba et al. [10] to prove the following theorem.

Theorem 4. (♠)[1] *Let G be a graph given with a tree decomposition $(T = (V_T, E_T), \beta)$ of width k. Then* MIN/MAX-CMSO$[\psi]$ *can be solved in time $\mathcal{O}(\tau(k) \cdot n^{1+(1/\log p)})$ time and $\mathcal{O}(\tau(k) \cdot p \log_p n)$ space algorithm, for any parameter $2 \le p \le n$. Here, $|V| = n$ and τ is a function of k alone.*

In fact we prove a weighted variant of Theorem 4. We also obtain an algorithm that not only outputs the weight of a value of a maximum weighted subset (or a minimum weighted subset) S such that $(G, S) \models \psi$, but also the set S. We call this version of the problem CONSTRUCTIVE-WEIGHTED-MIN/MAX-CMSO$[\psi]$.

Theorem 5. (♠) *Let G be a graph given with a tree decomposition $(T = (V_T, E_T), \beta)$ of width k. Then* CONSTRUCTIVE-WEIGHTED-MIN/MAX-CMSO$[\psi]$ *can be solved in time $\mathcal{O}(\tau(k) \cdot n^{2+(2/\log p)})$ time and $\mathcal{O}(\tau(k) \cdot p \log_p n)$ space algorithm, for any parameter $2 \le p \le n$. Here, $|V| = n$ and τ is a function of k alone.*

[1] Proofs of results marked with (♠) will appear in full version.

5 Conclusion

We have shown that several optimization problems can be solved on trees and bounded treewidth graphs using logarithmic number of extra variables, in polynomial time even when the input tree is given in a read-only memory. We achieve this by modifying the standard dynamic programming algorithms to use only a stack and using the recent stack compression routine to reduce space. Barba et al. [10] also provide a stack compression scheme that can be used to reduce work space to $\mathcal{O}(1)$ words provided the (full stack) algorithm satisfies what they called a "green" property. The standard dynamic programming algorithms we use are not "green". It would be interesting to see whether our approach can be extended to obtain algorithms using only $\mathcal{O}(1)$ or even $o(\log n)$ words. This would give an alternate $o(\log n)$ (words of) space version of Courcelle's theorem. Another open problem is whether this approach helps to give an alternate logarithmic space version of Bodlaender's theorem [12].

Bodlaender and Telle [15] give a divide and conquer strategy to find the optimum set in $\mathcal{O}(n \log n)$ time (against a naive $O(n^2)$ time) using $\mathcal{O}(\log n)$ words of extra space, when the tree of the tree-decomposition has a constant number of children for each node. Extending this (in the read-only memory model) to the case when each node has an arbitrary number of children, or to obtain a 'nice-tree decomposition' from a general tree-decomposition and implementing their approach in logarithmic (words of) space in read-only memory model, are challenging problems. It would also be interesting to find other applications of our generalized stack compression framework.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. Journal of Algorithms **12**, 308–340 (1991)
2. Asano, T.: Constant-Working-Space Algorithms for Image Processing. In: Nielsen, F. (ed.) ETVC 2008. LNCS, vol. 5416, pp. 268–283. Springer, Heidelberg (2009)
3. Asano, T.: Constant-Working-Space Algorithms: How Fast Can We Solve Problems without Using Any Extra Array? In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 1–1. Springer, Heidelberg (2008)
4. Asano, T.: Designing Algorithms with Limited Work Space. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 1–1. Springer, Heidelberg (2011)
5. Asano, T., Doerr, B.: Memory-constrained algorithms for shortest path problem. In: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, CCCG (2011)
6. Asano, T., Kirkpatrick, D., Nakagawa, K., Watanabe, O.: $\widetilde{O}(\sqrt{n})$-Space and polynomial-time algorithm for planar directed graph reachability. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 45–56. Springer, Heidelberg (2014)
7. Asano, T., Mulzer, W., Rote, G., Wang, Y.: Constant-work-space algorithms for geometric problems. JoCG **2**(1), 46–68 (2011)
8. Asano, T., Mulzer, W., Wang, Y.: Constant-work-space algorithms for shortest paths in trees and simple polygons. J. Graph Algorithms Appl. **15**(5), 569–586 (2011)

9. Aspvall, B., Telle, J.A., Proskurowski, A.: Memory requirements for table computations in partial k-tree algorithms. Algorithmica **27**(3), 382–394 (2000)
10. Barba, L., Korman, M., Langerman, S., Sadakane, K., Silveira, R.: Space-time trade-offs for stack-based algorithms. Algorithmica (2014) (in press)
11. Bhattacharya, B.K., De, M., Nandy, S.C., Roy, S.: Maximum independent set for interval graphs and trees in space efficient models. In: Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG (2014)
12. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996)
13. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**(1–2), 1–45 (1998)
14. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. Comput. J. **51**(3), 255–269 (2008)
15. Bodlaende, H.L., Telle, J.A.: Space-efficient construction variants of dynamic programming. Nord. J. Comput. **11**(4), 374–385 (2004)
16. Borie, R.B.: Gary Parker, R., Tovey, C.A.: Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. Algorithmica **7**(5&6), 555–581 (1992)
17. Bose, P., Morin, P.: An improved algorithm for subdivision traversal without extra storage. Int. J. Comput. Geometry Appl. **12**(4), 297–308 (2002)
18. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
19. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Inform. and Comput. **85**, 12–75 (1990)
20. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, pp. 313–400. World Sci. Publ., River Edge (1997)
21. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic. Cambridge University Press (2012)
22. Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: Planar graph isomorphism is in log-space. In: Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, pp. 203–214 (2009)
23. De, M., Nandy, S.C., Roy, S.: Convex hull and linear programming in read-only setup with limited work-space. CoRR, abs/1212.5353 (2012)
24. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of bodlaender and courcelle. In: 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, pp. 143–152 (2010)
25. Grohe, M., Marx, D.: Structure theorem and isomorphism test for graphs with excluded topological subgraphs. SIAM J. Comput. **44**(1), 114–159 (2015)
26. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
27. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4), 17:1–17:24 (2008)

Reducing Rank of the Adjacency Matrix by Graph Modification

S.M. Meesum[1](✉), Pranabendu Misra[1], and Saket Saurabh[1,2]

[1] Institute of Mathematical Sciences, Chennai, India
{meesum,pranabendu,saket}@imsc.res.in
[2] University of Bergen, Bergen, Norway

Abstract. The main topic of this article is to study a class of graph modification problems. A typical graph modification problem takes as input a graph G, a positive integer k and the objective is to add/delete k vertices (edges) so that the resulting graph belongs to a particular family, $\mathcal{F}$, of graphs. In general the family $\mathcal{F}$ is defined by forbidden subgraph/minor characterization. In this paper rather than taking a structural route to define $\mathcal{F}$, we take algebraic route. More formally, given a fixed positive integer r, we define $\mathcal{F}_r$ as the family of graphs where for each $G \in \mathcal{F}_r$, the rank of the adjacency matrix of G is at most r. Using the family $\mathcal{F}_r$ we initiate algorithmic study, both in classical and parameterized complexity, of following graph modification problems: r-Rank Vertex Deletion, r-Rank Edge Deletion and r-Rank Editing. These problems generalize the classical Vertex Cover problem and a variant of the d-Cluster Editing problem. We first show that all the three problems are NP-Complete. Then we show that these problems are fixed parameter tractable (FPT) by designing an algorithm with running time $2^{\mathcal{O}(k \log r)} n^{\mathcal{O}(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion and r-Rank Editing running in time $2^{\mathcal{O}(f(r)\sqrt{k} \log k)} n^{\mathcal{O}(1)}$. We complement our FPT result by designing polynomial kernels for these problems.

1 Introduction

A typical graph modification problem takes as an input a graph G, a positive integer k and the objective is to add/delete k vertices (edges) so that the resulting graph belongs to a particular family, $\mathcal{F}$, of graphs. One of the classical way to define $\mathcal{F}$ is by defining what is called *graph property*. A *graph property* Π is a set of graphs, which is closed under isomorphism. The property Π is non-trivial if it includes infinitely many graphs and also excludes infinitely many graphs. The property Π is called *hereditary* if for any graph $G \in \Pi$, all induced subgraphs of G are also present in Π. A hereditary property Π has an *induced forbidden set* characterization if there is a family $\mathsf{Forb}(\Pi)$ of graphs such that, a graph G is in Π if and only if no induced subgraph of G is in $\mathsf{Forb}(\Pi)$. Another way of defining $\mathcal{F}$ is by excluding some forbidden *minors*. A graph H is said to be a *minor* of a graph G, if H can be obtained from G be deleting some edges and

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 361–373, 2015.
DOI: 10.1007/978-3-319-21398-9_29

vertices, and contracting some edges. A hereditary property Π has a *forbidden minor* characterization if there is a family $\mathsf{Forb}(\Pi)$ of graphs such that no graph in Π has a graph in $\mathsf{Forb}(\Pi)$ as a minor. For a graph property Π the Π-Graph Modification problem is defined as follows. Given a graph G and a positive integer k, can we delete (edit) at most k vertices (edges) so that the resulting graph G' is in Π?

Lewis and Yannakakis have shown that for any Π which is non-trivial and hereditary, the corresponding vertex deletion problems are NP-Complete [13,18]. In addition for several graph properties, the corresponding edge editing problem are known to be NP-Complete as well [2]. This motivates the study of these problems in algorithmic paradigms that are meant for coping with NP-hardness, such as approximation algorithms and parameterized complexity. In this paper, we study yet another kinds of graph modification problems in the framework of *parameterized complexity*. In this framework, each instance of the problem is *parameterized*, i.e. assigned a number k, which is called the *parameter*, which represents some property of the instance. For example, a typical parameter is the size of the optimum solution to the instance. The problem is called *fixed parameter tractable* (FPT), if a parameterized instance (x, k) of the problem is solvable in time $f(k)n^{\mathcal{O}(1)}$, where n is the size of the instance. A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm that given an instance of the problem, outputs an equivalent instance whose size is bounded by a polynomial in the parameter. It is known that whenever Π has *finite* induced forbidden set characterization (that is, $|\mathsf{Forb}(\Pi)|$ is finite), the corresponding deletion problems are known to be FPT [3]. Similarly, whenever Π is characterized by a *finite* forbidden set of minors, the corresponding problems are FPT by a celebrated result of Robertson and Seymour [16]. These problems include classical problems such as Vertex Cover, Feedback Vertex Set, Split Vertex Deletion. There has also been extensive study of graph modification problems when the corresponding $\mathsf{Forb}(\Pi)$ is not finite, such as Chordal Vertex Deletion, Interval Vertex Deletion, Chordal Completion and Odd Cycle Transversal [4,5,8,15].

In this paper we step aside and study a class of graph modification problems that are defined algebraically rather than structurally. Given a graph G, two most important matrices that can be associated with it are its adjacency matrix A_G or the corresponding laplacian L_G. One could study graph modification problems where after editing edges/vertices, the resulting graph has a certain kind of eigenvalue spectrum or has a certain upper bound on the second eigenvalue or the corresponding adjacency matrix satisfies certain properties. The topic of this paper is one of these kind of problems. In particular we want the adjacency matrix of the resulting graph to be some fixed constant r. To define these problems formally, for a positive integer r, define Π_r to be the set of graphs G such that the rank of the adjacency matrix A_G is at most r. The rank of the adjacency matrix A_G is an important quantity in graph theory. It also has connections to many fundamental graph parameters such as the clique number, diameter, domination number etc. [1]. All these motivate the study of the following problems.

r-Rank Vertex Deletion	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Can we delete at most k vertices from G so that $rank(A_G) \leq r$?	

We use $\triangle$ to denote the standard symmetric difference of sets.

r-Rank Editing	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Can we find a set $F \subseteq V(G) \times V(G)$ of size at most k such that $\mathsf{rank}(A_{G'}) \leq r$, where $G' = (V(G), E(G) \triangle F)$?	

The set F denotes a set of edits to be performed on the graph G. The $\triangle$ operation acts on the sets as follows: if $(u, v) \in F$ and $(u, v) \in E(G)$ then we delete the corresponding edge from G and if $(u, v) \in F$ and $(u, v) \notin E(G)$ then we add the corresponding edge to G. We also consider a variant of r-Rank Editing, called r-Rank Edge Deletion, where we are allowed to only delete edges.

These problems are also related to some well known problems in graph algorithms. Observe that if $\mathsf{rank}(A_G) = 0$, then G is an empty graph and if $\mathsf{rank}(A_G) = 2$ then G is a complete bipartite graph with some isolated vertices. There are no graphs such that $\mathsf{rank}(A_G) = 1$. So for $r = 0$, r-Rank Vertex Deletion is the well known Vertex Cover problem. Similarly for $r = 2$, a solution to r-Rank Edge Deletion is a complement of a solution to Maximum Edge Biclique, where the goal is to find a bi-clique subgraph of the given graph with maximum number of edges [14]. A graph G is called a *Cluster Graph* if every component is a clique. We may also view the above problems as variants of the d-Cluster Vertex Deletion and d-Cluster Editing. In d-Cluster Vertex Deletion, we wish to delete at most k vertices of the graph, so that the resulting graph is a cluster graph with at most d components. Similarly in d-Cluster Editing we wish to add/delete at most k edges to the graph, so that the resulting graph is a cluster graph with at most d components. These problems are known to be NP-hard and they admit FPT algorithms parameterized by the solution size [7,9,10]. In our problems, instead of reducing the graph to a disjoint union of d cliques, we ask that the graph be reduced to a graph of low rank adjacency matrix. In this paper, we obtain following results about these problems.

- We first show that all the three problems are NP-Complete.
- Then we show that these problems are FPT by designing an algorithm with running time $2^{\mathcal{O}(k \log r)} n^{\mathcal{O}(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion and r-Rank Editing running in time $2^{\mathcal{O}(f(r)\sqrt{k} \log k)} n^{\mathcal{O}(1)}$. Observe that the edge-editing problem admits a subexponential FPT algorithm.
- Finally, we design polynomial kernels for these problems.

Our results are based on structural observations on graphs with low rank adjacency matrix and applications of some elementary methods in parameterized complexity.

2 Preliminaries

We use $\mathbb{R}$ and $\mathbb{N}$ to denote the set of reals and integers, respectively. For two sets X and Y, we define $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$, i.e. the set of all elements which are exactly in X or Y but not both. A vector v of length n is an ordered sequence of n values from $\mathbb{R}$. A collection of vectors $\{v_1, v_2, \ldots, v_k\}$ are said to be linearly dependent if there exist values $a_1, a_2, \ldots, a_k$, not all zeros, such that $\sum_{i=1}^{k} a_i v_i = 0$. Otherwise these vectors are called linearly independent. A matrix A of dimension $n \times m$, is a sequence of values (a_{ij}). The i-th row of A is defined as the vector $(a_{i1}, a_{i2}, \ldots, a_{im})$ and the j-th column of A is defined as the vector $(a_{1j}, a_{2j}, \ldots, a_{nj})$. The row set and the column set of A are denoted by $\mathbf{R}(A)$ and $\mathbf{C}(A)$ respectively. For $I \subseteq \mathbf{R}(A)$ and $J \subseteq \mathbf{C}(A)$, we define $A_{I,J} = \big(a_{ij} \mid i \in I,\ j \in J\big)$, i.e. it is the submatrix (or minor) of A with the row set I and the column set J. The *rank* of a matrix is the cardinality of the maximum sized collection of columns which are linearly independent. Equivalently, the rank of a matrix is the cardinality of the maximum sized collection of rows which are linearly independent. It is denoted by $\mathsf{rank}(A)$.

For a graph G, we use $V(G)$ and $E(G)$ to denote the vertex set and the edge set of G. For a set of vertices V and a set of edges E on V, we use (V, E) to denote the graph formed by them. Let the vertex set be ordered as $V(G) = \{v_1, v_2, \ldots, v_n\}$. We only consider simple graphs in this paper, i.e. every edge is distinct and the two endpoints of an edge are also distinct. The complement of a graph G is defined as the graph $\overline{G} = (V(G), \overline{E(G)})$ where $\overline{E(G)} = \{(u,v) | (u,v) \in (V(G) \times V(G)) \setminus E(G), u \neq v\}$. The *adjacency matrix* of G, denoted by A_G, is defined as the $n \times n$ matrix whose rows and columns are indexed by $V(G)$, such that $A_G(i,j) = 1$ if and only if $(v_i, v_j) \in E(G)$. For a vertex $v \in V(G)$, we use $R(v)$ to denote the row of A_G corresponding to v. Similarly we define $C(v)$ to denote the column of A_G corresponding to v. For a set of vertices $S \subseteq V(G)$, we use $R(X)$ and $C(Y)$ to denote the set of rows and columns corresponding to the vertices in S. Similarly for $X, Y \subseteq V(G)$, we use $A_{X,Y}$ to denote the submatrix of A_G corresponding to rows in $R(X)$ and columns in $C(Y)$. For a vertex $v \in G$, $N(v) = \{u \in G | (u,v) \in E(G)\}$ denotes the neighbourhood of v, and $N[v] = N(v) \cup \{v\}$ denotes the *closed neighbourhood* of v. An *independent set* in a graph G is a set of vertices X such that for every pair of vertices $u, v \in X$, $(u,v) \notin E$. A *clique* on n vertices, denoted by K_n, is a graph on n vertices such that every pair of vertices has an edge between them. A *bi-clique* is a graph G, where $V(G) = V_1 \uplus V_2$ and for every pair of vertices $v_1 \in V_1$ and $v_2 \in V_2$, there is an edge $(v_1, v_2) \in E(G)$. When $|V_1| = |V_2| = n$ we denote the biclique by $K_{n,n}$. We can similarly define complete multipartite graphs. For a set of vertices U and a graph G, $G[U]$ denotes the induced subgraph of G on $U \cap V(G)$, and $G \setminus U$ denotes the graph $G[V(G) \setminus U]$.

Given a graph G, we define a relation $\sim$ on $V(G)$. Two vertices u and v are $u \sim v$ if and only if $N(u) = N(v)$. This definition gives us the following lemma.

Lemma 1. *(i) $u \sim v$, if and only if $R(u) = R(v)$ and $C(u) = C(v)$.*
(ii) $\sim$ partitions V as $V_1, V_2, \ldots, V_l$ where each V_i is an independent set in G.

(iii) For each pair of V_i and V_j, either there are no edges between V_i and V_j, or $G[V_i \uplus V_j]$ is a bi-clique.

The subsets $V_1, \dots, V_l$ are called *independent modules* of the graph G. We also refer to V_i as an equivalence class in G. The reduced graph of G is the graph $G^{\sim}$ as follows. The vertex set is $V(G^{\sim}) = \{u_1, \dots, u_l\}$ where l is the number of independent modules of G. The edge set is $E(G^{\sim}) = \{(u_i, u_j) | G[V_i \uplus V_j]$ is a bi-clique $\}$.

We denote the adjacency matrix of $G^{\sim}$ by $A_G^{\sim}$. Observe that every vertex of $G^{\sim}$ has a distinct neighbourhood, therefore all the columns of $A_G^{\sim}$ are distinct. We now have the following lemma.

Lemma 2 (Proposition 1. [1]). $\mathsf{rank}(A_G) = \mathsf{rank}(A_G^{\sim})$.

Following result by Lovašz [12] gives us a bound on the number of vertices in $G^{\sim}$, when $\mathsf{rank}(A_G) = r$.

Theorem 1. *If the rank of the adjacency matrix of a graph G is r then the number of vertices in $G^{\sim}$ is at most $c \cdot 2^{\frac{r}{2}}$ for an absolute constant c.*

The following corollary of the above theorem is used extensively in our algorithms.

Theorem 2 ($\star$[1]). *For every fixed r, the number of distinct reduced graphs $G^{\sim}$ such that $\mathsf{rank}(G^{\sim}) = r$, is upper bounded by $c \cdot 2^{2^r}$, for some absolute constant c.*

We also require the following lemma from [1](Proposition 7).

Lemma 3. *The only reduced graph of rank r with no isolated vertices and K_r as a subgraph is K_r itself.*

We also require the following results in the subsequent proofs.

Lemma 4 ($\star$). *$G^{\sim}$ is isomorphic to an induced subgraph of G.*

Observation 3 ($\star$) *If G' is an induced subgraph of G, then $\mathsf{rank}(A_{G'}) \leq \mathsf{rank}(A_G)$.*

3 Reducing Rank by Deleting Vertices

In this section, we consider r-Rank Vertex Deletion.

The set of vertices whose deletion reduces the rank to r is called solution set and k the solution size. We call r the *target rank*. We begin with a structural observation on the solution set of a given instance of r-Rank Vertex Deletion.

[1] Due to space constraints, proofs of the results marked $\star$ have been omitted. These will appear in the full version of the paper.

Lemma 5. *Let (G, k) be an instance to r-*Rank Vertex Deletion *and $\sim$ be the equivalence relation on $V(G)$. If $S \subseteq V(G)$ is a minimal solution for the instance (G, k) then it either contains all the vertices of an equivalence class defined by $\sim$ on $V(G)$ or none of it.*

Proof. Let $S \subseteq V(G)$ be a minimal solution, such that S contains at least one vertex of an equivalence class V_1 but it does not contain all of the vertices in V_1. Observe that all the vertices of $V_1 \setminus S$ have the same neighbourhood in $G \setminus S$, and so they go to the same equivalence class in $G \setminus S$. Let $S' = S \setminus V_1$. We claim that S' is a valid solution.

Let $V_1', \ldots, V_l'$ be a partition of $V(G \setminus S)$ into independent modules such that $(V_1 \setminus S) \subseteq V_1'$. Now, consider the graph $G \setminus S'$ and the partition of $V(G \setminus S')$ as $V_1'', V_2'', \ldots, V_l''$, where $V_1'' = V_1' \cup V_1$ and $V_i'' = V_i'$ $\forall i \geq 2$. Now observe that for any $u, v \in V_i''$, $N(u) = N(v)$. And for any u and v from different classes $N(u) \neq N(v)$, because $G \setminus S$ is an induced subgraph of $G \setminus S'$, and $(N(u) \cap V(G \setminus S)) \neq (N(v) \cap V(G \setminus S))$. Thus $V_1'', \ldots, V_l''$ is a partition of $V(G \setminus S')$ into independent modules. Further observe that $(V_i'' \cup V_j'')$ induces a complete bi-clique in $G \setminus S'$ if and only if $(V_i' \cup V_j')$ induces a complete bi-clique in $G \setminus S'$. Therefore $G \setminus S$ and $G \setminus S'$ have the same reduced graph. So by Lemma 2, $\mathsf{rank}(A_{G \setminus S}) = \mathsf{rank}(A_{G \setminus S'})$.

So $S' \subsetneq S$ is also a solution. But this contradicts the minimality of S, which implies that no such S exists. □

We obtain the following useful corollary of the above lemma.

Corollary 1. *A minimal solution to an instance (G, k) of r-*Rank Vertex Deletion *is disjoint from any equivalence class of G of cardinality greater than k.*

We also have the following useful observation.

Observation 4 ($\star$) *Let S be a solution to an instance (G, k) of r-*Rank Vertex Deletion*. Let $U \subseteq V(G)$. Then $S \cap U$ is a solution to the instance $(G[U], k)$.*

3.1 NP Completeness

Recall that r-Rank Vertex Deletion may be defined as a node-deletion problem with respect to the graph class Π_r. Since Π_r is non-trivial and hereditary, therefore this problem is NP-Complete [13,18].

Theorem 5 ($\star$). *r-*Rank Vertex Deletion *is* NP-Complete.

The proof of the theorem above appears in the full version of the paper, it gives a reduction from the Vertex Cover problem.

3.2 A Parameterized Algorithm for r-Rank Vertex Deletion

Let (G, k) be an instance of r-Rank Vertex Deletion. Let $G^{\sim}$ be the reduced graph of G. We have the following corollary of Lemma 5.

Corollary 2 ($\star$). *Let G' be obtained from G by removing all but $k+1$ vertices from each equivalence class of vertices in G. Then the instances (G, k) and (G', k) are equivalent instances of r-*Rank Vertex Deletion.

The rank of a matrix can be defined alternatively in terms of determinant of its square submatrices as follows.

Definition 1. *A matrix A over real numbers has rank equal to r if all the $(r+1) \times (r+1)$ submatrices of A have determinant zero and there exists a submatrix of size $r \times r$ such that its determinant is non-zero.*

Let $\mathcal{H}$ be a collection of subsets of a set U. Then $S \subseteq U$ is called a *hitting set* of $\mathcal{H}$, if S intersects every set in the collection $\mathcal{H}$. We shall use the notion of hitting set to show that r-Rank Vertex Deletion admits a polynomial kernel. Let G be a graph with adjacency matrix $A_{n \times n}$. Let $\mathcal{H}(G) = \{X \cup Y : X, Y \subseteq V(G), |X| = |Y| = r+1, \mathsf{rank}(A_{X,Y}) = r+1\}$ be a family of sets over $V(G)$.

Lemma 6. *For any $S \subseteq V(G)$, the rank of the adjacency matrix of $G \setminus S$ is at most r if and only if S is a hitting set of $\mathcal{H}(G)$.*

Proof. Let A be the adjacency matrix of G and let $\mathcal{H} = \mathcal{H}(G)$. For $S \subseteq V(G)$, let $A \setminus S$ denote the adjacency matrix of the graph $G \setminus S$. Observe that $A \setminus S$ is obtained from A by deleting the rows and columns corresponding to the vertices in S.

Let $S \subseteq V(G)$ be such that $A \setminus S$ has rank at most r, but S is not a hitting set of $\mathcal{H}$. Then there is a set in $\mathcal{H}$ which is not hit by S and corresponds to a set of rows and columns whose submatrix has rank equal to $r+1$, which is present in $A \setminus S$. This is a contradiction.

Conversely, let S be any hitting set of $\mathcal{H}$, but $A \setminus S$ has rank greater than $r+1$. Then there is a submatrix of $A \setminus S$ of size $(r+1) \times (r+1)$ which has rank equal to $r+1$, which is also a submatrix of A. Then by the definition of $\mathcal{H}$, the set of vertices corresponding to this submatrix is present in $\mathcal{H}$. But then S hits this set, which contradicts the fact that the rows and columns corresponding to these vertices are present in $A \setminus S$. $\square$

As a corollary of the above lemma, we immediately obtain a FPT algorithm for r-Rank Vertex Deletion by branching on any set $X \cup Y$ in $\mathcal{H}(G)$.

Theorem 6 ($\star$). r-Rank Vertex Deletion *admits an FPT algorithm running in time $2^{\mathcal{O}(k \log r)} n^{O(1)}$.*

Now, we show that r-Rank Vertex Deletion admits a polynomial kernel, by an application of the well known Sunflower lemma. We begin with the definition of a sunflower. A sunflower with k petals and a core Y is a collection of sets $S_1, \ldots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i \setminus Y$ are petals, and we require that none of the sets S_i is empty. Note that a family of pairwise disjoint sets is also a sunflower.

Lemma 7 (Sunflower Lemma [11]). *Let $\mathcal{F}$ be a family of sets with each set having cardinality equal to s. If $|\mathcal{F}| > s!(k+1)^s$ then $\mathcal{F}$ contains a sunflower with $k+2$ petals.*

Theorem 7 ($\star$). *r-Rank Vertex Deletion admits a kernel having at most $(2(r+1)) \cdot (2(r+1))!(k+1)^{2r+2}$ vertices.*

4 Reducing Rank by Editing Edges

In this section, we consider the problem of reducing the rank of a given graph by editing it's edge set. As before, we parameterize the problem by the solution size k. For the ease of presentation, we define the following notation. Let G be a graph and F be a set of edits. Let $G' = G \triangle F$ and $H = G'^{\sim}$. Then each vertex h of H corresponds to an equivalence class V'_h of G'. Let ϕ be a map from $V(G)$ to $V(H)$, which maps a vertex $v \in V(G)$ to the vertex $h \in V(H)$ if $v \in V_h$. Observe that, for a given graph G, F uniquely determines (ϕ, H). And let (ϕ, H) be such that for every vertex $h \in V(H)$ there is a vertex in $v \in V(G)$ such that $\phi(v) = h$. Then we can find a set F of edits to $E(G)$ such that the reduced graph of H is the same as the reduced graph $G \triangle F$. For each pair of vertices u, v in G such that $\phi(u) \neq \phi(v)$, if $(u, v) \in E(G)$ and $(\phi(u), \phi(v)) \notin E(H)$, then we need to delete the edge (u, v). Therefore we add (u, v) to F. Similarly, if $(u, v) \notin E(G)$ and $(\phi(u), \phi(v)) \in E(H)$ then we need to add the edge (u, v). So we add (u, v) to F. This completes the description of F and observe that it is uniquely determined by ϕ and H.

We now have the following lemma about the structure of minimal solutions of r-Rank Editing.

Lemma 8. *Let F be a minimal solution to an instance (G, k) of r-*Rank Editing*. Then for any two independent equivalence classes V_i and V_j of G, either F contains all the edges $V_i \times V_j$ or it contains none of them.*

Proof. Let G' denote the graph obtained after performing the edits in F on G. Let $H = G'^{\sim}$ and ϕ be the map as defined above. Suppose there is an equivalence class V_1 in G whose vertices go into different equivalence classes of G'. Let u be any vertex in V_1 which received the minimum number of edits among all the vertices in V_1, and let $h_u = \phi(u)$. Pick any vertex $v \in V_1$ which goes into a different equivalence class than that of u in the edited graph G'. Define the map ψ from $V(G)$ to $V(H)$ as $\psi(v) = h_u$ and $\psi(w) = \phi(w)$ for all $w \in V \setminus \{v\}$. Let F'' be the set of edits corresponding to (ψ, H) and $G'' = G \triangle F''$. Let H' be the induced subgraph of H such that every vertex of H' is the image of some vertex of G under the map ϕ. If $V(H') = V(H)$ then $G''^{\sim}$ is the graph H. Otherwise, by Lemma 4, $H'^{\sim}$ is an induced subgraph of H'. Since H' is an induced subgraph of H, therefore $H'^{\sim}$ is isomorphic to an induced subgraph of H. Since $G''^{\sim} = H'^{\sim}$, therefore by Observation 3, $\mathsf{rank}(A_{G''}) \leq \mathsf{rank}(A_H) = \mathsf{rank}(A_{G'})$. Let F'_u be the set of edits in F' with u as one endpoint, and let $F''_v = \{(v, w) | (u, v) \in F'_u, w \neq v\}$. Observe that $F'' = (F' \cup F''_v) \setminus (F'_v \cup \{(u, v)\})$, where F'_v denotes the edits in F' with v as one end point. Then clearly $|F''| \leq |F'|$.

We iteratively perform the above operation for all those vertices in V_1 which are mapped to a different equivalence class in G'. Thus we can find a solution such that all the vertices of V_1 go to the same equivalence class after editing. Moreover, applying this operation ensures that all the vertices in an equivalence class in G receive the same set of edits.

To complete the proof, observe that the graph induced on two equivalence classes is a complete bipartite graph. After editing, suppose V_i and V_j are contained in W_i and W_j respectively, where W_i and W_j are the equivalence classes of the edited graph. If $W_i = W_j$ then all the vertices between V_i and V_j have been deleted. If $W_i \neq W_j$ then all the edges of $V_i \times V_j$ are present if and only if W_i and W_j have an edge. $\square$

We can show a similar lemma for r-Rank Edge Deletion.

4.1 NP Completeness

We give a reduction from the d-Clustering problem, which is defined as follows. Given a graph G and two positive integers d and k, find a set F of k edges such that the graph $(V, E \triangle F)$ is the partition of at most d disjoint cliques. This problem is known to be NP-Complete for any $d \geq 2$ [17]. However it is FPT when parameterized by k [7].

Theorem 8. *r-Rank Editing is NP-Complete for any $r \geq 3$*

Proof. It is clear that the problem is in NP, since we can verify any claimed solution in polynomial time.

To show that the problem is NP hard for a fixed $r \geq 3$, we give a reduction from the 2-Clustering problem. Given an instance (G, k); let $V = V(G)$, $E = E(G)$ and $n = |V|$. We define an instance of r-Rank Editing as follows. Let Z be the complete $(r-2)$-partite graph $K_{k+2,\ldots,k+2}$, where each partition has exactly $k+2$ vertices, and all the edges between any two partitions are present. Observe that it has rank $r-2$ and has K_{r-2} as it's reduced graph. We construct the graph H by taking the complement $\overline{G}$ of G, a copy of the graph Z and adding all edges between the vertices of Z and vertices in $\overline{G}$. Then the instance of r-Rank Editing is (H, k). Let $\overline{E} = E(\overline{G})$.

In the forward direction, suppose that the instance (G, k) has a minimal solution F of size at most k such that $G' = (V, E \triangle F)$ is a partition of two disjoint cliques X and Y. Then observe that $H' = (V \cup V(Z),$

$(\overline{E} \triangle F) \cup E(Z) \cup E(Z, V(G)))$ is a graph with K_r or K_{r-1} as it's reduced graph. Thus H' has rank r or $r-1$.

In the reverse direction, suppose F is a minimal solution of size k for (H, k). Let $H' = \big(V \cup V(Z), (\overline{E} \triangle F) \cup E(Z) \cup E(Z, V(G))\big)$. Observe that there are no isolated vertices in H', since every vertex has at least $k+1$ neighbours in H. Since each equivalence class of $H[Z]$ has $k+2$ vertices and $|F| \leq k$, therefore by Lemma 8, $F \cap \{(E(Z) \cup E(Z, V(G))\} = \phi$. Thus $F \subseteq V(G) \times V(G)$, and any independent equivalence class of H' is contained in either $V(G)$ or $V(Z)$. Now suppose that X and Y are two equivalence classes of H' which are contained in $V(G)$ such that $H'[X \cup Y]$ is a bi-clique. Then observe that the induced subgraph $H'[X \cup Y \cup V(Z)]$ has no isolated vertices and it has K_r as it's reduced subgraph. Therefore by Lemma 3, the reduced graph of H' is also K_r, which implies that X and Y form a partition of $V(G)$. And if we cannot find such an X and Y, then $H'[V(G)]$ contains no edges and so the reduced graph of H' is K_{r-1}. Therefore $\overline{H'[V(G)]}$ is either an independent set or a bi-clique, and so $\overline{H'[V(G)]}$ can be partitioned into at most 2 cliques. Since $(V, E \triangle F) = \overline{H'[V(G)]}$, we have that F is a solution to the instance (G, k). □

For r-Rank Edge Deletion, we can show the following theorem, by reducing from the Maximum Edge Bi-Clique Problem [14].

Theorem 9 ($\star$). *For any fixed $r \geq 2$, r-*Rank Edge Deletion *is* NP-Complete.

4.2 A Parameterized Algorithm for r-Rank Editing

We now show that r-Rank Editing is FPT parameterized by the solution size and admits a polynomial kernel. The results for r-Rank Edge Deletion follow in a similar manner. We call a vertex $v \in V$, an *affected vertex* with respect to the solution F, if there is some edge in F which is incident on v. Observe that by Lemma 8, if a vertex in an equivalence class is affected with respect to a minimal solution F, then every vertex in V_1 is also affected. In that case, we say that the equivalence class V_1 is affected by F.

Lemma 9 ($\star$). *For any instance (G, k) of the r-*Rank Editing *problem. If $G^{\sim}$ has more than $c \cdot 2^{r/2} + 2k$ vertices then the instance has no solution.*

As a corollary of the above result and Lemma 8, we have the following kernel for r-Rank Editing.

Theorem 10 ($\star$). r-Rank Editing *admits a kernel with $\mathcal{O}((2^{r/2}+2k) \cdot (k+1))$ vertices.*

Next, let A be the adjacency matrix of G, and let $A_{X,Y}$ be a submatrix of A having rank $r+1$ where X and Y correspond to a subset of rows and columns respectively. Then observe that any solution to the instance (G, k), must edit an edge contained in $A_{X,Y}$. This observation immediately gives us an FPT algorithm for r-Rank Editing, in a similar way to Theorem 6.

Theorem 11. r-Rank Editing *admits an FPT algorithm running in time* $2^{O(k \log r)} n^{O(1)}$.

However we can obtain a sub-exponential algorithm for r-Rank Editing, by using an algorithm of Damaschke et. al. [6]. We begin with the definition of the *closed neighbourhood relation* on G. We define a relation $\approx$ on vertices of G, where $u \approx v$ if and only if $N[u] = N[v]$ in G. Observe that $\approx$ is an equivalence relationship on vertices of G, where each equivalence class is a clique. Let $V_1, \ldots, V_l$ be a partition of vertices $V(G)$. We define the $\approx$-reduced graph $G^{\approx}$ as follows: the vertex set is $V(G^{\approx}) = \{u_1, \ldots, u_l\}$ and the edge set is $E(G^{\approx}) = \{(u_i, u_j)$ if and only if all the edges between V_i and V_j are present in $E(G)\}$. Note that $G^{\approx}$ may contain several isolated vertices.

The H-Bag Editing problem is defined as follows.

H-Bag Editing	*Parameter: k*

Input: Graphs G, H and an integer k.
Question: Can we find a set F of k edges such that $\approx$-reduced graph of $G' = (V(G), E(G) \triangle F)$ is an induced subgraph of H?

Theorem 12 ([6]). *Any bag modification problem with a fixed graph H can be solved in $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$ time*[2].

We show how to use the above algorithm to obtain a sub-exponential FPT algorithm for r-Rank Editing.

Lemma 10. *Given a graph G, let $G^{\sim}$ denote the reduced graph with respect to the excluded neighborhood relation $\sim$ and let $G^{\approx}$ denote the reduced graph with respect to the closed neighborhood relation $\approx$. Then the graph $(\overline{G})^{\approx}$ is the complement of $G^{\sim}$.*

Proof. Observe that it suffices to prove that the equivalence classes of $\approx$ in $\overline{G}$ are exactly the same as the equivalence classes of $\sim$ in G. Consider the forward direction. Suppose $u \sim v$ for $u, v \in V[G]$, we want to prove that $u \approx v$ in $\overline{G}$. As $u \sim v$, $N_G(u) = N_G(v)$ therefore $N_{\overline{G}}[u] = V[G] \backslash N_G(u) = V[G] \backslash N_G(v) = N_{\overline{G}}[v]$, moreover $(u, v) \in E[\overline{G}]$, which implies $u \approx v$ in $\overline{G}$. We can show the reverse direction in a similar way. Since the $\sim$-equivalence classes of G are same as the $\approx$-equivalence classes of $\overline{G}$, their corresponding reduced graphs are complements of each other. $\square$

Corollary 3. *If H is a $\sim$-reduced graph then its complement is a $\approx$-reduced graph.*

Theorem 13 ($\star$). *An instance (G, k)* r-Rank Editing *can be solved in $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$ time.*

[2] The $\mathcal{O}^*$ notation hides the terms depending on r which is assumed to be a constant, and polynomial multiplicative factors.

5 Conclusion

In this paper we studied the vertex deletion and edge edition problems of reducing the "rank of the graph". We saw that the problems are NP-Complete and they admit FPT algorithms and kernels.

We conclude with a few open problems. Is it possible to obtain improved kernels and algorithms for these problems? In particular, is it possible to improve the exponent of the subexponential algorithm for r-Rank Editing? Further, what is complexity of the problem of reducing the number of distinct eigenvalues of a graph by deleting a few vertices or editing a few edges?

References

1. Akbari, S., Cameron, P.J., Khosrovshahi, G.B.: Ranks and signatures of adjacency matrices (2004)
2. Burzyn, P., Bonomo, F., Durán, G.: Np-completeness results for edge modification problems. Discrete Applied Mathematics **154**(13), 1824–1844 (2006)
3. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processing Letters **58**(4), 171–176 (1996)
4. Cao, Y., Marx, D.: Chordal editing is fixed-parameter tractable. In: STACS, pp. 214–225 (2014)
5. Cao, Y., Marx, D.: Interval deletion is fixed-parameter tractable. ACM Transactions on Algorithms (TALG) **11**(3), 21 (2015)
6. Damaschke, P., Mogren, O.: Editing the simplest graphs. In: Pal, S.P., Sadakane, K. (eds.) WALCOM 2014. LNCS, vol. 8344, pp. 249–260. Springer, Heidelberg (2014)
7. Fomin, F.V., Kratsch, S., Pilipczuk, M., Pilipczuk, M., Villanger, Y.: Tight bounds for parameterized complexity of cluster editing with a small number of clusters. Journal of Computer and System Sciences **80**(7), 1430–1447 (2014)
8. Fomin, F.V., Villanger, Y.: Subexponential parameterized algorithm for minimum fill-in. SIAM Journal on Computing **42**(6), 2197–2216 (2013)
9. Guo, J.: A more effective linear kernelization for cluster editing. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 36–47. Springer, Heidelberg (2007)
10. Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-parameter algorithms for cluster vertex deletion. Theory of Computing Systems **47**(1) (2010)
11. Jukna, S.: Extremal combinatorics: with applications in computer science. Springer Science & Business Media (2011)
12. Kotlov, A., Lovász, L.: The rank and size of graphs. Journal of Graph Theory **23**(2), 185–189 (1996)
13. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is np-complete. Journal of Computer and System Sciences **20**(2), 219–230 (1980)
14. Peeters, R.: The maximum edge biclique problem is np-complete. Discrete Applied Mathematics **131**(3), 651–654 (2003)

15. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters **32**(4), 299–301 (2004)
16. Robertson, N., Seymour, P.D.: Graph minors. xiii. the disjoint paths problem. Journal of Combinatorial Theory, Series B **63**(1), 65–110 (1995)
17. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. **144**(1-2) (2004)
18. Yannakakis, M.: Node-and edge-deletion np-complete problems. In: STOC, pp. 253–264. ACM (1978)

Knapsack and Allocation

On the Number of Anchored Rectangle Packings for a Planar Point Set

Kevin Balas[1,2] and Csaba D. Tóth[1,3](✉)

[1] California State University Northridge, Los Angeles, CA, USA
balask@lamission.edu, cdtoth@acm.org
[2] Los Angeles Mission College, Sylmar, CA, USA
[3] Tufts University, Medford, MA, USA

Abstract. We consider packing axis-aligned rectangles $r_1, \ldots, r_n$ in the unit square $[0,1]^2$ such that a vertex of each rectangle r_i is a given point p_i (i.e., r_i is *anchored* at p_i); and explore the combinatorial structure of all locally maximal configurations. When the given points are lower-left corners of the rectangles, then the number of maximal packings is shown to be at most $2^n C_n$, where C_n is the nth Catalan number. The number of maximal packings remains exponential in n when the points may be arbitrary corners of the rectangles. Our upper bounds are complemented with exponential lower bounds.

1 Introduction

Let P be a finite set of points $p_1, \ldots, p_n$ in the unit square $[0,1]^2$. An *anchored rectangle packing* for P is a set of axis-aligned empty rectangles $r_1, \ldots, r_n$, that lie in $[0,1]^2$, are interior-disjoint, and point p_i is one of the four corners of r_i for $i = 1, \ldots, n$. We say that rectangle r_i is *anchored* at p_i. In a *lower-left anchored rectangle packing* (*L-anchored packing*, for short), p_i is the lower-left corner of r_i for all i.

Anchored rectangle packings have applications in map labeling in geographic information systems [15–17] and VLSI design [18]. A fundamental problem is to find the maximum total area $A(P)$ (resp., $A_L(P)$) of the rectangles in an anchored (resp., L-anchored) rectangle packing of P. Allen Freedman conjectured (c.f. [23,24]) that if $(0,0) \in P$, then P admits an L-anchored rectangle packing of area at least $1/2$, that is $A_L(P) \geq 1/2$. The currently known best lower bound in this case is $A_L(P) \geq 0.091$ due to Dumitrescu and Tóth [11].

A rectangle r_i with lower-left anchor $p_i = (a_i, b_i)$, can be parameterized by two variables x_i and y_i such that $r_i = [a_i, x_i] \times [b_i, y_i]$. Consequently, the area of an L-anchored rectangle packing is a continuous multivariable function in $2n$ variables $\sum_{i=1}^{n} \text{area}(r_i) = \sum_{i=1}^{n} (x_i - a_i)(y_i - b_i)$, over a domain determined by the geometric constraints of the packing. We call an L-anchored rectangle packing *maximum* (resp., *maximal*) if it attains the global (resp., a local) maximum of this function. We define *maximum* and *maximal* anchored rectangle packing analogously.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 377–389, 2015.
DOI: 10.1007/978-3-319-21398-9_30

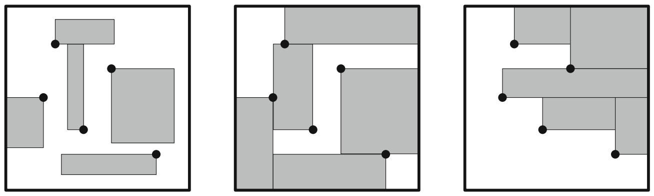


Fig. 1. Left: a set P of 5 points in the unit square $[0, 1]^2$ and an anchored rectangle packing for P. Middle: a maximal anchored rectangle packing for P. Right: A maximal L-anchored rectangle packing for P.

For computing the maximum area, $A_L(P)$ or $A(P)$, for a given point set P, it is instrumental to estimate the *number* of maximum packings. It is easily seen that the number of maximum packings is at least exponential in $n = |P|$ if, for example, P contains n points on a diagonal of $[0, 1]^2$. The enumeration of locally maximal configurations, which can be computed greedily, combined with reverse search [7] yields a simple strategy for finding the global maximum. In this paper, we control the number of (locally) maximal anchored and L-anchored rectangle packings. For an integer $n \in \mathbb{N}$, let $M(n)$ (resp., $M_L(n)$) denote the largest number of maximal rectangle packings over all sets $P \subset [0, 1]^2$ of n noncorectilinear points (two points are *corectilinear* if they have the same x- or y-coordinate).

Results. In this paper, we prove exponential upper and lower bounds for $M_L(n)$ and $M(n)$. Our upper bound for $M_L(n)$ is expressed in terms of the nth Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n} \sim 4^n/(n^{3/2}\sqrt{\pi})$.

Theorem 1. *We have* $\Omega(4^n/\sqrt{n}) \le M_L(n) \le C_n 2^n = \Theta(8^n/n^{3/2})$.

Note that both the lower and upper bounds are larger than C_n. The lower bound follows from an explicit construction. The upper bound is the combination of two tight upper bounds. Each L-anchored rectangle packing induces a subdivision of $[0, 1]^2$ into "staircases" (*L-subdivisions*, defined in Sec. 3). We show that the number of L-subdivisions for n points is at most C_n, and this bound is attained when the points form an antichain under the product order. We also show that each L-subdivision is induced by at most 2^{n-1} L-anchored rectangle packings, and this bound is attained when the points form a chain under the product order.

The machinery developed for the proof of Theorem 1 does not extend to general anchored rectangle packings. Nevertheless, we can prove that the number of maximal (any corner) anchored rectangle packings is exponential

Theorem 2. *There exist constants* $1 < c_1 < c_2$ *such that* $\Omega(c_1^n) \le M(n) \le O(c_2^n)$.

We derive an exponential upper bound using the contact graph of the rectangles in a packing. Specifically, we show that the contact graph can be represented by a planar embedding of the contact graph in which the vertices are points in the rectangles, and the edges are represented by polylines with at most one bend per edge. The number of graphs with such an embedding is known to be exponential [12]. We can encode all maximal anchored rectangle packings for P using one such graph and $O(n)$ bits of additional information. This leads to an exponential upper bound.

Remark. In a maximal anchored or L-anchored rectangle packing, we may assume that all vertices of all rectangles lie on one of the $(n+2)^2$ "grid points" induced by the vertical and horizontal lines passing through the n points in P and the corners of $[0,1]^2$ (cf. Sec. 2). This crucial property discretizes the problem, but is insufficient for establishing an exponential upper bound. By choosing the points (x_i, y_i) among the grid points, we obtain only a weak upper bound of $(n-1)^n$ (resp., $(n!)^2$ for L-anchored packings).

Related Work. Packing axis-aligned rectangles in a rectangular container, albeit without anchors, is the unifying theme of several classic optimization problems. The 2D knapsack problem, strip packing, and 2D bin packing involve arranging a set of given rectangles in the most economic fashion [3,8,14]. The maximum weight independent set for rectangles involves selecting a maximal area packing from a set of given rectangles [4]. These optimization problems are NP-hard, and there is a rich literature on the best approximation algorithms. Our problem setup is fundamentally different: the rectangles have variable sizes, but their location is constrained by the anchors. In this sense, it is reminiscent to classic Voronoi diagrams for n points in the plane. However, the Voronoi cells tile the space without gaps. Area maximization problems arise in the context of Voronoi games [5,9], where two players alternately choose points in a bounding box and wish to maximize the total area of the Voronoi cells of their points.

Combinatorial bounds for the number of some other geometric configurations on n points in the plane have been studied extensively. Determining the maximum number of (geometric) triangulations on n points in the plane captivated researchers for decades. The current best upper and lower bounds are $\Omega(8.65^n)$ and $O(30^n)$ [10,20]. Ackerman et. al. [1,2] established an upper bound of $O(18^n/n^4)$ for the number of *rectangulations* of n points in $[0,1]^2$, where a *rectangulation* is a subdivision of $[0,1]^2$ into $n+1$ rectangles by n axis-parallel segments, each containing a given point. This structure is reminiscent of L-subdivisions, defined in Sec. 3, for which we prove a tight upper bound of $C_n \leq O(4^n/n^{3/2})$. The number of anchored rectangle packings has not been studied before. It is not known if finding the maximum area of an anchored rectangle packing of n given points is NP-hard.

2 Discretization of Maximal Anchored Rectangle Packings

Let $P \subset [0,1]^2$ be a set of noncorectilinear points $p_1, \ldots, p_n$. The vertical and horizontal lines that pass through the points in P and the edges of the bounding box are called *grid lines*. The *grid points* are the intersections of the grid lines.

It is easy to see that all vertices of a maximal L-anchored rectangle packing must be grid points.

Proposition 1. *If an L-anchored rectangle packing for P has maximal area, then all corners of all rectangles are grid points.*

Proof. Consider an anchored rectangle packing of maximal area. The left and bottom edges are on grid lines. This implies that each rectangle may only expand up and to the right. Because the packing is of maximal area, no rectangle can expand (while other rectangles are fixed). The upper and right edges of each rectangle are necessarily in contact with the bottom and left edges of other rectangles or with the bounding box. This places the upper-right vertex at the intersection of two grid lines and thus on a grid point. We have shown that the lower-left and the upper-right corner of every rectangle is a grid point. From the definition of grid lines, this implies that all corners of all rectangles are grid points. □

The situation is more subtle when the rectangles can be anchored at arbitrary corners. Specifically, a local maximum may be attained at a "plateau" where the configuration can vary continuously while maintaining the same maximal area. A transformation that maintains the total area of the rectangles is called *equiareal*.

Proposition 2. *If an anchored rectangle packing for P has maximal area, then*

- *the local maximum is isolated, and all vertices of all rectangles are grid points, or*
- *there is an equiareal continuous deformation to an anchored rectangle packing in which all vertices of all rectangles are grid points; furthermore, the deformation either creates a contact between two previously disjoint rectangles, or decreases the area of some rectangle to 0.*

Proof. Consider a maximal anchored rectangle packing for P. Suppose that at least one rectangle has a vertical or horizontal edge not on a grid line. Assume first that a vertical edge of a rectangle is not on a grid line. Let ℓ be the vertical line through the leftmost such edge. Denote by L the set of rectangles whose right edges intersect ℓ, and R the set of rectangles whose left edges intersect ℓ. We can deform the rectangles in L and R simultaneously by translating ℓ. The sum of heights of rectangles in L equals the sum of heights of rectangles in R, otherwise translating ℓ in one of the two possible directions increases the total area before ℓ becomes a grid line. When ℓ shifts to the left, the rectangles in L shrink and may potentially reach 0 area; while the rectangles in R expand and may potentially reach another rectangle. However, because of the choice of ℓ, all edges of such a rectangle lie on grid lines. Translate ℓ until the area of a rectangle in L drops to 0, or the left edge of a rectangle in R reaches the boundary of a new rectangle or the bounding box. Repeat this operation for the next leftmost line ℓ until all vertical edges are on grid lines.

Note that horizontal edges were not affected by the above transformations. We can now deform the horizontal edges of the rectangles (independent of the vertical edges) by repeating the argument starting with the topmost horizontal line. Necessarily, all vertices of all rectangles become grid points. □

In the remainder of the paper, we consider maximal anchored rectangle packings in which the vertices of all rectangles are grid points.

3 Lower-Left Anchored Rectangle Packings

The key tool for the proof of Theorem 1 is a subdivision of the unit square $[0,1]^2$ into staircase polygons, defined below. Let $P = \{p_1, \ldots, p_n\}$ be a set of noncorectilinear points in $[0,1]^2$. We may assume that $(0,0) \notin P$ (by scaling P, if necessary, since maximality is an affine invariant). Let $q = (0,0)$ denote the lower-left corner of $[0,1]^2$.

An *L-shape* for a point p_i $(i = 1, \ldots, n)$ is the union of a horizontal and a vertical segment whose left and bottom endpoint, respectively, is p_i. Refer to Fig. 2(a). An *L-subdivision* for P is formed by n L-shapes for p_i $(i = 1, \ldots, n)$ such that the top and right endpoint of each L-shape lies in another L-shape or the boundary of $[0,1]^2$. The L-shapes subdivide $[0,1]^2$ into $n+1$ simple polygons, called *staircases*. By construction, the lower-left corner of each staircase is either $q = (0,0)$ or a point in P. The upper-right vertices of a staircase are called *steps* of the staircase.

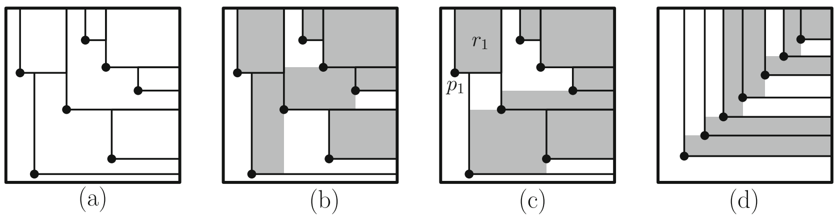


Fig. 2. (a) An L-subdivision for P. (b) An L-subdivion induced by a maximal L-anchored rectangle packing. (c) Maximal anchored rectangles in the staircases that do not form a maximal L-anchored rectangle packing: rectangle r_1 could expand. (d) For n points on the line $y = x$, $M_L(P) = 2^{n-1}$.

Proposition 3. *In every L-subdivision for P, the n staircases anchored at the points in P jointly have at most $2n-1$ steps.*

Proof. Each step of a staircase is either the upper-right corner of $[0,1]^2$, or a top or right endpoint of an L-shape. Every such point is the step of a unique staircase. The n L-shapes yield $2n$ steps, and the upper-right corner of $[0,1]^2$ yields one step. The staircase anchored at $q = (0,0)$ has at least two steps, hence the remaining staircases jointly have at most $2n + 1 - 2 = 2n - 1$ steps. □

Maximal L-anchored packings versus L-subdivisions

Proposition 4. *For every maximal L-anchored rectangle packing of P, there is an L-subdivision such that rectangle r_i lies in the staircase anchored at p_i for $i = 1, \dots, n$.*

Proof. Let $r_1, \dots, r_n$ be an L-anchored rectangle packing for $p_1, \dots, p_n \in [0,1]^2$. For each $i = 1, \dots, n-1$, successively draw an L-shape as follows (refer to Fig. 2(b)). First extend the bottom edge of r_i to the right until it hits the bounding box, the left edge of another rectangle, or a previously drawn L-shape. Similarly, extend the left edge of r_i up until it hits the bounding box, the bottom edge of another rectangle, or a previously drawn L-shape. The n L-shapes form an L-subdivision. By construction, the L-shapes are disjoint from the interior of the rectangles $r_1, \dots, r_n$, hence each rectangle lies in a staircase. Since the lower-left corner of each staircase is $q = (0,0)$ or a point in P, each staircase with lower-left corner $p_i \in P$ contains the rectangle anchored at p_i. □

In the L-subdivision described in Proposition 4, each rectangle r_i ($i = 1, \dots, n$) is a maximal rectangle within a staircase polygon. However, the converse is not necessarily true. Choose maximal rectangles, in all staircases, with lower-left corners in P. This need not produce a maximal L-anchored rectangle packing for P. See an example in Fig. 2(c). Nevertheless, we can derive an upper bound for $M_L(P)$.

Proposition 5. *In every L-subdivision for P, $|P| = n$, there are at most 2^{n-1} possible ways to choose a maximal rectangle in each staircase whose lower-left corner is in P. This bound is the best possible.*

Proof. If the staircases anchored at the n points in P have $t_1, \dots, t_n$ steps, then there are precisely $\prod_{i=1}^n t_i$ different ways to choose a maximal anchored rectangle in each. By Proposition 3 and the arithmetic-geometric mean inequality yields

$$\prod_{i=1}^n t_i \le \left(\frac{1}{n}\sum_{i=1}^n t_i\right)^n = \left(2 - \frac{1}{n}\right)^n < 2^n. \tag{1}$$

The maximum of $\prod_{i=1}^n t_i$ subject to $\sum_{i=1}^n t_i = 2n-1$ and $t_1, \dots, t_n \in \mathbb{N}$ is attained when the $t_i's$ are distributed as evenly as possible, say, $t_1 = \dots = t_{n-1} = 2$ and $t_n = 1$. Consequently, $\prod_{i=1}^n t_i \le 2^{n-1}$. This upper bound is attained when the points in P form a chain in the product order (e.g., points on the line $y = x$), then $n-1$ staircases have 2 steps, and the staircase incident to $(1,1)$ has only 1 step (Fig. 2(d)). □

Let $S(P)$ be the number of all L-subdivisions for a noncorectilinear point set P; and let $S(n) = \max_{|P|=n} S(P)$. By Proposition 5, we have $M_L(P) \le S(P)2^n$ and $M_L(n) \le S(n)2^n$.

The Number of L-subdivisions. We prove a tight upper bound for $S(n)$, the maximum number of L-subdivisions for a set of n points in the unit square. Our upper bound is expressed in terms of the nth Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n} \sim 4^n/\sqrt{\pi n^3}$.

Lemma 1. *For every $n \in \mathbb{N}$, we have $S(n) = C_n$.*

Proof. Lower bound. Let P be a set of n points that form an antichain under the product order (e.g., points on the line $y = 1 - x$). In this case, each staircase anchored at a point in P is a rectangle, and it is well known [21,22] that the number of rectangular subdivisions is the Catalan number C_n. Hence $S(P) = C_n$ in this case.

Upper Bound. Let P be an arbitrary noncorectilinear set of n points in $[0,1]^2$. We may assume that the points $p_1, \ldots p_n$ are sorted by their x-coordinates, that is, $x_1 < \ldots < x_n$. If the points form an antichain under the product order, then their y-coordinates are monotone decreasing, and $S(P) = C_n$. Otherwise, we incrementally modify the y-coordinates of the points to become monotone decreasing such that the number of L-subdivisions increases. In each incremental step, we modify the y-coordinate of one point.

Suppose that the points in P do not form an antichain under the product order; and i is the smallest index such that the points with larger indices, $\{p_j \in P : j > i\}$, form an antichain and are incomparable to all other points (refer to Fig. 3). Let Z_0 be the minimum axis-aligned rectangle incident to $(0,1)$ that contains the points $p_1, \ldots, p_{i-1}$; and let Z_1 be the minimum axis-aligned rectangle incident to $(1,0)$ that contains the points $p_{i+1}, \ldots, p_n$. By the choice of i, the boxes Z_0 and Z_1 are on opposite sides of the vertical line $x = x_i$, as well as a horizontal line below $y = \min_{1 \le k \le i} y_k$. Let p'_i be the intersection of these two lines.

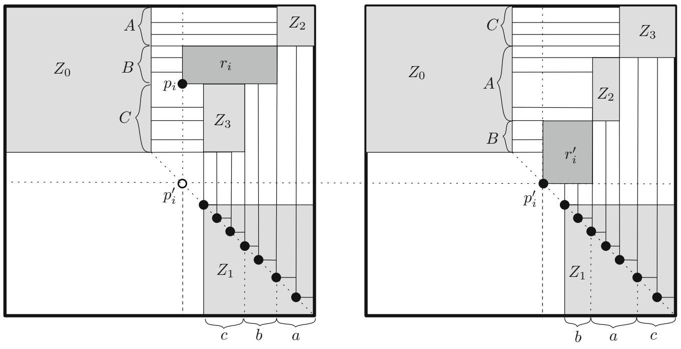


Fig. 3. Left: A schematic image of an L-subdivision D for P. Right: The corresponding L-subdivision D' for the modified point set P'.

We move point p_i to p'_i. Denote by P' the modified point set. In order to show $S(P) \le S(P')$, we construct an injective map $f : S(P) \to S(P')$. For every L-subdivision D of the point set P, we construct a unique L-subdivision $D' = f(D)$ of the modified point set P'. Let D be an L-subdivision of P (Fig. 3, left). Since no other point dominates p_i, the staircase anchored at p_i is a rectangle, that we denote by r_i. We introduce some notation. Some horizontal segments of L-shapes of points in Z_0 cross the right edge of Z_0: Let A, B, and C, respectively, denote the number of L-shapes whose horizontal segments pass above r_i, hit the left edge of r_i, and pass below r_i. Similarly, some vertical segments of L-shapes of points in Z_1 cross the top edge of Z_1: Let a, b, and c, respectively, denote the number of L-shapes whose vertical segments pass right of r_i, hit the bottom edge of r_i, and end strictly below the bottom edge of r_i. Let Z_2 be the axis-aligned rectangle that contains all intersections between the A horizontal segments passing above r_i and the a vertical segments right of r_i. Similarly, let Z_3 contain the intersections between the C horizontal segments passing below r_i and the c vertical segments that end strictly below r_i.

We can now define the L-subdivision D' for the modified point set P'. The arrangement of L-shapes restricted to the boxes Z_0 and Z_1 remains the same. Consequently, $A + B + C$ horizontal segments exit the right edge of Z_0, and $a + b + c$ vertical segments exit the top edge of Z_1. Draw an L-shape for the point p'_i such that it blocks the B lowest horizontal segments that exit Z_0 and the b leftmost vertical segments that exit Z_1. Group the remaining horizontal (resp., vertical) segments into bundles of size A and C (resp., a and c). Let the groups of size A and a intersect in the same pattern as in Z_2, and the groups of size C and c as in Z_3. This completes the description of the L-subdivision D'. By construction $D' = f(D)$ is a unique L-subdivision, and the function f is injective. □

We are now ready to prove Theorem 1.

Theorem 1. *We have* $\Omega(4^n/\sqrt{n}) \leq M_L(n) \leq C_n 2^n = \Theta(8^n/n^{3/2})$.

Proof. Let P be a set of n noncorectilinear points in the unit square. By Proposition 4, every maximal L-anchored rectangle packing for P can be constructed by considering an L-subdivision for P, and then choosing a maximal rectangle from each staircase anchored at a point in P. By Lemma 1, we have $S(P) \leq S(n) = C_n$ L-subdivisions for P. By Proposition 5, there are at most 2^{n-1} ways to choose maximal rectangles in the staircases. Consequently, we have $M_L(P) \leq S(P)2^{n-1} \leq C_n 2^{n-1}$.

Even though both Proposition 5 and Lemma 1 are tight, their combination is not tight, since they are attained on different point configurations: n points that form a chain or an antichain under the product order. Our lower bound

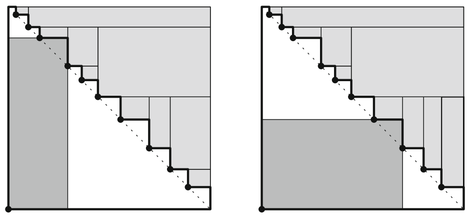


Fig. 4. One point at the origin and $n - 1$ points on the line $y = 1 - x$

is based on the following construction (refer to Fig. 4). Place one point at the origin and $n-1$ points on the line $y = 1 - x$. The L-shape of the first point is contained in the boundary of $[0,1]^2$, and the last $n-1$ points admit $C_{n-1} = \frac{1}{n}\binom{2n-2}{n-1} \sim 4^{n-1}/\sqrt{(n-1)^3\pi}$ L-subdivisions. The first point has a staircase with n steps ($t_1 = n$), all other staircases are rectangles ($t_i = 1$, for $i = 2, \ldots, n$). Consequently, $M_L(P) = S(P)\prod_{i=1}^{n} t_i = C_{n-1} \cdot n = \Theta(4^n/\sqrt{n})$, as required. $\square$

4 General Anchored Rectangle Packings

In this section, we prove Theorem 2. We show that a maximal anchored rectangle packing for a point set P can be reconstructed from the contact graph of the rectangles, and from $O(n)$ bits of additional information. Since a maximal rectangle packing may contain rectangles of 0 area (cf. Proposition 2), we need to be careful defining contact graphs.

The *contact graph* of a rectangle packing is a graph $G = (V, E)$, where V corresponds to the set of vertices, E to the set of edges, and two vertices are connected by an edge iff the corresponding rectangles have positive area and intersect in a nontrivial line segment; or one rectangle has 0 area and lies on the boundary of the other rectangle. It is easy to see that the contact graph of a rectangle packing is planar. However, the number of n-vertex planar graphs is super-exponential [13]. The number of graphs reduces to exponential with suitable geometric conditions. For a set P of n points in the plane, for example, the number of straight-line graphs with vertex set P is only exponential. An $\exp(O(n))$ bound was first shown by Ajtai et al. [6] using the crossing number method. The current best upper bound is $O(187.53^n)$, due to Sharir and Sheffer [20]. The contact graphs of any anchored rectangle packings for P can be embedded in the plane such that the vertex set is P, but these graphs cannot always be realized by straight-line edges. It turns out that a weaker condition will suffice: a *1-bend* embedding of a planar graph $G = (V, E)$ is an embedding in which the vertices are distinct points in the plane, and the edges are polylines with one bend per edge (that is, each edge is the union of two incident line segments). Frankeke and Tóth [12] proved recently that for every n-element point set, the number of such graphs is at most $\exp(O(n))$.

Lemma 2. *Let $P = \{p_1, \ldots, p_n\}$ be a noncorectilinar set in $[0,1]^2$. The contact graph of every maximal anchored rectangle packing for P has a 1-bend embedding in which the vertex representing rectangle r_i is point p_i for $i = 1, \ldots, n$.*

The proof would be straightforward if the anchors were in the interior of the rectangles. In that case, we could simply choose a *bend point* on the common boundary between two rectangles in contact, and then draw a 1-bend edge between their anchors via the bend point. When the anchors are at corners of the rectangles, we need to be more careful to prevent any overlap between adjacent edges.

Proof. Let $r_1, \ldots, r_n$ be a maximal anchored rectangle packing for P. For every $i = 1, \ldots, n$, point p_i is a corner of the rectangle r_i. For every two rectangles

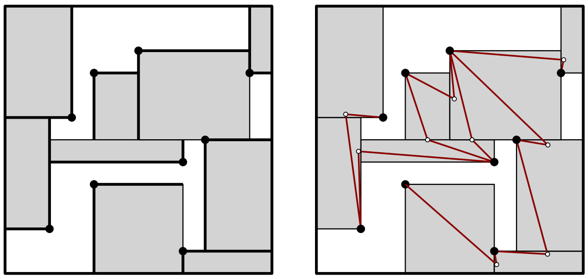


Fig. 5. Left: A maximal anchored rectangle packing for P. Thick lines indicate the L-shapes incident to the points in P. Right: A 1-bend embedding of the contact graph of the rectangles.

in contact, r_i and r_j, choose an arbitrary *preliminary* bend point $q_0(i,j)$ on the common boundary of r_i and r_j.

Define the *L-shape* anchored at p_i as the union of the two edges of r_i incident to p_i. Note that the relative interiors of the n L-shapes are pairwise disjoint, since the points $p_1, \ldots, p_n$ are noncorectilinear. Let the bend point $q(i,j) = q_0(i,j)$ if the preliminary bend point is not on an L-shape or if one of the rectangles has 0 area. Otherwise, assume $q_0(i,j)$ is on the L-shape of p_i. Then choose a bend point $q(i,j)$ in the interior of r_i in a sufficiently small neighborhood of the preliminary point $q_0(i,j)$. Now, for any two rectangles in contact, r_i and r_j, draw a 1-bend edge between p_i and p_j via $q(i,j)$. No two edges cross or overlap, and hence we obtain a 1-bend embedding of the contact graph of the rectangles. $\square$

For a fixed point set P, by Lemma 2, the contact graph of every maximal anchored rectangle packing admits a 1-bend embedding on the vertex set P. However, several maximal anchored rectangle packings may yield the same contact graph (as an abstract graph). We show that all maximal anchored rectangle packings for P can be encoded by their contact graphs and $O(n)$ bits of additional information. By Proposition 2, we may assume that all vertices of a maximal rectangle packing are grid points. Furthermore, we may also assume that there is no equiareal continuous deformation that creates a new contact or reduces the area of a rectangle to 0.

Fix a noncorectilinear point set $P = \{p_1, \ldots, p_n\}$. Every maximal anchored rectangle packing $r_1, \ldots, r_n$ is encoded by the following information:

(1) The contact graph G of the rectangles $r_1, \ldots, r_n$;
(2) for $i = 1, \ldots n$, an indicator variable σ_i such that $\sigma_i = 0$ iff area$(r_i) = 0$;
(3) for $i = 1, \ldots n$, the position of the anchor p_i in r_i (lower-left, lower-right, etc.);
(4) for each edge (i,j) of G, the orientation of the line segment $r_i \cap r_j$.

We now show that we can uniquely reconstruct a maximal anchored rectangle packing from this information.

Lemma 3. *For every noncorectilinear point set P, every code described above determines at most one maximal anchored rectangle packing for P, which can be (re)constructed in polynomial time.*

Proof. We are given the points $p_1, \dots, p_n$, and for every $i = 1, \dots, n$, we know which corner of the rectangle r_i is p_i. To reconstruct the rectangles r_i ($i = 1, \dots, n$), it is enough to find the corner of r_i opposite to p_i, which we denote by (x_i, y_i). That is $r_i = [\min(a_i, x_i), \max(a_i, x_i)] \times [\min(b_i, y_i), \max(b_i, y_i)]$. We determine the parameters x_i (resp., y_i) with the following strategy.

> Consider a rectangle r_i, and assume without loss of generality that p_i is the lower-left corner of r_i. If r_i is not in contact with any rectangle r_j such that $r_i \cap r_j$ is vertical and $a_i < a_j$, then $x_i = 1$ (that is, r_i extends to the right edge of the bounding box $[0, 1]^2$). If r_i is in contact with a rectangle r_j such that the segment $r_i \cap r_j$ is vertical, $a_i < a_j$, and the anchor p_j is the lower-left or upper-left corner of r_j, then we have $x_i = a_j$. Analogous conditions determine y_i in some cases.

We now show that our assumptions from Proposition 2 ensure that the above strategy determines x_i and y_i for all $i = 1, \dots, n$. If the above strategy fails to find x_i, then r_i is in contact with a rectangle r_j such that the segment $r_i \cap r_j$ is vertical, $a_i < a_j$, but p_j is the lower-right or upper-right corner of r_j. In this case, we call (r_i, r_j) a *horizontal pair*. Analogously, if the strategy does not find y_i, then r_i is part of some *vertical pair* (r_i, r_j). The horizontal (resp., vertical) pairs define a subgraph of the contact graph, that we denote by G_H (resp., G_V). Each connected component C of the graph G_H (resp., G_V) corresponds to rectangles whose left or right edge lies on some common vertical (resp., horizontal) line ℓ.

Consider a component C of G_H (the argument is analogous for G_V). The line ℓ must be right of all lower-left and upper-left anchors of rectangles in C, and left of all lower-right and upper-right anchors. Suppose that there exists a maximal anchored rectangle packing that satisfies these constraints. Denote by $L \subset C$ (resp., $R \subset C$) the set of rectangles whose right (resp., left) edges lie on ℓ. Similarly to the proof of Proposition 2, we deform the rectangles in L and R simultaneously by translating ℓ. If the sum of heights of rectangles in L and R differ, then translating ℓ in one of the two possible directions increases the total area, contradicting maximality. If the sum of heights of rectangles in L and R are equal, then translating ℓ in any direction is an equiareal deformation. We can now translate ℓ left until the area of a rectangle in L drops to 0 or a rectangle in R is in contact with a new rectangle on the left of ℓ. This contradicts our assumption that equiareal deformations create neither new contacts nor new rectangles of 0 area. Consequently, G_H (resp., G_V) is the empty graph, there are neither horizontal nor vertical pairs, and the above strategy uniquely determines x_i and y_i for all $i = 1, \dots, n$. $\square$

Theorem 2. *There exist constants* $1 < c_1 < c_2$ *such that* $\Omega(c_1^n) \leq M(n) \leq O(c_2^n)$.

Proof. The combination of Lemmas 2 and 3 yields the upper bound. Theorem 1 gives the lower bound.

5 Conclusions

We have considered two variants of anchored rectangle packings: the anchors p_i were required to be either the lower-left or arbitrary corners of the rectangles r_i. We could consider a variant that we call *relaxed anchored rectangle packing*, where the anchors p_i are contained in the rectangles r_i. In this case, the maximum area of a rectangle packing is always 1, since the bounding box can be subdivided into n parallel strips, each containing a point in P. Note that a rectangle $r_i = [x_i, x_i'] \times [y_i, y_i']$ is now described by 4 variables. In a relaxed anchored rectangle packing, however, a local maximum need not attain the global maximum. Nevertheless, the technique of Section 4 extends to this variant: each maximal rectangle packing can be reconstructed from the contact graphs of the rectangles (which has an embedding using polylines with at most one bend per edge), and $O(1)$ bits of additional information per rectangle. Consequently, the number of locally maximal packings for an n-element point set is bounded by $\exp(O(n))$.

Analogous problems arise for anchored packings with other simple geometric shapes, such as circular disks or positive homothets of some convex body. For packings with object of bounded description complexity, the configuration space can be parameterized with $O(n)$ variables, and some of the techniques developed here do generalize. However, several crucial steps in our work have relied on properties of axis-aligned rectangles. Determining the maximum area covered by a packing remains open for both anchored and L-anchored rectangle packings. For other geometric shapes (e.g., circular disks), finding the maximum area covered by relaxed anchored variants is already a challenging problem.

References

1. Ackerman, E.: Counting problems for geometric structures: rectangulations, floorplans, and quasi-planar graphs, PhD thesis, Technion (2016)
2. Ackerman, E., Barequet, G., Pinter, R.: On the number of rectangulations of a planar point set. J. Combin. Theory, Ser. A **113**(6), 1072–1091 (2006)
3. Adamaszek, A., Wiese, A.: Approximation schemes for maximum weight independent set of rectangles. In: Proc. 54th FOCS. IEEE (2013)
4. Adamaszek, A., Wiese, A.: A quasi-PTAS for the two-dimensional geometric knapsack problem. In: Proc. 26th SODA. SIAM (2015)
5. Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M., van Oostrum, R.: Competitive facility location: the Voronoi game. Theoret. Comput. Sci. **310**, 457–467 (2004)
6. Ajtai, M., Chvátal, V., Newborn, M., Szemerédi, E.: Crossing-free subgraphs. Annals Discrete Math. **12**, 9–12 (1982)
7. Avis, D., Fukuda, K.: Reverse search for enumeration Discrete Appl. Math. **65**, 21–46 (1996)
8. Bansal, N., Khan, A.: Improved approximation algorithm for two-dimensional bin packing. In: Proc. 25th SODA, pp. 13–25. SIAM (2014)
9. Cheong, O., Har-Peled, S., Linial, N., Matoušek, J.: The one-round Voronoi game. Discrete Comput. Geom. **31**, 125–138 (2004)

10. Dumitrescu, A., Schulz, A., Sheffer, A., Tóth, C.D.: Bounds on the maximum multiplicity of some common geometric graphs. SIAM J. Discrete Math. **27**(2), 802–826 (2013)
11. Dumitrescu, A., Tóth, C.D.: Packing anchored rectangles. In: Proc. 23rd SODA, pp. 294–305. SIAM (2012); and Combinatorica **35**(1), 39–61 (2015)
12. Francke, A., Tóth, C.D.: A census of plane graphs with polyline edges. In: Proc. 30th SoCG, pp. 242–250. ACM Press (2014)
13. Giménez, O., Noy, M.: Asymptotic enumeration and limit laws of planar graphs. J. AMS **22**, 309–329 (2009)
14. Harren, R., Jansen, K., Prädel, L., van Stee, R.: A $(5/3 + \varepsilon)$-approximation for strip packing. Comput. Geom. **47**(2), 248–267 (2014)
15. Kakoulis, K.G., Tollis, I.G.: Labeling algorithms, chap. 28. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization. CRC Press (2013)
16. Knuth, D., Raghunathan, A.: The problem of compatible representatives. SIAM J. Discete Math. **5**, 36–47 (1992)
17. van Kreveld, M., Strijk, T., Wolff, A.: Point labeling with sliding labels. Comput. Geom. **13**, 21–47 (1999)
18. Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: VLSI module placement based on rectangle-packing by the sequence-pair. IEEE Trans. CAD Integrated Circuits and Systems **15**(12) (1996)
19. Santos, F., Seidel, R.: A better upper bound on the number of triangulations of a planar point set. J. Combin. Theory, Ser. A **102**, 186–193 (2003)
20. Sharir, M., Sheffer, A.: Counting plane graphs: cross-graph charging schemes. Combinat. Probab. Comput. **22**, 935–954 (2013)
21. Stanley, R.: Problem k^8 in Catalan addendum to Enumerative Combinatorics, vol. 2, May 25, 2013. http://www-math.mit.edu/~rstan/ec/catadd.pdf
22. Thomas, H.: New combinatorial descriptions of the triangulations of cyclic polytopes and the second higher Stasheff-Tamari posets. Order **19**(4), 327–342 (2002)
23. Tutte, W.: Recent Progress in Combinatorics: Proceedings of the 3rd Waterloo Conference on Combinatorics, May 1968. Academic Press, New York (1969)
24. Winkler, P.: Packing rectangles. In: Mathematical Mind-Benders, pp. 133–134, A.K. Peters Ltd., Wellesley (2007)

Approximate Truthful Mechanism Design for Two-Dimensional Orthogonal Knapsack Problem

Deshi Ye(✉) and Guochuan Zhang

College of Computer Science, Zhejiang University, Hangzhou 310027, China
{yedeshi,zgc}@zju.edu.cn

Abstract. This paper provides a technique for designing truthful mechanisms for a combinatorial optimization problem, which requires composition algorithms. We show that the composition algorithm $A \circ B$ is monotone if the algorithm A and the algorithm B are both monotone. Then, we apply this technique to the two-dimensional orthogonal knapsack problem with provable approximation bounds, improving the previous results in [5].

Keywords: Mechanism design · Knapsack auction · Approximation algorithms

1 Introduction

Traditional optimization problems assume that the input data are available to the algorithm designer. However, in many Internet applications, such as combinatorial auction, the algorithms whose inputs are provided by selfish agents prefer to lie if there are benefits for themselves. Mechanism design is to deal with such selfish settings. The mechanism designer proposes allocation and payment algorithms for all agents beforehand, the agents then decide to report their own input data. Without loss of generality, assume that agents are rational and attempt to maximize their own utilities. Most of previous research work concerns on incentive compatible or truthful mechanisms, in which a dominate strategy for an agent is to report the true input data. Designing efficient truthful mechanisms that approximate the optimal social welfare was first considered by Nisan and Ronen [27]. Two different social objective functions of mechanisms were studied in this approach. One is about the utilitarian optimization problems, such as combinatorial auction and the knapsack problem. The other is to minimize the makespan of parallel machines with private speeds.

Technique designing for combinatorial auctions was well studied. Mu'alem and Nisan [26] provided several ways to combine two allocations algorithms, such as the MAX operator and the If-Then-Else operator. They studied the

D. Ye—Research was supported in part by China Scholarship Council.

G. Zhang—Research was supported by NSFC(11271325).

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 390–401, 2015.
DOI: 10.1007/978-3-319-21398-9_31

mechanisms for restricted combinatorial auctions where the subset of items of each bidder is known and only the valuation of these items is unknown by the mechanism (the single parameter case), a 2-approximation mechanism based on the greedy method for the knapsack problem was provided. Briest et al. [10] designed a new approach in rounding scheme that leads to monotone FPTASs, and therefore a monotone FPTAS for knapsack problem. Moreover, the problems considered in [10] are of multi-parameter. Chekuri and Gamzu [12] studied the greedy iterative packing truthful mechanism for the multiple knapsack problem. By presenting a property of loser-independent, they gave a truthful $(2+\varepsilon)$-approximate mechanism among single-minded agents, and $(e/(e-1)+\varepsilon)$-approximate mechanism for knapsacks with identical capacity. When the number of knapsacks is a fixed constant, Briest et al. [10] presented a monotone PTAS for the multiple knapsack problem. Grandoni et al. [19] designed monotone truthful multi-criteria FPTASs for multi-objective problems, which implies a monotone truthful FPTAS for the multi-dimensional knapsack problem. However, their FPTASs may violate each budget constraint by a factor $(1+\varepsilon)$. In general, various techniques appeared for the multi-dimensional packing problems, such as the convex-decomposition technique [24] as well as maximal-in-range [17].

In this work we aim at a technique for designing truthful mechanisms for the *two-dimensional orthogonal knapsack* with composition algorithms. In two-dimensional orthogonal knapsack(2DOK) (or Rectangle packing (RP)), it is asked to pack a set of rectangles into a bin (or a larger rectangle), where each rectangle is associated with a value. The goal is to maximize the total value of the selected rectangles that can be packed into the bin. This problem is motivated by the scenario in the advertising auction. In web applications there is a rectangle space available for advertisements. A set of advertisers would like to bid a room for displaying their graphical advertisements. An advertisement is usually a rectangle. The auctioneer (the owner of the space) has to choose a set of rectangles that can be packed into the space while the social welfare is maximized. If all the advertisements have the same width, the problem is reduced to the classical knapsack auction. Note that our problem differs from the multi-dimensional knapsack auction [19] in the fact our problem involves geometric constraints. Babaioff and Blumrosen [5] dealt with selling advertisement space on a newspaper page that can be modelled by packing convex figures in a plane, in which they show an $O(R)$-approximation truthful mechanism if convex figures are rectangles, where R is the ratio of the maximum diameter of a rectangle and the minimum width of a rectangle.

1.1 Related Work

The maximization problem of rectangle packing was considered in the literature. Jansen and Zhang [21] designed a $(2+\varepsilon)$-approximation algorithm. For the special case of maximizing the number of packed rectangles, Jansen and Zhang provided an asymptotic FPTAS (AFPTAS) as well as a PTAS. Resource augmentation was studied by Fishkin et al. [18]. For the multiple knapsack problem, FPTAS was ruled out even for two knapsacks unless P=NP [11,13]. Kellerer [22] devised a PTAS if the knapsacks are identical. Chekuri and Khanna [13] provided a PTAS for the general multiple knapsack problem. Jansen [20] showed that there exists an EPTAS. However, there is no EPTAS for the two-dimensional knapsack problem [23].

1.2 Our Contribution

Our main result is to design truthful mechanisms for the two-dimensional orthogonal knapsack problem via a composable algorithm. We show that the composition algorithm $A \circ B$ is monotone if the algorithm A (selection part) is monotone and the algorithm B (allocation part) is monotone. Suppose that $\varepsilon > 0$ is a given number. We first provide a monotone truthful and deterministic algorithm $A \circ B$ and show that the approximation ratio is at most $7 + \varepsilon$. If rotation of 90 degrees is allowed, we obtain an approximation ratio of at most $3 + \varepsilon$. Moreover, for multiple knapsacks, we derive a $(9 + \varepsilon)$-approximation algorithm for fixed number of knapsacks and a 14.2378-approximation algorithm for any arbitrary number of knapsacks. Again, with rotation, the bound can be improved to $7.5 + \varepsilon$. For square packing we can achieve a better bound of $3 + \varepsilon$. Therefore we give small constant truthful approximation bounds and improve the previous bounds in [5].

2 The Two-Dimensional Orthogonal Knapsack Problem

The optimization version of the two-dimensional orthogonal knapsack problem is to select a set of rectangles with maximum value such that they can be packed into a given knapsack (a rectangle). Suppose that the knapsack has capacity $C = (a, b)$, where C is a rectangle with width a and height b. In the view of mechanism design, items are controlled by selfish agents, and each rectangle (or item) j has its true type (a_j, b_j, v_j), where $0 < a_j \leq a$ is the width of rectangle j and $0 < b_j \leq b$ is the height of rectangle j, and v_j is the value of this item. Each agent j sends her bid (a'_j, b'_j, v'_j) to a mechanism, and then the mechanism computes the output O and the payments for every agent based on the bids of all agents. Thus, the mechanism for two-dimensional orthogonal knapsack problem is an allocation algorithm A and a payment function p^A. The mechanism's goal is to maximize the social welfare, i.e., the total value of selected items assigned in the knapsack.

Let $d = (d_1, d_2, \ldots, d_n)$ be the bidders of all agents, where d_j is the declaration of agent j and n is the number of agents. If agent j reports her true type, then $d_j = (a_j, b_j, v_j)$. We consider *single-minded* version introduced by Lehmann, Oćallaghan, and Shoham [25]. Let $O(d)$ be the allocations of the mechanism based on the reporting d and each $O_j(d) \in O(d)$ is the allocation for each agent j. Each $O_j(d)$ is a rectangle, and $O_j(d)$ is empty if agent j is not selected. We define $(a_j, b_j) \leq O_j(d)$ if a_j is no more than the width of $O_j(d)$ and b_j is no more than the height of $O_j(d)$, i.e., the rectangle (a_j, b_j) can be packed inside the rectangle $O_j(d)$. The value function $v_j(O_j(d))$ for each output $O_j(d)$ is given as below.

$$v_j(O_j(d)) = \begin{cases} v_j, \, if(a_j, b_j) \leq O_j(d) \\ 0, \;\; otherwise. \end{cases} \tag{1}$$

In addition, we only consider *unknown* size of this problem, meaning that the size of rectangle is only known by the corresponding agent. Each agent j may declare any value of a_j and b_j. An agent j's utility in a mechanism (A, p^A) is

$$u_j(d) = v_j(O_j(d)) - p_j^A$$

by the bidding of d, where p_j^A is the payment for agent j. Each agent attempts to maximize her utility, and thus might manipulate the mechanism by declaring a false type. A mechanism is *truthful* or *incentive compatible*, if no agent j would increase her utility by any false declaration, i.e.

$$u_j((a_j, b_j, v_j), (a'_{-j}, b'_{-j}, v'_{-j})) \geq u_j((a'_j, b'_j, v'_j), (a'_{-j}, b'_{-j}, v'_{-j}))$$

for any declaration (a'_j, b'_j, v'_j).

For any instance I, let $SC(I)$ be the total value obtained by a mechanism, and $OPT(I)$ be the total value obtained by an optimal solution, then the mechanism is ρ-approximation if $\frac{OPT(I)}{SC(I)} \leq \rho$.

2.1 The Mechanism

A bid (a'_j, b'_j, v'_j) of agent j is a *winning declaration* if $(a_j, b_j) \leq O_j$ (this item is selected in the knapsack), otherwise, it is a *losing declaration.* For a bid (a'_j, b'_j, v'_j), a declaration (a''_j, b''_j, v''_j) is said to be a *higher declaration* if $a''_j \leq a'_j$ and $b''_j \leq b'_j$, and $v''_j \geq v'_j$. For the simplification we let $d = (a, b, v) = ((a_1, b_1, v_1), (a_2, b_2, v_2), \ldots, (a_n, b_n, v_n))$. Let (a_{-j}, b_{-j}, v_{-j}) denote the declaration without agent j, which can be represented as

$$((a_1, b_1, v_1), \ldots, (a_{j-1}, b_{j-1}, v_{j-1}), (a_{j+1}, b_{j+1}, v_{j+1}), \ldots, (a_n, b_n, v_n)).$$

Definition 2.1. *(Monotone) We say that an algorithm A for two-dimensional orthogonal knapsack problem is monotone if, for any agent bidder j, (a'_j, b'_j, v'_j) is a winning declaration then any higher declaration also wins.*

From the property of monotone, we observe that algorithm A defines a *critical value* θ_j^A, which is the minimum value v'_j such that (a'_j, b'_j, v'_j) is a winning declaration if we fix the declaration of $(a'_{-j}, b'_{-j}, v'_{-j})$ and declaration (a'_j, b'_j). We say that algorithm A is *exact* if $O_j(d') = (a'_j, b'_j)$ or $O_j(d') = \emptyset$ for each declaration (a'_j, b'_j, v'_j) of d'.

Definition 2.2. *The payment p^A associated with the monotone allocation algorithm A that is based on the critical value is defined by $p_j^A = \theta_j^A$ if agent j wins, and $p_j^A = 0$ otherwise.*

A mechanism $M_A = (A, p^A)$ is *normalized* , if its payment p^A is defined as in Definition 2.2, i.e. agents that are not selected pay 0.

Theorem 2.3. [10] *Let A be a monotone and exact algorithm for some utilitarian problem and single-minded agents. Then the normalized mechanism $M_A = (A, p^A)$ is truthful.*

Proof. Briest et al. [10] showed that the theorem is valid for a single dimensional knapsack problem, generalized from the combinatorial auction problem [25]. This result can be easily extended to our 2-dimensional orthogonal knapsack problem as it is utilitarian. For the sake of completeness, we give the sketch of the proof.

For any agent j, let us fix the declarations of any other agents. The true type of agent j is (a_j, b_j, v_j). The first step is to prove that the utility function of the declaration (a'_j, b'_j, v_j) is at least that of the arbitrary declaration (a'_j, b'_j, v'_j) for any v'_j. It is worth to mention that the critical value is independent of $j's$ declaration of v'_j. Let θ_j be the critical value of declaration (a'_j, b'_j, v'_j). If agent j is selected or not selected in both declarations, the utilities are the same. If the agent j is not selected in declaration (a'_j, b'_j, v_j) and selected in declaration (a'_j, b'_j, v'_j), we have $v'_j \geq \theta_j > v_j$. Thus, the utility of agent j in declaration of (a'_j, b'_j, v'_j) is negative, while the utility of agent j in declaration of (a'_j, b'_j, v_j) is zero. Conversely, if the agent j is selected in declaration (a'_j, b'_j, v_j) and not selected in declaration (a'_j, b'_j, v'_j), we have the utility of agent j is non-negative and zero for these two declarations, respectively.

The second step is to show the utility of declaration of (a_j, b_j, v_j) is no less than (a'_j, b'_j, v_j) for any (a'_j, b'_j). If $a'_j < a_j$ or $b'_j < b_j$, from the exactness, the value of agent j is zero and hence the utility is non positive. Observe that the utility of a truth declaration is non-negative. Therefore, the utility is not increasing by such a kind of lying.

Let us focus on the lying that $a'_j \geq a_j$ and $b'_j \geq b_j$. Let θ_j and θ'_j be the critical values according to the true declaration and the lying declaration, respectively. We have $\theta_j \leq \theta'_j$ from the monotonicity of the allocation algorithm A. If the agent j is not selected by lying declaration, its utility is zero, while the true declaration is non-negative. If the agent j is selected in both declaration, the utility of true declaration is $v_j - \theta_j$ that is no less than the utility of lying declaration $v_j - \theta'_j$. Now, we only consider that the agent j is not selected in true declaration. In this case we have $v_j < \theta_j \leq \theta'_j$, which indicates the utility of agent j is negative if it is selected by lying to (a'_j, b'_j, v_j). □

Hence it is sufficient to design a monotone allocation algorithm to obtain a truthful mechanism for our problem by Theorem 2.3. To this end, in the following, we provide a composition algorithm.

2.2 Composition Algorithm

Given two algorithms A and B, we define the composition of algorithm $A \circ B$ in the following way: For any given input I, run the algorithm A on I and let O_1

be the set of winners. Then run the algorithm B on O_1 and let O_2 be the set of winners. The output of algorithm $A \circ B$ is the allocation of items in O_2.

For any instance I of the two-dimensional knapsack problem (a, b, v), we define a new instance I', which is a one-dimensional knapsack problem (s, v), where $s_j = a_j \cdot b_j$ is the area of the rectangle item j, and the capacity of the knapsack is $C = a \cdot b$. A bit overuse of notations, in the remainder of this paper, C is the area of the rectangle (a, b) when we refer to the capacity of the one-dimensional knapsack problem, and C is the rectangle (a, b) when we refer to the two-dimensional knapsack.

Composition Algorithm $A \circ B$

1. Run 1-dimensional knapsack algorithm A for the new instance I', return the set of selected items O_1.
2. Run algorithm B on the instance O_1 for 2-dimensional orthogonal knapsack problem, return the set of selected items O_2.

Theorem 2.4. *If the algorithm A and the algorithm B are both monotone, then its composition algorithm $A \circ B$ is monotone.*

Proof. Suppose that the agent j with true type (a_j, b_j, v_j) is a winning declaration. Then we need to prove that a higher declaration $j' = (a'_j, b'_j, v'_j)$ is also a winner, i.e. $a_j \geq a'_j$, $b_j \geq b'_j$ and $v_j \leq v'_j$. Since j is a winner, then $j \in O_2$ and we have $j \in O_1$ too. It holds that $j' \in O_1$ since algorithm A is monotone. We know $j' \in O_2$ because of the algorithm B is monotone, which therefore j' is also a winner. $\square$

2.3 Monotone Algorithm

An item R_j is called *big* if $a_j > a/2$ and $b_j > b/2$; it is *wide* if $a_j > a/2$ and $b_j \leq b/2$; it is *tall* if $a_j \leq a/2$ and $b_j > b/2$; and it is small if $a_j \leq a/2$ and $b_j \leq b/2$.

Lemma 2.5. [21] *If the total area of a set T of items is at most $C/2$ and there are no tall items (or there are no wide items), then the items in T can be packed into a bin with capacity C.*

Now we are ready to show a monotone algorithm for the two-dimensional orthogonal knapsack problem.

Lemma 2.6. *Algorithm 1 is monotone.*

Proof. Note that Algorithm 1 is composable, which consists of algorithms A and B. For any agent j that is a winner, if its declaration is higher, we have $s'_j \leq s_j$ and $v'_j \geq v_j$. The algorithm A in line 2 has already been proved to be monotone in the accordingly references.

For algorithm B, the monotone is shown as below. If the output of original declaration is due to G_1, i.e., it returns the m items with largest value, then, clearly, it is monotone since item j will also be a winner from $v'_j \geq v_j$.

Algorithm 1.. Allocation algorithm for 2-dimensional knapsack problem

```
1: For any input $I$ $(a_i, b_i, v_i)$, the new instance $I'$ is $(s_i, v_i)$, where $s_i = a_i \cdot b_i$, and the
   capacity of the knapsack is $C = ab$, where $(a, b)$ is the capacity of the knapsack in
   input $I$.
2: **Algorithm A**: Run a monotone algorithm $A$ for 1-dimensional knapsack prob-
   lem on this new instance $I'$. Let $O_1$ be the output, and $V(O_1)$ be the value of
   total selected items. Specifically, in case of single knapsack, we adopt the mono-
   tone FPTAS [10] for the 1-dimensional knapsack problem as algorithm $A$. For
   multiple knapsack problem with fixed number of knapsacks, we adopt the mono-
   tone PTAS [10] for the general assignment problem as algorithm A. For multiple
   knapsack problem with arbitrarily number of knapsacks, we adopt the monotone
   algorithm in [12] as algorithm A.
3: **Algorithm B**: Let $m$ be the number of knapsacks. Select $m$ items with maximal
   value from $O_1$. Denote these $m$ items as $G_1$. Let $\alpha$ be a constant that will be given
   later.
4: **if** $V(G_1) \geq \alpha V(O_1)$, **then**
5:     **return** Assign each item in $G_1$ to a different knapsack respectively.
6: **else**
7:     Remove all the big items from $O_1$.
8:     Then consider two sets of remaining items, $T_1$ consists of wide items and small
       items, $T_2$ consists of tall items and small items. We choose the set $T_h$ with larger
       value, i.e. $V(T_h) = \max(V(T_1), V(T_2))$.
9:     We adopt the monotone algorithm A for 1-dimensional knapsack problem on the
       instance $T_h$. However, we set the capacity of the knapsacks to be $C/2$. Denote
       the selected items in this step as $G_2$.
10:    For the output $G_2$, we adopt the 2-dimensional packing algorithm as indicated
       in Lemma 2.5 by Jansen and Zhang [21] to pack these items in the original
       knapsack with capacity $C = (a, b)$.
11: **end if**
```

If the output of original declaration is G_2, fixing other declarations, agent j reports a higher declaration j'. We denote it as d'. The output of d' is either G_1' or G_2'. If it is G_1', i.e. the total value of the largest m items is larger than $\alpha V(O_1)$, noting that the total value of the largest m items in original declarations is smaller than $\alpha V(O_1)$ due to the output is G_2, then item j must be selected.

If the output of d' is G_2', we know that all big items are not included in T_h. The item j is a wide item (or tall) or a small item. If j is a small item, its higher bidder j' is also a small item. If j is a wide (or tall) item, its higher bidder either is a wide (or tall) item or a small item, i.e. the item with a high declaration is also in T_h. A higher declaration of item j ensures that item j must be selected due to the monotone of the algorithm A. Thus, the algorithm B is monotone. Since, algorithm A and B are both monotone, by Theorem 2.4, Algorithm 1 is monotone. □

Lemma 2.7. *Let the approximation ratio of algorithm A be $\rho_A \geq 1$. For any given $\varepsilon > 0$, the approximation ratio of Algorithm 1 for $m = 1$ is $7\rho_A^2$ by letting*

$\alpha = 1/7$. *The approximation ratio of Algorithm 1 for* $m \geq 2$ *is* $9\rho_A^2$ *by letting* $\alpha = 9$.

Proof. Let V^{opt} be the value of selected items achieved by any optimal algorithm. We know $V^{opt} \leq \rho_A V(O_1)$. Let $V(O_2)$ be the value of selected items achieved by Algorithm 1. If the algorithm B outputs the m items with the maximum value, we have $V(O_2) \geq \alpha V(O_1)$, and then $V^{opt} \leq \frac{\rho_A}{\alpha} V(O_2)$ follows.

If the final accepted items are due to G_2, we show its value is at least $\alpha V(O_1)$. In this case, the maximum value among these items is no more than $\alpha V(O_1)$.

Note that in T_h, there are no both wide items and tall items. W.l.o.g, we assume there is no tall item, i.e. each item has height at most of $b/2$. According to the algorithm B, we apply the algorithm A to select items among T_h with the capacity of a knapsack $C/2$.

Now we are going to find a lower bound of the optimal value of selecting items in T_h with capacity $C/2$, which is denoted by $V^*(T_h)$. For the O_1 items in each knapsack, it is a feasible solution for a single knapsack with capacity C. Let us consider each knapsack j. Denote the value of items in the knapsack j to be $V(T_h, j)$, and we have $V(T_h) = \sum_{j=1}^{m} V(T_h, j)$. We split the knapsack j into two identical sub-knapsacks, each with capacity $C/2$ with width a and height $b/2$. See Figure 1 for an illustration. We get three sets of items, the items below the divided line, the item crossing the divided line, and the items above the divided line, respectively. It is worth to note that the area of all these three sets is at most of $C/2$. Then select one set with maximum value in each knapsack implies that the optimal value is at least $V(T_h, j)/3$ for any m knapsacks.

On the other hand, if we remove $\gamma \geq 2$ big items in the knapsack j, then the total area in this knapsack is at most of $C/2$, and all items from T_h in knapsack j can be selected by an optimal algorithm with capacity $C/2$.

In all, the value of the selected items in an optimal solution in knapsack j is at least $V(T_h, j)/3$ if $\gamma \leq 1$, or at least $V(T_h, j)$ if $\gamma \geq 2$, where γ is the number of big items in knapsack j from O_1.

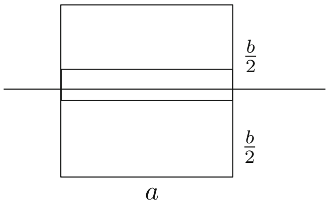


Fig. 1. Illustration of the split knapsack

Regarding the approximation ratios, let us first consider $m = 1$, i.e., the single knapsack problem. Suppose that γ big items are removed, the value lost is at most $\gamma\alpha V(O_1)$. In case of $m = 1$, we have $\gamma \leq 3$. Hence the total value of items in O_1 without counting removed items is at least $(1-\gamma\alpha)V(O_1)$, then the value of the selected items T_h is $V(T_h) = \max(V(T_1), V(T_2)) \geq \frac{1-\gamma\alpha}{2}V(O_1)$.

We have $V^*(T_h) \geq V(T_h)/3 \geq \frac{1-\alpha}{6}V(O_1)$ if $\gamma \leq 1$, and $V^*(T_h) \geq V(T_h) \geq \frac{(1-3\alpha)}{2}V(O_1)$ if $\gamma \geq 2$. It holds that $V^*(T_h) \geq \min\{\frac{1-\alpha}{6}V(O_1), \frac{(1-3\alpha)}{2}V(O_1)\}$. Let $\alpha = 1/7$. Note that $\min\{\frac{1-\alpha}{6}, \frac{1-3\alpha}{2}\} = 1/7$.

$$\begin{aligned} V(O_2) &\geq V^*(T_h)/\rho_A \\ &\geq \min\{\alpha, \frac{1-\alpha}{6}, \frac{1-3\alpha}{2}\}V(O_1)/\rho_A \\ &= \frac{1}{7}V(O_1)/\rho_A \geq \frac{1}{7\rho_A^2}V^{opt}. \end{aligned}$$

Now we consider the approximation ratios for multiple knapsacks, in which $m \geq 2$. W.l.o.g, we assume that each of the first k knapsacks removes at most one big item. Then in the left $m-k$ knapsacks, each will remove at least two big items. Hence, we know $V^*(T_h) \geq \frac{1}{3}\sum_{j=1}^{k} V(T_h, j) + \sum_{j=k+1}^{m} V(T_h, j) \geq V(T_h)/3$. On the other hand, $2V(T_h) \geq V(O_1) - \sum_{j=1}^{m} V(B, j)$, where $V(B, j)$ is the total value of big jobs in knapsack j in O_1. From the fact that the final output is G_2, the value of the largest m items is at most $\alpha V(O_1)$, and there are at most $3m$ big items, we have $\sum_{j=1}^{m} V(B, j) \leq 3\alpha V(O_1)$. Thus, $V^*(T_h) \geq V(T_h)/3 \geq \frac{1-3\alpha}{6}V(O_1)$.

The approximation ratio $9\rho_A^2$ follows from the following inequality by letting $\alpha = 9$.

$$\begin{aligned} V(O_2) &\geq \min\{\alpha V(O_1), V^*(T_h)/\rho_A\} \\ &\geq \min\{\alpha, \frac{1-3\alpha}{6}\}V(O_1) \\ &\geq \min\{\alpha, \frac{1-3\alpha}{6}\}V^{opt}/\rho_A^2. \end{aligned}$$

□

Theorem 2.8. *Let the mechanism employ Algorithm 1 as the allocation algorithm and adopt the associated critical value payment scheme. Then it is an approximation truthful mechanism for a number of knapsack problems including the 2-dimensional orthogonal knapsack problem, the multiple orthogonal knapsack problem with a constant number of knapsacks, and the multiple orthogonal knapsack problem with arbitrary number of knapsacks. The respective approximation ratios are at most $7+\varepsilon$, $9+\varepsilon$, and $9e/(e-1)+\varepsilon$, for any fixed $\varepsilon > 0$.*

Proof. The monotonicity has been proved from Lemma 2.6. Let us consider the approximation ratios by Lemma 2.7. For $m = 1$, let $\delta > 0$ be an arbitrarily small positive number, we adopt the monotone FPTAS [10] as the algorithm A, hence $\rho_A = 1 + \delta$. Thus, let $\varepsilon > 0$, and select δ such that $14\delta + 7\delta^2 \leq \varepsilon$, the approximation ratio for single 2-dimensional knapsack problem is at most $7 + \varepsilon$.

For $m \geq 2$, Briest, Krysta, and Vöcking [10] provided a monotone PTAS for the generalized assignment problem with constant number of knapsacks, where knapsack problem is a special case of generalized assignment problem. Hence, for any $\varepsilon > 0$, we obtain a $(9+\varepsilon)$-approximation algorithm for any fixed m knapsacks. While, for arbitrary number of knapsacks, Chekuri and Gamzu [12] provided $e/(e-1)+\varepsilon$ monotone algorithm for multiple knapsack problem with identical capacity, which indicates a $9e/(e-1)+\varepsilon \approx 14.2378+\varepsilon$ for our problem. □

2.4 Packing Square Items or Allowing Rotation

In this section, we consider a special case of packing rotatable items, i.e., the allocation algorithm allows to rotate an item by 90 degrees. In particular, we consider the items are squares, i.e. for each item i we have $a_i = b_i$. These two special cases have a common property that an algorithm can always ensure that there are no both wide items and tall items. In square items version, all items are either a big item or a small item. In the rotatable version, one can rotate a tall item or a wide item such that only one kind of such item remains.

The key idea to improve the approximation ratio is that we do not need to choose between wide items and tall items. We revise allocation algorithm B in Algorithm 1 as below: Apply algorithm A to solve the instance O_1 after rotating all tall items into wide items if necessary, while the capacity of the knapsack is set to $C/2$.

Corollary 2.9. *If all the items are squares or rotation of items are allowed in the allocation algorithm, then for any given $\varepsilon > 0$, assuming the approximation ratio of algorithm A in Algorithm 1 to be ρ_A, the approximation ratio for the multiple 2-dimensional knapsack problem is $3\rho_A^2$.*

Proof. The monotonicity of the revised algorithm is analogous to Lemma 2.6, because any higher declaration of an item from T_h still belongs to T_h because there are no both width items and tall items.

Lemma 2.5 guarantees the feasibility of packing of the selected items. Let us consider the approximation ratio. We have the optimal value $V^*(T_h)$ for the knapsack problem with capacity $C/2$ is at least $V(O_1)/3$ from the proof of Lemma 2.7. Let $V(O_2)$ be the value of final selected items by our algorithm, and V^{opt} be the optimal value. Clearly, $V^{opt} \leq \rho_A \cdot V(O_1)$ and $V(O_1)/3 \leq V^*(T_h) \leq \rho_A \cdot V(O_2)$. Therefore, we have $V^{opt} \leq 3\rho_A^2 \cdot V(O_2)$, which gives the approximation ratio at most of $3\rho_A^2$. □

Remark: For $m = 1$ and multiple knapsacks with constant number of knapsacks, the approximation ratio for the case of square items or with rotation is $3+\varepsilon$ due to algorithm A is $1+\varepsilon$ approximated [10]. For multiple knapsacks with an arbitrary number of knapsacks, the approximation ratio is $3(e/(e-1))^2+\varepsilon \approx 7.5+\varepsilon$ due to algorithm A [12] is $e/(e-1)+\varepsilon$ approximated.

3 Concluding Remarks

We have presented techniques for designing truthful mechanisms for the two-dimensional orthogonal knapsack problem. We hope that our technique can be flexible and extended to other problems with composition algorithms. Note that for the two-dimensional orthogonal knapsack problem, if item sizes are public, then the $(3+\varepsilon)$-approximation algorithm provided by Jansen and Zhang [21] is a truthful mechanism. Recall that the non-strategic algorithm achieves an upper bound of $2+\varepsilon$ [21]. It is worth to see a better truthful mechanism. Another interesting problem is to see if it is possible to design a truthful PTAS for the multiple one-dimensional knapsack problem with an arbitrary number of knapsacks, which therefore reduces the approximation ratio for the multiple two-dimensional orthogonal knapsack problems automatically.

References

1. Andelman, N., Azar, Y., Sorani, M.: Truthful approximation mechanisms for scheduling selfish related machines. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 69–82. Springer, Heidelberg (2005)
2. Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 482–491 (2001)
3. Archer, A.F.: Mechanisms for discrete optimization with rational agents. Ph.D. thesis, Cornell University (2004)
4. Auletta, V., De Prisco, R., Penna, P., Persiano, G.: Deterministic truthful approximation mechanisms for scheduling related machines. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 608–619. Springer, Heidelberg (2004)
5. Babaioff, M., Blumrosen, L.: Computationally-feasible truthful auctions for convex bundles. Games and Economic Behavior **63**(2), 588–620 (2008)
6. Bougeret, M., Dutot, P.-F., Jansen, K., Otte, C., Trystram, D.: A fast 5/2-approximation algorithm for hierarchical scheduling. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010, Part I. LNCS, vol. 6271, pp. 157–167. Springer, Heidelberg (2010)
7. Bougeret, M., Dutot, P.F., Jansen, K., Otte, C., Trystram, D.: Approximating the non-contiguous multiple organization packing problem. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 316–327. Springer, Heidelberg (2010)
8. Bougeret, M., Dutot, P.F., Jansen, K., Otte, C., Trystram, D.: Approximation algorithms for multiple strip packing. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 37–48. Springer, Heidelberg (2010)
9. Bougeret, M., Dutot, P.F., Trystram, D.: An extention of the 5/2-approximation algorithm using oracle. Research Report (2010)
10. Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. SIAM Journal on Computing **40**(6), 1587–1622 (2011)
11. Caprara, A., Kellerer, H., Pferschy, U.: The multiple subset sum problem. SIAM Journal on Optimization **11**, 308–319 (2000)
12. Chekuri, C., Gamzu, I.: Truthful mechanisms via greedy iterative packing. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) Approximation, Randomization, and Combinatorial Optimization. LNCS, vol. 5687, pp. 56–69. Springer, Heidelberg (2009)
13. Chekuri, C., Khanna, S.: On multi-dimensional packing problems. In: Proceedings of the 10th annual ACM-SIAM symposium on Discrete Algorithms (SODA), pp. 185–194 (1999)

14. Christodoulou, G., Kovács, A.: A deterministic truthful ptas for scheduling related machines. In: Proceedings of the 21th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1005–1016 (2010)
15. Coffman, E.G., Garey, M.R., Johnson, D.S., Tarjan, R.E.: Performance bounds for level oriented two-dimensional packing algorithms. SIAM Journal on Computing **9**, 808–826 (1980)
16. Dhangwatnotai, P., Dobzinski, S., Dughmi, S., Roughgarden, T.: Truthful approximation schemes for single-parameter agents. In: Proceedings of 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 15–24 (2008)
17. Dughmi, S., Roughgarden, T.: Black-box randomized reductions in algorithmic mechanism design. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 775–784 (2010)
18. Fishkin, A.V., Gerber, O., Jansen, K., Solis-Oba, R.: On packing rectangles with resource augmentation: Maximizing the profit. Algorithmic Operations Research **3**(1), 1–12 (2008)
19. Grandoni, F., Krysta, P., Leonardi, S., Ventre, C.: Utilitarian mechanism design for multi-objective optimization. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 573–584 (2010)
20. Jansen, K.: Parameterized approximation scheme for the multiple knapsack problem. SIAM Journal on Computing **39**, 1392–1412 (2009)
21. Jansen, K., Zhang, G.: Maximizing the total profit of rectangles packed into a rectangle. Algorithmica **47**(3), 323–342 (2007)
22. Kellerer, H.: A polynomial time approximation scheme for the multiple knapsack problem. In: Hochbaum, D.S., Jansen, K., Rolim, J.D.P., Sinclair, A. (eds.) RANDOM 1999 and APPROX 1999. LNCS, vol. 1671, pp. 51–62. Springer, Heidelberg (1999)
23. Kulik, A., Shachnai, H.: There is no eptas for two-dimensional knapsack. Information Processing Letters **110**(16), 707–710 (2010)
24. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. Journal of the ACM **58**(6), 25 (2011)
25. Lehmann, D., Oćallaghan, L.I., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. Journal of the ACM (JACM) **49**(5), 577–602 (2002)
26. Mu'Alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions. Games and Economic Behavior **64**(2), 612–631 (2008)
27. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proceedings of the thirty-first annual ACM Symposium on Theory of Computing (STOC), pp. 129–140 (1999)
28. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM Journal on Computing **26**, 401–409 (1997)
29. Ye, D., Han, X., Zhang, G.: Online multiple-strip packing. Theoretical Computer Science **412**(3), 233–239 (2011)
30. Ye, D., Zhang, G.: Coordination mechanisms for selfish parallel jobs scheduling - (extended abstract). In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 225–236. Springer, Heidelberg (2012)
31. Zhuk, S.: Approximate algorithms to pack rectangles into several strips. Discrete Mathematics and Applications **16**(1), 73–85 (2006)

Online Integrated Allocation of Berths and Quay Cranes in Container Terminals with 1-Lookahead

Jiayin Pan[1,2(✉)] and Yinfeng Xu[1,2]

[1] School of Management, Xi'an Jiaotong University, Xi'an 710049, China
panjy1991@stu.xjtu.edu.cn
[2] The State Key Lab for Manufacturing Systems Engineering, Xi'an 710049, China
yfxu@mail.xjtu.edu.cn

Abstract. This paper studies an online over-list model of the integrated allocation of berths and quay cranes in container terminals with 1-lookahead ability. The objective is to minimize the maximum completion time of container vessels, i.e., the makespan. We focus on two different types of vessels, three berths and a small number of QCs in the hybrid berth layout, with 1-lookahead information. We propose a $(1+\sqrt{2})/2$-competitive algorithm for the case with 4 cranes and a 5/4-competitive algorithm for the case with 5 cranes, respectively. Both of the algorithms are proved to be optimal.

Keywords: Scheduling · Online algorithm · Lookahead information · Container terminal

1 Introduction

Recently, the optimization problems involved in seaside operations planning attract increasing attention in the operation research and transportation research literatures. The seaside operations planning in container terminals basically comprises the berth allocation problem (BAP), the quay crane assignment problem (QCAP), and the quay crane scheduling problem (QCSP). Since BAP and QCAP are much interrelated in practice, a trend towards an integrated solution of berth and QC(quay crane) resources is observed in the recent literatures (see Bierwirth and Meisel, 2010; Carlo *et al.*, 2013)

Most literatures investigated the offline version of the BAP-QCAP problem such that the scheduler knows the complete information of all vessels at the beginning. Park and Kim (2003) provided pioneering integration approaches, they decided on the berthing position, the berthing time, and the number of cranes to assign to each vessel together. Lokuge and Alahakoon (2007) studied a dynamic BAP-QCAP problem with hybrid berths, aiming at minimizing total waiting time and minimizing total tardiness. They developed a multi-agent system which constitutes a feedback loop integration of the BAP and the QCAP. Giallombardo *et al.*(2010), Blazewicz *et al.*(2011) and Zhen *et al.*(2011) presented a mixed integer programming and gave a tatu search algorithm for a

DOI: 10.1007/978-3-319-21398-9_32

dynamic BAP-QCAP problem in the discrete berth layout. Liang *et al.*(2011) considered a dynamic discrete BAP-QCAP problem as a non-linear bi-objective programming model. Chen *et al.* (2012) presented a mixed integer programming to solve the problem which the objective was minimizing tardiness.

Zhang *et al.*(2008) studied the online version of container vessel scheduling at the earliest. They considered the QC scheduling problem with non-crossing constraints, and considered both online over-list and over-time model, with the objective of minimizing the makespan. Zhang *et al.* developed an algorithm that proved a $m/\lceil \log 2(m+1) \rceil$-competitive for the over-list model and an algorithm that proved a 3-competitive for the general model of over-time model where m is the number of QCs.

There are some related studies about online model with lookahead information in the field of scheduling. Mandelbaum *et al.* (2010) studied online parallel machines scheduling with k-lookahead, where k is the number of lookahead jobs. For the objective of minimizing the makespan, they found a 1-competitive online algorithm when there were only two job types. Zheng *et al.*(2013) studied online single machine scheduling, also with k-lookahead jobs case, showed that the lookahead ability can effective improve the competitive performance of online strategy.

In this paper, we investigate the online vesion of the BAP-QCAP problem with 1-lookahead in the hybrid berth layout (Imai *et al.*, 2005), focus on the over-list model that vessels are released one by one. As in practice, the vessels dynamically arrive at the container terminal. The scheduler can foresee the information of the vessels that arrive in the next a couple of days, and produce a schedule to be executed in the next day for the vessels based on FCFS rule. Namely, the scheduler makes the decision on the assignment of a current vessel with the information of the current vessel and next k vessels. Hence, we model the above FCFS-based assignment pattern as the online over-list BAP-QCAP problem with k-lookahead information, and evaluate the performance of such kind of strategies. More precisely, we consider the cases with three berths, four or five QCs, and have 1-lookahead information. We prove that there exist optimal online algorithms with competitive ratios of $(1+\sqrt{2})/2$ and $5/4$, respectively.

The rest of this work is organized as follows. Section 2 describes the problem and gives some notations. In Section 3 and 4, we deal with the case with 4 and 5 QCs respectively, and present a matching upper and lower bound for each case. Finally Section 5 concludes this work.

2 Problem Statement and Basic Notations

There are some vessels that request service (loading or unloading containers) by berths and QCs in the quay. In this paper, we only consider two types of vessels, each type corresponds to a service request with uniform vessel size and processing load. More precisely, the small request corresponds to small vessel size which needs one berth to assign, and its processing load is equal to one. However, the large request corresponds to large vessel size which needs two consecutive berths to assign, and its processing load is equal to Δ (≥ 2). For simplicity, let $r_i = 1$ and $r_i = \Delta$ denote a small request and a large request respectively. Considering the physical restrictions imposed on the number of QCs to be assigned to a vessel, we further assume that a small (or large) request can be serviced by two (or four) QCs at most simultaneously. Namely, the processing time of a small (or large) request r_i is equal to $1/m_i$ (or Δ/m_i) units of time where m_i is the number of QCs for processing the request. A list of requests $\mathcal{I} = \{r_1, r_2, \ldots, r_n\}$ ($n \geq 1$) are to be released one by one. The scheduler must immediately allocate the service combination of berth and QC as well as starting time. In addition to the information about the current request in list, the scheduler has all the information about the next 1 request in the list. The objective is to minimize the makespan, i.e., the end time of the last completed request.

As both berth and QC are expensive resources in the container terminal, generally, many ports in China have very limited number of berths and QCs. Hence, we focus on the scenario with three consecutive berths. Which serve either three small requests or one small request and one large request simultaneously, implying there at most need 6 QCs. According to the actual situation, in this paper, we consider 4 and 5 QCs. The QCs move in the same rail, satisfying the non-cross constraint. We denote the three berths by b_1, b_2 and b_3 from left to right, and by $q_1, q_2, \ldots, q_m$ the m ($4 \leq m \leq 6$) QCs from left to right in the quay.

For a request r_i, we denote its start and end time by s_i and e_i. $t_{i,m}$ ($m = \{1, 2, 3\}$) denotes the earliest time by which at least m consecutive berths have completed all of their currently allocated requests. $C_{i,j}$ denotes the earliest time by which berth b_j has completed all of its currently allocated requests.

Adopting the quadruple notation scheme in Bierwirth and Meisel (2010), we denotes this problem as $hybr\ |online - over - list,\ LD = 1\ |BAP - QCAP|\ C_{\max}$, where LD means the number of lookahead requests.

To evaluate the performance of an online strategy $\mathcal{A}$, we often use the competitive ratio (Borodin and El-Yaniv, 1998). For any request input instance $\mathcal{I}$, let $C_{\mathcal{A}}(\mathcal{I})$, $C^*(\mathcal{I})$ be the makespan of schedule produced by an online algorithm $\mathcal{A}$ and that of an optimal schedule respectively. Then algorithm $\mathcal{A}$ is ρ-competitive where

$$\rho = \inf_r \left\{ r \,\middle|\, \frac{C_{\mathcal{A}}(\mathcal{I})}{C^*(\mathcal{I})} \leq r \right\}$$

3 The Case with Four QCs

In this section we focus on the case with 3 berths and 4 QCs in the problem $hybr\ |online - over - list,\ LD = 1\ |BAP - QCAP|\ C_{\max}$. Firstly, we analyze the lower bound, i.e., no online algorithm can perform better than it from a worst-case point of view. Secondly, we present an online algorithm named MLIST. Before analyzing the lower bound and algorithm, we first introduce two definitions.

Definition 1. *Waste time segment* and *available time segment:* For an idle time segment $[s_i, e_i)$, let q be the number of idle QCs. If $0 < q(e_i - s_i) < 1$, then $[s_i, e_i)$ is a waste time segment, denoted by T_w. Otherwise, $[s_i, e_i)$ is an available time segment, denoted by T_a.

The segment of each request r_i may induce a forced idle time segment, i.e., an idle time segment $[s_i, e_i)$, only for during $[s_i, e_i)$, there exist idle QCs. If $0 < q(e_i - s_i) < 1$, where q is the number of the idle QCs, even a small request later on cannot be satisfied within the time segment, and thus $[s_i, e_i)$ is a waste time segment. Otherwise, $[s_i, e_i)$ is an available time segment. Set T_a=$[t_1$,$t_2)$ where t_1=s_i, t_2=e_i, and T_a=[0,0) initially. And let $|T_a|$ denote the total available time, $|T_w|$ denote the total waste time.

3.1 Lower Bound

Theorem 1. *For problem* $hybr\,|online - over - list,\ LD = 1\ |BAP - QCAP|$ $C_{\max}$*, any online scheduling algorithm has a competitive ratio of at least* $(1 + \sqrt{2})/2$*.*

Proof. To prove the theorem, we construct a request input sequence $\mathcal{I}$. $\mathcal{I}$ contains at least two requests. Let $C_{max}(\mathcal{I})$ and $C^*(\mathcal{I})$ be the makespan of schedule produced by $\mathcal{A}$ and an optimal offline algorithm *OPT*, respectively.

According to the requires of the request for the berth and quay cranes, if request $r_i = \varDelta$, then $\mathcal{A}$ has four choices: processes r_i on two consecutive berths with one, two, three or four QCs. For processing r_i with one QC, there always exists at least one idle QC during $[s_i, e_i)$, $\mathcal{A}$ won't select it. If request $r_i = 1$, then $\mathcal{A}$ has two choices: processes r_i on a single berth with one or two QCs.

Assume $\mathcal{I}$ contains the first request $r_1 = \varDelta$, with lookahead $r_2 = 1$. If $\mathcal{A}$ processes r_1 with four QCs on berths b_1, b_2, no more requests arrive, which implying $C_{\max}(\mathcal{I}) \geq 1/2 + \varDelta/4$. While *OPT* processes the requests r_1 and r_2 with three and one QC respectively, at time 0. $C^*(\mathcal{I}) = \max\{\varDelta/3, 1\}$. Except $\varDelta \geq 6$, *OPT* processes the request r_1 and r_2 with four and two QCs respectively, at time 0. $C^*(\mathcal{I}) = 1/2 + \varDelta/4$. If $\mathcal{A}$ processes r_1 with three QCs on berth b_1, b_2, offline adversary releases request $r_3 = 1$ with no more requests, implying $C_{\max}(\mathcal{I}) \geq \min\{\max\{\varDelta/3, 2\}, 1/2 + \varDelta/3\}$. While *OPT* processes the request r_1 with four QCs on berths b_1, b_2, r_2 with two QCs on berth b_1, and r_3 with two QCs on berth b_2, at time 0. $C^*(\mathcal{I}) = 1/2 + \varDelta/4$. If $\mathcal{A}$ processes r_1 with two QCs on berth b_1, b_2, then no more requests arrive. $C_{\max}(\mathcal{I}) \geq \varDelta/2$, while *OPT* processes the request r_1 and r_2 with three and one QC respectively,at time 0. $C^*(\mathcal{I}) = \max\{\varDelta/3, 1\}$. Except$\varDelta \geq 6$, *OPT* processes the request r_1 and r_2 with four and two QCs respectively, at time 0, $C^*(\mathcal{I}) = 1/2 + \varDelta/4$.

After processed. We have $\rho \geq (1 + \sqrt{2})/2$, setting $\varDelta = (6 + 12\sqrt{2})/7$. For the sake of completeness, this middle paragraph is presented in Appendix A. The theorem follows. □

3.2 Algorithm MLIST

On the release of any request r_i $(i \geq 1)$, if $i = n$, then the scheduler can lookahead r_n is the last request of consequence $\mathcal{I}$, so we can optimal assign this request to minimize $C_{\max}$. Otherwise, $1 \leq i \leq n-1$, MLIST behaves as follows.

Case 1. $r_i = \Delta, r_{i+1} = 1$. There are two subcases in the following:
- Case 1.1. When $(2+2\sqrt{10})/3 \leq \Delta \leq (6+12\sqrt{2})/7$, for the case with $|T_a| \geq 3$, assign r_i to the two idle consecutive berths with three QCs, and reset T_a. Otherwise, set $s_i = \max\{C_{i,1}, C_{i,2}\}$, and assign r_i to the leftmost two berths with three QCs.
- Case 1.2. When $2 \leq \Delta \leq (2+2\sqrt{10})/3$ or $\Delta \geq (6+12\sqrt{2})/7$, assign r_i to the leftmost two berths with four QCs. Set $s_i = \max\{C_{i,1}, C_{i,2}\}$.

Case 2. $r_i = \Delta, r_{i+1} = \Delta$. For the case with $|T_a| \geq 3$, assign r_i to the two idle consecutive berths with three QCs, and reset T_a. Otherwise, assign r_i to the leftmost two berths with four QCs. Set $s_i = \max\{C_{i,1}, C_{i,2}\}$.

Case 3. $r_i = 1, r_{i+1} = 1$. When $T_a = [0,0)$, assign r_i to the berth b_1 with two QCs. Set $s_i = C_{i,1}$. Otherwise, assign r_i to the idle berth, reset T_a.

Case 4. $r_i = 1, r_{i+1} = \Delta$. There are two subcases below:
- Case 4.1. When $(2+2\sqrt{10})/3 \leq \Delta \leq (6+12\sqrt{2})/7$, for the case with $T_a = [0,0)$, assign r_i to the berth b_1 with one QC. Set $s_i = \max\{C_{i,1}, C_{i,2}, C_{i,3}\}$. For the other case, assign r_i to the idle berth, reset T_a.
- Case 4.2. when $2 \leq \Delta \leq (2+2\sqrt{10})/3$ or $\Delta \geq (6+12\sqrt{2})/7$, for the case with $T_a = [0,0)$, assign r_i to the berth b_1 with two QCs, set $s_i = C_{i,1}$. Otherwise, assign r_i to the idle berth, reset T_a.

Theorem 2. *For problem hybr $|online - over - list,\ LD = 1\ |BAP - QCAP|$ $C_{\max}$, with 3 berths and 4 QCs, MLIST is $(1+\sqrt{2})/2$ -competitive.*

Proof. Given any request input sequence $\mathcal{I} = \{r_1, r_2, ..., r_n\}$. Let $C_\sigma(\mathcal{I})$ and $C^*(\mathcal{I})$ be the makespan of a schedule produced by MLIST and by an optimal offline algorithm OPT. Based on the interval of Δ, we consider two cases.

Case 1. $(2+2\sqrt{10})/3 \leq \Delta \leq (6+12\sqrt{2})/7$, there are four subcases in the following.

Case 1.1. The last two requests are $r_{n-1} = \Delta,\ r_n = 1$.

Case 1.1.1. $T_a = [0,0)$, we have $C^*(\mathcal{I}) = C_\sigma(\mathcal{I}) = C_{n-2}(I) + max\{\frac{\Delta}{3}, 1\}$.

Case 1.1.2. $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ *$(k \geq 1)$*, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, and those requests consist of k large and k small requests. Assume that excluding those $\in\|$ requests, $C_{\max} = t$ before r_{n-1}, then $C_\sigma(\mathcal{I}) = t + (k+1)max\{\Delta/3, 1\}$, $C^*(\mathcal{I}) \geq t + (k+1)(\Delta+1)/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.1.3. $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ *$(k \geq 0)$* and $|T_a| = 3$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, those requests consist of k large and k small requests. And $r_{n-2} = 1$. Assume that excluding those 2k requests, $C_{\max} = t$ before r_{n-2}, then $C_\sigma(\mathcal{I}) = t + k\max\{\Delta/3, 1\} + \Delta/3 + 1/2$, $C^*(\mathcal{I}) \geq t + (k+1)(\Delta+1) + 1/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (1+\sqrt{2})/2$.

Case 1.2. The last two requests are $r_{n-1} = \Delta, r_n = \Delta$.

Case 1.2.1. $T_a = [0, 0)$, we have $C^*(\mathcal{I}) = C_\sigma(\mathcal{I}) = C_{n-2}(\mathcal{I}) + \Delta/2$.

Case 1.2.2. $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ $(k \geq 1)$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, and those requests consist of k large and k small requests. Assume that excluding those 2k requests, $C_{\max} = t$ before r_{n-1}, then $C_\sigma(\mathcal{I}) = t + k\max\{\Delta/3, 1\} + \Delta/2$, $C^*(\mathcal{I}) \geq t + [k(\Delta+1) + 2\Delta]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.2.3. $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ $(k \geq 0)$ and $|T_a| = 3$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, those requests consist of k large and k small requests. And $r_{n-2} = 1$. Assume that excluding those $2k$ requests, $C_{\max} = t$ before r_{n-2}, then $C_\sigma(\mathcal{I}) = t + (k+1)\max\{\Delta/3, 1\} + \Delta/4$, $C^*(\mathcal{I}) \geq t + [(k+1)(\Delta+1) + \Delta]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.3. The last two requests are $r_{n-1} = 1, r_n = 1$.

Case 1.3.1. $T_a = [0, 0)$, we have $C^*(\mathcal{I}) = C_\sigma(\mathcal{I}) = C_{n-2}(\mathcal{I}) + 1/2$.

Case 1.3.2. $|T_a| = 1$, it indicates that the list doesn't contain $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ sequence before r_{n-1}, then $C^*(\mathcal{I}) = C_\sigma(\mathcal{I}) = C_{n-2}(\mathcal{I}) + 1/2$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) = 1$.

Case 1.3.3. $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ $(k \geq 1)$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, and those requests consist of k large and k small requests. Assume that excluding those $2k$ requests, $C_{\max} = t$ before r_{n-1} , then $C_\sigma(\mathcal{I}) = t + 1/2 + k\max\{\Delta/3, 1\}$, $C^*(\mathcal{I}) \geq t + [k(\Delta+1) + 2]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.3.4. $|T_a| = 1$ and $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ $(k \geq 1)$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, those requests consist of k large and k small requests. And $r_{n-2} = 1$. Assume that excluding those $2k$ requests, $C_{\max} = t$ before r_{n-2}, then $C_\sigma(\mathcal{I}) \leq t + 1 + k\max\{\Delta/3, 1\}$, $C^*(\mathcal{I}) \geq t + [k(\Delta+1) + 3]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (28 - 4\sqrt{10})/13$.

Case 1.3.5. $|T_a| = \Delta/3$ and $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ $(k \geq 0)$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, those requests consist of k large and k small requests. And $r_{n-2} = \Delta$. Assume that excluding those $2k$ requests, $C_{\max} = t$ before r_{n-2}, then $C_\sigma(\mathcal{I}) = t + 1/2 + k\max\{\Delta/3, 1\}$, $C^*(\mathcal{I}) \geq t + [(k+1)(\Delta+1) + 1]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.4. The last two requests are $r_{n-1} = 1, r_n = \Delta$.

Case 1.4.1. $T_a = [0, 0)$, we have $C^*(\mathcal{I}) = C_\sigma(\mathcal{I}) = C_{n-2}(\mathcal{I}) + max\{\frac{\Delta}{3}, 1\}$.

Case 1.4.2. $|T_a| = \Delta/3$, it indicates $r_{n-2} = \Delta$. Assume $C_{\max} = t$ before r_{n-2}, then $C^*(\mathcal{I}) = C_\sigma(I) = t + \Delta/4 + \max\{\Delta/3, 1\}$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) = 1$.

Case 1.4.3. $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ $(k \geq 1)$, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, and those requests consist of k large and k small requests. Assume that excluding those $2k$ requests, $C_{\max} = t$ before r_{n-1} , then $C_\sigma(\mathcal{I}) = t + (k+1)\max\{\Delta/3, 1\}$, $C^*(\mathcal{I}) \geq t + (k+1)(\Delta+1)/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.4.4. $|T_a| = \Delta/3$ and $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ *(*$k \geq 1$*)*, it indicates that the sequence contains pairs of $(r_i = \Delta, r_{i+1} = 1)$ or $(r_i = 1, r_{i+1} = \Delta)$ before r_{n-1}, and those requests consist of k large and k small requests, and $r_{n-2} = \Delta$. Assume that excluding those $2k$ requests, $C_{\max} = t$ before r_{n-2}, then $C_\sigma(\mathcal{I}) = t + (k+1)\max\{\Delta/3, 1\} + \Delta/4$, $C^*(\mathcal{I}) \geq t + [(k+1)(\Delta+1) + \Delta]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 1.4.5. $|T_a| = 1$ and $|T_w| = k(\Delta/3 - 1)$ or $3k(1 - \Delta/3)$ *(*$k \geq 0$*)*, it indicates that the sequence contains pairs of $(r_i = \Delta,\ r_{i+1} = 1)$ or $(r_i = 1,\ r_{i+1} = \Delta)$ before r_{n-1}, and those requests consist of k large and k small requests, and $r_{n-2} = 1$. Assume that excluding those $2k$ requests, the $C_{\max} = t$ before r_{n-2}, then $C_\sigma(\mathcal{I}) = t + 1/2 + k\max\{\Delta/3, 1\} + \Delta/4$, $C^*(\mathcal{I}) \geq t + [(k+1)(\Delta+1) + 1]/4$, $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (8\sqrt{10} - 20)/5$.

Case 2. Otherwise, $2 \leq \Delta \leq (2 + 2\sqrt{10})/3$ or $\Delta \geq (6 + 12\sqrt{2})/7$. In this condition, the algorithm is the same as that in Zheng *et al.*, we can refer to their proof(see Appendix B). With the limit of interval of Δ, we can get $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq (1 + \sqrt{2})/2$.

The theorem follows. $\square$

4 The Case with Five QCs

In this section we focus on the case with 3 berths and 5 QCs in the problem $hybr\,|online - over - list,\ LD = 1\,|BAP - QCAP|\,C_{\max}$, analyze the lower bound and present an online algorithm named GTR. Similarly, before analyzing the lower bound and algorithm, we first introduce two definitions (Zheng *et al.*).

Definition 2. *Strict waste time segment* and *strict available time segment:* For an idle time segment $[e_k, s_i)$, with r_i is a large request, and request r_k is scheduled before the request r_i. If $0 < s_i - e_k < 1/2$, the idle time segment $[e_k, s_i)$ is a strict waste time segment, denoted by T^*_w. Otherwise, $[e_k, s_i)$ is a strict available time segment, denoted by T^*_a.

The segment of each large request r_i may induce a forced idle time segment, i.e., an idle time segment $[e_k, s_i)$ (for some $1 \leq k < i$, request r_k is scheduled before the request r_i, no matter which berths and QCs assigned)only for during $[e_k, s_i)$, there exists idle QCs,. If $0 < s_i - e_k < 1/2$, a small request later on cannot be satisfied within the time segment, thus it is a waste time segment. Otherwise, it is an available time segment. Set T^*_a=[t_1,t_2) where t_1=e_k, t_2=s_i, and T^*_a=[0,0) initially.

4.1 Lower Bound

Theorem 3. *For problem* $hybr\,|online - over - list,\ LD = 1\,|BAP - QCAP|$ $C_{\max}$*, any online scheduling algorithm has a competitive ratio of at least* $5/4$*.*

Proof. To prove the theorem, we construct a request input sequence $\mathcal{I}$. $\mathcal{I}$ contains at least two requests. Let $C_{max}(\mathcal{I})$ and $C^*(\mathcal{I})$ be the makespan of schedule produced by $\mathcal{A}$ and an optimal offline algorithm *OPT*, respectively.

The first request $r_1 = 1$, and lookahead $r_2 = 1$. If A processes r_1 with two QCs on berth b_1, offline adversary releases request $r_3 = \Delta$ and $r_4 = \Delta$, $C_{\max}(\mathcal{I}) \geq 1/2 + \Delta/2$. While OPT processes the request r_1 with one QC on berth b_1 at time 0, r_2 with one QC on berth b_1 after completing r_1, and r_3 with four QCs on berth b_2, b_3 at time 0, r_4 with four QCs on berth b_2, b_3 after completing r_3. $C^*(\mathcal{I}) = \max\{2, \Delta/2\}$. Setting $\Delta = 4$, we have $C_{\max}(\mathcal{I})/C^*(\mathcal{I}) \geq 5/4$. If $\mathcal{A}$ processes r_1 with one QC on berth b_1, no more requests arrive. $C_{\max}(\mathcal{I}) \geq 1$. While OPT processes r_1 ,r_2 with two QCs on different berths at time 0. $C^*(\mathcal{I}) = 1/2$. $C_{\max}(\mathcal{I})/C^*(\mathcal{I}) \geq 2$. The theorem follows. □

4.2 Algorithm GTR

On the release of any request r_i $(i \geq 1)$, if $i = n$, then the scheduler can lookahead r_n is the last request of consequence $\mathcal{I}$, so we can optimal assign this request to minimize $C_{\max}$. Otherwise, $1 \leq i \leq n-1$, $\mathcal{A}$ behaves as follows.

Case 1. $r_i = \Delta, r_{i+1} = 1$. There are four subcases in the following:
- Case 1.1. $t_{i,1} = t_{i,2} = t_{i,3}$. If $\Delta \geq 3$, assign r_i to the two leftmost berths with four QCs. If $3 \geq \Delta \geq 2$, assign r_i to the rightmost two berths with three QCs. Set $s_i = t_{i,1}$.
- Case 1.2. $t_{i,1} = t_{i,2} < t_{i,3}$. If $C_{i,1} = t_{i,1}$, assign r_i to the leftmost two berths with four QCs. If $C_{i,1} = t_{i,3}$, assign r_i to the rightmost two berths with three QCs. Set $s_i = t_{i,3}$.
- Case 1.3. $t_{i,1} < t_{i,2} = t_{i,3}$. If $\Delta \geq 3$, assign r_i to the leftmost two berths with four QCs. If $3 \geq \Delta \geq 2$, when $C_{i,1} = t_{i,1}$, assign r_i to the rightmost two berths with three QCs. When $C_{i,1} = t_{i,3}$, assign r_i to the leftmost two berths with four QCs. Set $s_i = t_{i,3}$.
- Case 1.4. $t_{i,1} < t_{i,2} < t_{i,3}$. Assign r_i to the leftmost two berths with four QCs, set $s_i = t_{i,3}$.

case 2. $r_i = \Delta, r_{i+1} = \Delta$. If $t_{i,1} = t_{i,2} < t_{i,3}$, and $C_{i,1} = t_{i,3}$, assign r_i to the rightmost two berths with three QCs. Otherwise, assign r_i to the leftmost two berths with four QCs. Set $s_i = t_{i,3}$.

case 3. $r_i = 1, r_{i+1} = 1$. If $T^*_a = [t_1, t_2) \neq [0, 0)$, set $s_i = t_2$ and assign r_i to idle berth. Otherwise if $T^*_a = [0, 0)$, there are the following four subcases.
- Case 3.1. $t_{i,1} = t_{i,2} = t_{i,3}$. Assign r_i to the leftmost berth with two QCs, set $s_i = t_{i,1}$.
- Case 3.2. $t_{i,1} = t_{i,2} < t_{i,3}$. Assign r_i to the middle berth with two QCs, set $s_i = t_{i,1}$.
- Case 3.3. $t_{i,1} < t_{i,2} = t_{i,3}$. When $C_{i,1} = t_{i,3}$, if $t_{i,2} - t_{i,1} < 1/2$, assign r_i to the leftmost berth with two QCs, set $s_i = t_{i,3}$. If $t_{i,2} - t_{i,1} \geq 1/2$, assign r_i to the rightmost berth with one QC, set $s_i = t_{i,1}$. When $C_{i,1} = t_{i,1}$,assign r_i to the leftmost berth with two QCs, set $s_i = t_{i,1}$.

  - Case 3.4. $t_{i,1} < t_{i,2} < t_{i,3}$. Assign r_i to the middle berth with two QCs, set $s_i = t_{i,1}$.

case 4. $r_i = 1, r_{i+1} = \Delta$. If $T_a^* = [t_1, t_2) \neq [0, 0)$, set $s_i = t_2$ and assign r_i to the idle berth. Otherwise if $T_a^* = [0, 0)$, there are the following four subcases.

  - Case 4.1. $t_{i,1} = t_{i,2} = t_{i,3}$. If $\Delta \geq 3$, assign r_i to the rightmost berths with one QC. If $3 \geq \Delta \geq 2$, assign r_i to the leftmost berth with two QCs. Set $s_i = t_{i,1}$.
  - Case 4.2. $t_{i,1} = t_{i,2} < t_{i,3}$. When $\Delta \geq 3$, if $C_{i,1} = t_{i,3}$, $t_{i,2} - t_{i,1} = 1/2$ and $\Delta \geq 6$, assign r_i to the middle berth with two QCs. If $C_{i,1} = t_{i,1}$, assign r_i to the rightmost berth with one QC. Otherwise, assign r_i to the leftmost berth with two QCs. When $3 \geq \Delta \geq 2$, assign r_i to the middle berth with two QCs.
  - Case 4.3. $t_{i,1} < t_{i,2} = t_{i,3}$. If $\Delta \geq 3$, assign r_i to the rightmost berths with one QC. If $3 \geq \Delta \geq 2$, when $C_{i,1} = t_{i,1}$, assign r_i to the leftmost berth with two QCs. When $C_{i,1} = t_{i,3}$, assign r_i to the rightmost berth with one QC. Set $s_i = t_{i,1}$.
  - Case 4.4. $t_{i,1} < t_{i,2} < t_{i,3}$. Assign r_i to the middle berth with two QCs, set $s_i = t_{i,1}$.

Lemma 1. *When $\Delta \geq 3$, if there exists a T_w^* segment in schedule σ (the schedule produced by GTR), then the schedule σ should have the sequence $r_{i-2} = 1$, $r_{i-1} = 1$, $r_i = \Delta$, $r_j = \Delta$ $(j > i)$, with the condition $t_{i-2,1} = t_{i-2,2} = t_{i-2,3}$ and $\Delta \leq 6$. When $3 \geq \Delta \geq 2$, if there exists a T_w^* segment in schedule σ, then the schedule σ should have the sequence $r_{i-1} = 1$, $r_i = \Delta$ or $r_{i-1} = \Delta$, $r_i = 1$, $r_j = \Delta$ $(j > i)$, with the condition $t_{i-1,1} = t_{i-1,2} = t_{i-1,3}$.*

Proof. By the definition of T_w^* and the description of the algorithm, T_w^* exists because the assignment of at least two large requests, denoted by r_i, r_j $(j > i)$, and r_i is assigned to the rightmost berths with three QCs, r_j is assigned to the leftmost berths with four QCs. T_w^* is on b_1.

When $\Delta \geq 3$, by the description of the algorithm, only in case 1.2 $(C_{i,1} = t_{i,3})$ and 2.1, the large request will be assigned to the rightmost berths with three QCs. For the case 1.2 $(C_{i,1} = t_{i,3})$ and 2.1, $t_{i,1} = t_{i,2} < t_{i,3}$ and $C_{i,1} = t_{i,3}$, it occurs by $r_{i-2} = 1$ and $r_{i-1} = 1$, r_{i-2} assigned to the leftmost berth. For the case r_{i-2} assigned to berth b_1, should satisfy the condition that $t_{i-2,1} = t_{i-2,2} = t_{i-2,3}$ or $t_{i-2,1} < t_{i-2,2} = t_{i-2,3} = C_{i-2,1}$ with $t_{i-2,2} - t_{i-2,1} < 1/2$. For the case r_{i-1} assigned to berth b_1, should satisfy the condition that $C_{i-2,1} = t_{i-2,1} = t_{i-2,2} < t_{i-2,3}$, $t_{i-2,2} - t_{i-2,1} < 1/2$ and $\Delta \leq 6$. So we can find that if there exists a T_w^* segment, the schedule σ should have the sequence $r_{i-2} = 1$, $r_{i-1} = 1$, $r_i = \Delta$, $r_j = \Delta$ $(j > i)$, $t_{i-2,1} = t_{i-2,2} = t_{i-2,3}$ and $\Delta \leq 6$.

When $3 \geq \Delta \geq 2$, the same proof as $\Delta \geq 3$, we can find that if it exists a T_w^* segment, the schedule σ should have the sequence $r_{i-1} = 1$, $r_i = \Delta$ or $r_{i-1} = \Delta$, $r_i = 1$, $r_j = \Delta$ $(j > i)$ and $t_{i-1,1} = t_{i-1,2} = t_{i-1,3}$.

The lemma follows. □

Lemma 2. *There is at most one T_a^* segment in schedule σ, and it is by the same condition as T_w^*.*

Proof. By the definition of T^*_a, it is easy to find T^*_a segment formed by the same condition as T^*_w segment. But different with T^*_w segment, $0 < s_i - e_k < 1/2$. In a T^*_a segment, $s_i - e_k \geq 1/2$. That means, once the schedule σ has a T^*_a segment, the after small requests will be assigned in the idle berths, until $0 < s_i - e_k < 1/2$. So there is at most one T^*_a segment in schedule σ. The lemma follows. □

Theorem 4. *For problem hybr* $|online - over - list,\ LD = 1\ |BAP - QCAP|$ $C_{\max}$ *with 3 berths and 5 QCs, GTR is* $5/4$ *-competitive.*

Proof. Given any request input sequence $\mathcal{I} = \{r_1, r_2, ..., r_n\}$. Let $C_\sigma(\mathcal{I})$ and $C^*(\mathcal{I})$ be the makespan of a schedule produced by GTR and by an optimal offline algorithm *OPT*. Based on the interval of $\varDelta$, we consider two cases .

Case 1. $\varDelta \geq 3$, we consider four subcases by the existence of T^*_a, T^*_w.

Case 1.1. $T^*_w = \phi$, $T^*_a = [0, 0)$. We consider five subcases by the assignment of r_{n-1}, r_n.

Case 1.1.1. $t_{n-1,1} = t_{n-1,2} = t_{n-1,3}$. It is easy to prove $C_\sigma(\mathcal{I}) = C^*(\mathcal{I})$.

Case 1.1.2. $r_{n-1} = \varDelta$, $r_n = 1$. When r_{n-1} is assigned by algorithm cases 1.2($C_{i,1} = t_{i,3}$), 1.3, 1.4, for $T^*_a = [0, 0)$, $T^*_w = \phi$, thus the left four QCs on leftmost two berths are kept busy during $[0, C_\sigma(\mathcal{I}))$, we have $C_\sigma(\mathcal{I}) \leq t_{n-1,3} + \max\{1, \varDelta/4\}$, $C^*(\mathcal{I}) \geq t_{n-1,3} + (1+\varDelta)/5 - (t_{n-1,3} - t_{n-1,1})/5$, implying $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 5/4$. When r_{n-1} is assigned by algorithm case 1.2 ($C_{i,1} = t_{i,1}$), implying $\varDelta \leq 6$, $t_{n-1,3} - t_{n-1,1} = 1$ and $t_{n-1,3} \geq 1$, thus before assign r_{n-1}, $T^*_w = \phi$, and maybe exist one T^*_a segment with $t_2 - t_1 = 1/2$ or not. If before assign r_{n-1}, $T^*_a = [0, 0)$, then $C_\sigma(\mathcal{I}) = \max\{t_{n-1,1} + \varDelta/3, t_{n-1,3} + 1/2\}$, $C^*(\mathcal{I}) \geq t_{n-1,1} + (1+\varDelta)/5 + (t_{n-1,3} - t_{n-1,1})/5$, implying $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 5/4$. If before assign r_{n-1}, $T^*_a = [t_1, t_2)$ with $t_2 - t_1 = 1/2$, thus $C_\sigma(\mathcal{I}) = \max\{t_{n-1,1} + \varDelta/3, t_{n-1,3}\}$, $C^*(\mathcal{I}) \geq t_{n-1,1} + \varDelta/5 + (t_{n-1,3} - t_{n-1,1})/5$. Since there exist a T^*_a segment, then $t_{n-1,1} \geq \varDelta/3$, we get $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 20/17$.

Case 1.1.3. $r_{n-1} = \varDelta$, $r_n = \varDelta$. For $T^*_w = \phi$, $T^*_a = [0, 0)$ and thus the left QCs on leftmost two berths are kept busy during $[0, C_\sigma(\mathcal{I}))$, we have $C^*(\mathcal{I}) \geq 4C_\sigma(\mathcal{I})/5$.

Case 1.1.4. $r_{n-1} = 1$, $r_n = 1$. Before assign r_{n-1}, $T^*_w = \phi$, but there maybe exist a T^*_a segment with $t_2 - t_1 = 1$ or $t_2 - t_1 = 1/2$. If before assign r_{n-1}, $T^*_a = [0, 0)$, then we have $C_\sigma(I) \leq t_{n-1,3} + 1/2$ and $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 \geq 4t_{n-1,3}/5 + 2/5$, so $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 5/4$. If before assign r_{n-1}, $T^*_a = [t_1, t_2)$ and $t_2 - t_1 = 1/2$, then the last completed request before r_{n-1} is a large request. Notice that there is a waste of QC utility equal to 1 in T^*_a, thus $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 - 1/5$. By the case condition, meaning $\varDelta = 3(k+1)/2 \geq 9/2$, $C_\sigma(\mathcal{I}) = t_{n-1,3}$, then $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) < 5/4$. If before assign r_{n-1}, $T^*_a = [t_1, t_2)$ and $t_2 - t_1 = 1$. Notice that there is a waste of QC utility equal to 2 in T^*_a, so $C_\sigma(\mathcal{I}) = t_{n-1,3}$ and $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 - 2/5$, then $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 5/4$.

Case 1.1.5. $r_{n-1} = 1$, $r_n = \varDelta$. For $T^*_w = \phi$, $T^*_a = [0, 0)$ and thus the left four QCs on leftmost two berths are kept busy during $[0, C_\sigma(\mathcal{I}))$, we have $C^*(\mathcal{I}) \geq 4C_\sigma(\mathcal{I})/5$.

Case 1.2. $T^*_a = [0,0)$ and $T^*_w = \phi$. Assume there are p T^*_w segments, for each $T^*_w = [t^*_1, t^*_2)$, $t^*_2 - t^*_1 = 1/2$, then $t_{n,1} \geq pt_2$. we have $C^*(\mathcal{I}) \geq C_\sigma(\mathcal{I}) - (C_\sigma(\mathcal{I}) - t_{n,1})/5 - p(t_2 - t_1)/5 \geq 4C_\sigma(\mathcal{I})/5$.

Case 1.3. $T^*_a = [t_1, t_2) \neq [0,0)$ and $T^*_w = \phi$. By the case condition, $t_{n,1} \geq t_2$, $C^*(\mathcal{I}) \geq C_\sigma(\mathcal{I}) - (C_\sigma(\mathcal{I}) - t_{n,1})/5 - (t_2 - t_1)/5 \geq 4C_\sigma(\mathcal{I})/5$.

Case 1.4. $T^*_a = [t_1, t_2) \neq [0,0)$ and $T^*_w = [t^*_1, t^*_2) \neq \phi$. By the lemma condition, implying $t_2 - t_1 = \Delta/3 - k/2 > 1/2$, $t^*_2 - t^*_1 = \Delta/3 - k^*/2 < 1/2$ and $t_{n,1} \geq t_2$. Assume there are p T^*_w segment, then $t_1 \geq p$. $C^*(\mathcal{I}) \geq C_\sigma(\mathcal{I}) - (C_\sigma(\mathcal{I}) - t_{n,1})/5 - (t_2 - t_1)/5 - p(t^*_2 - t^*_1)/5 \geq 4C_\sigma(\mathcal{I})/5$.

Case 2. $3 > \Delta \geq 2$, we consider four cases by the existence of T^*_a, T^*_w .

Case 2.1. $T^*_w = \phi$, $T^*_a = [0,0)$, we consider four subcases by the assignment of r_{n-1}, r_n.

Case 2.1.1. $r_{n-1} = \Delta$, $r_n = 1$. There are two situations in this case, before assign r_{n-1}, $T^*_a = [0,0)$ or $T^*_a = [t_1, t_2)$, $t_2 - t_1 = 1/2$. If before assign r_{n-1}, $T^*_a = [0,0)$, then $C_\sigma(\mathcal{I}) \leq t_{n-1,3} + \Delta/3$, $C^*(\mathcal{I}) \geq t_{n-1,3} + (1+\Delta)/5 - (t_{n-1,3} - t_{n-1,1})/5 \geq 4t_{n-1,3}/5 + (1+\Delta)/5$, thus $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) < 5/4$. If before assign r_{n-1}, $T^*_a = [t_1, t_2)$, $t_2 - t_1 = 1/2$. By the case condition, the last completed request in σ before r_{n-1}. So $C_\sigma(\mathcal{I}) \leq t_{n-1,3} + \Delta/4$, $C^*(\mathcal{I}) \geq t_{n-1,3} + (1+\Delta)/5 - (t_{n-1,3} - t_{n-1,1})/5 - 1/5 \geq 4t_{n-1,3}/5 + \Delta/5$. Thus $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) < 5/4$.

Case 2.1.2. $r_{n-1} = \Delta$, $r_n = \Delta$. For $T^*_a = [0,0)$, $T^*_w = \phi$ and thus the left QCs on leftmost two berths are kept busy during $[0, C_\sigma(\mathcal{I}))$, we have $C^*(\mathcal{I}) \geq 4C_\sigma(\mathcal{I})/5$.

Case 2.1.3. $r_{n-1} = 1$, $r_n = 1$. Before assign r_{n-1}, $T^*_w = \phi$, and there may exist a T^*_a segment with $t_2 - t_1 = 1$ or $t_2 - t_1 = 1/2$. If before assign r_{n-1}, $T^*_a = [0,0)$, then we have $C_\sigma(\mathcal{I}) \leq t_{n-1,3} + 1/2$, $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 \geq 4t_{n-1,3}/5 + 2/5 = 4C_\sigma(\mathcal{I})/5$. If before assign r_{n-1}, $T^*_a = [t_1, t_2)$ and $t_2 - t_1 = 1/2$, then the last completed request before r_{n-1} is a large request. Notice that there is a waste of QC utility equal to 1 in T^*_a, thus $C_\sigma(\mathcal{I}) = \max\{t_{n-1,1} + 1, t_{n-1,3}\}$, $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 - 1/5$. When $t_{n-1,3} \leq t_{n-1,1} + 1$, by the lemma condition, implying $t_{n-1,3} - t_{n-1,1} = \Delta/4$, $t_{n-1,1} \geq k\Delta/3$, $k(\Delta/3 - 1/2) = p/2$ and $p \geq 1$. We can get $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 8/7$. When $t_{n-1,3} > t_{n-1,1} + 1$, $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 - 1/5 \geq 4C_\sigma(\mathcal{I})/5$. If before assign r_{n-1}, $T^*_a = [t_1, t_2)$ and $t_2 - t_1 = 1$. Notice that there is a waste of QC utility equal to 2 in T_a, so $C_\sigma(\mathcal{I}) = t_{n-1,3}$ and $C^*(\mathcal{I}) \geq t_{n-1,3} + 2/5 - (t_{n-1,3} - t_{n-1,1})/5 - 1/5 \geq 4C_\sigma(\mathcal{I})/5$.

Case 2.1.4. $r_{n-1} = 1$, $r_n = \Delta$. For $T^*_a = [0,0)$, $T^*_w = \phi$ and thus the left QCs on leftmost two berths are kept busy during $[0, C_\sigma(\mathcal{I}))$, we have $C^*(\mathcal{I}) \geq 4C_\sigma(\mathcal{I})/5$.

Cases 2.2, 2.3, 2.4 are the same as cases 1.2, 1.3, 1.4 respectively.

The theorem follows. □

5 Conclusion

This paper considers an online integrated allocation of berths and quay cranes in container terminal with 1-lookahead. We focus on the hybrid layout with three berths and four, five cranes, and present an online deterministic algorithm respectively, which are proved to be optimal in competitiveness. In the future, we will focus on the case of six cranes, and consider k-lookahead case.

Acknowledgments. The authors would like to acknowledge the financial support of Grants (No.61221063.) from NSF of China and (No.IRT1173) from PCSIRT of China.

Appendix

A: Lower bound of the case with four QCs

we show the complete calculation process as follow.

If A processes r_1 with two QCs on berth b_1, we get:

$$C_{\max}(\mathcal{I}) \geq 1/2 + \Delta/4 \tag{1}$$
$$\Delta \geq 6,\ C^*(\mathcal{I}) = 1/2 + \Delta/4 \tag{2}$$
$$6 \geq \Delta \geq 3,\ C^*(\mathcal{I}) = \Delta/3 \tag{3}$$
$$3 \geq \Delta \geq 2,\ C^*(\mathcal{I}) = 1 \tag{4}$$

If $\mathcal{A}$ processes r_1 with three QCs on berth b_1, b_2, we get:

$$\Delta \geq 6,\ C_{\max}(\mathcal{I}) \geq \Delta/3 \tag{5}$$
$$6 \geq \Delta \geq 9/2,\ C_{\max}(\mathcal{I}) \geq 2 \tag{6}$$
$$9/2 \geq \Delta \geq 2,\ C_{\max}(\mathcal{I}) \geq 1/2 + \Delta/3 \tag{7}$$
$$C^*(\mathcal{I}) = 1/2 + \Delta/4 \tag{8}$$

If $\mathcal{A}$ processes r_1 with two QCs on berth b_1, b_2, we get:

$$C_{\max}(\mathcal{I}) \geq \Delta/2 \tag{9}$$
$$\Delta \geq 6,\ C^*(\mathcal{I}) = 1/2 + \Delta/4 \tag{10}$$
$$6 \geq \Delta \geq 3,\ C^*(\mathcal{I}) = \Delta/3 \tag{11}$$
$$3 \geq \Delta \geq 2,\ C^*(\mathcal{I}) = 1 \tag{12}$$

After processed Eqs.(1-4),we have:

$$\Delta \geq 6,\ \rho \geq 1 \tag{13}$$
$$6 \geq \Delta \geq 3,\ \rho \geq \max\left\{\frac{6+3\Delta}{4\Delta}\right\} \tag{14}$$
$$3 \geq \Delta \geq 2,\ \rho \geq \max\left\{\frac{2+\Delta}{4}\right\} \tag{15}$$

After processed Eqs.(5-8),we have:

$$\Delta \geq 6,\ \rho \geq \max\left\{\frac{4\Delta}{6+3\Delta}\right\} \geq 1 \tag{16}$$

$$6 \geq \Delta \geq 9/2,\ \rho \geq \max\left\{\frac{8}{2+\Delta}\right\} \tag{17}$$

$$9/2 \geq \Delta \geq 2,\ \rho \geq \max\left\{\frac{6+4\Delta}{6+3\Delta}\right\} \tag{18}$$

After processed Eqs.(9-12),we have:

$$\Delta \geq 6,\ \rho \geq \max\left\{\frac{2\Delta}{2+\Delta}\right\} \geq 1 \tag{19}$$

$$6 \geq \Delta \geq 3,\ \rho \geq \frac{3}{2} \tag{20}$$

$$3 \geq \Delta \geq 2,\ \rho \geq \frac{\Delta}{2} \tag{21}$$

From Eqs.(13-21),we get:

$$\Delta \geq 6,\ \rho \geq 1 \tag{22}$$

$$6 > \Delta \geq \frac{9}{2},\ \rho \geq \max\left\{\frac{3\Delta+6}{4\Delta}\right\},\ \rho \geq \frac{13}{12} \tag{23}$$

$$\frac{9}{2} > \Delta \geq \frac{6+12\sqrt{2}}{7},\ \rho \geq \max\left\{\frac{3\Delta+6}{4\Delta}\right\},\ \rho \geq \frac{1+\sqrt{2}}{2} \tag{24}$$

$$\frac{6+12\sqrt{2}}{7} > \Delta \geq 3,\ \rho \geq \max\left\{\frac{4\Delta+6}{3\Delta+6}\right\},\ \rho \geq \frac{1+\sqrt{2}}{2} \tag{25}$$

$$3 > \Delta \geq \frac{2+2\sqrt{10}}{3},\ \rho \geq \max\left\{\frac{4\Delta+6}{3\Delta+6}\right\},\ \rho \geq \frac{6}{5} \tag{26}$$

$$\frac{2+2\sqrt{10}}{3} > \Delta \geq 2,\ \rho \geq \frac{4+\sqrt{10}}{6} \tag{27}$$

Setting $\Delta = (6+12\sqrt{2})/7$, then $\rho \geq (1+\sqrt{2})/2$.

B: Theorem 2

As in case 2, namely, $2 \leq \Delta \leq (2+2\sqrt{10})/3$ or $\Delta \geq (6+12\sqrt{2})/7$, the algorithm is the same as that in Zheng *et al.*, we can refer to their proof. For the sake of completeness, we present their proof here:

Proof. Given any request input sequence $\mathcal{I} = \{r_1, r_2, ..., r_n\}$. let σ be the schedule produced by LIST. Let $C_\sigma(\mathcal{I})$ be the makespan of σ, and $C^*(\mathcal{I})$ that of a schedule produced by an optimal onine algorithm OPT. We consider two cases below.

Case 1. The last request is a large one, i.e., $r_n = \Delta$. For the case with $T^*_a = [0, 0)$, we have $C^*(\mathcal{I}) = C_\sigma(\mathcal{I}) = C_{n,1} + \Delta/4$ since neither LIST nor OPT makes any waste of QC utility within $[0, C_\sigma(\mathcal{I}))$.

For the other case with $T^*_a = [t_1, t_2) \neq [0, 0)$, let $k \geq 1$ be the number of large requests after time t_2. $C_\sigma(\mathcal{I}) = t_2 + k\Delta/4$. If $t_2 = 1/2$ and $k = 1$, implying $n = 2$, $r_1 = 1$ and $r_2 = \Delta$, then $C^*(\mathcal{I}) \geq \min\{1/2 + \Delta/4, \max\{1, \Delta/3\}\}$ and thus $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 5/4$; otherwise if either $t_2 \geq 1$ or $k \geq 2$, $C^*(\mathcal{I}) \geq t_2 + k\Delta/4 - 1/4$ and $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 6/5$.

Case 2. The last request is a small request, i.e., $r_n = 1$. We claim that $T^*_a = [0, 0)$ in σ since otherwise r_n would have been processed in the T^*_a. Divide this case into two subcases by whether $C_{n,1} = C_{n,2}$.

Case 2.1. $C_{n,1} = C_{n,2}$, In this case $C_\sigma(\mathcal{I}) = C_{n,1} + 1/2$. If either $n = 1$ or $n = 3$ with $r_1 = r_2 = r_3 = 1$, we have $C_{n,1} \leq 1/2$ and $C_\sigma(\mathcal{I}) = C^*(\mathcal{I})$; if $n = 2$ and $r_1 = \Delta$, then $C_{n,1} = \Delta/4$ and $C^*(\mathcal{I}) \geq \min\{1/2 + \Delta/4, \max\{1, \Delta/3\}\}$, implying $C_\sigma(\mathcal{I})/C^*(\mathcal{I}) \leq 5/4$; otherwise if $C_{n,1} \geq 1$, then $C^*(\mathcal{I}) \geq C_{n,1} + 1/4$, implying a ratio of at most 6/5.

Case 2.2. $C_{n,1} < C_{n,2}$ (or $C_{n,2} < C_{n,1}$). Then we have by previous analysis that $C_{n,2} = C_{n,1} + 1/2$ (or $C_{n,1} = C_{n,2} + 1/2$), and thus $C_\sigma(\mathcal{I}) = C^*(\mathcal{I})$.

The theorem follows. □

References

1. Bierwirth, C., Meisel, F.: A survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research **202**(3), 615–627 (2010)
2. Blazewicz, J., Cheng, T.C.E., Machowiak, M., Oguz, C.: Berth and quay crane allocation: a moldable task scheduling model. Journal of The Operational Research Society **62**, 1189–1197 (2011)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, New York (1998)
4. Carlo, H., Vis, I., Roodbergen, K.: Seaside operations in container terminals: literature overview, trends, and research directions. Flexible Services and Manufacturing Journal, 1–39 (2013)
5. Chen, J.H., Lee, D.H., Cao, J.X.: A combinatorial benders cuts algorithm for the quayside operation problem at container terminals. Transportation Research Part E: Logistics and Transportation Review **48**(1), 266–275 (2012), select Papers from the 19th International Symposium on Transportation and Traffic Theory
6. Giallombardo, G., Moccia, L., Salani, M., Vacca, I.: Modeling and solving the tactical berth allocation problem. Transportation Research Part B: Methodological **44**(2), 232–245 (2010)
7. Imai, A., Sun, X., Nishimura, E., Papadimitriou, S.: Berth allocation in a container port: using a continuous location space approach. Transportation Research Part B: Methodological **39**(3), 199–221 (2005)
8. Liang, C., Guo, J., Yang, Y.: Multi-objective hybrid genetic algorithm for quay crane dynamic assignment in berth allocation planning. Journal of Intelligent Manufacturing **22**(3), 471–479 (2011)

9. Lokuge, P., Alahakoon, D.: Improving the adaptability in automated vessel scheduling in container ports using intelligent software agents. European Journal of Operational Research **177**(3), 1985–2015 (2007)
10. Mandelbaum, M., Shabtay, D.: Scheduling unit length jobs on parallel machines with lookahead information. Journal of Scheduling **14**(4), 335–350 (2011)
11. Park, Y.M., Kim, K.H.: A scheduling method for berth and quay cranes. OR Spectrum **25**(1), 1–23 (2003)
12. Zhang, L., Khammuang, K., Wirth, A.: On-line scheduling with non-crossing constraints. Operations Research Letters **36**(5), 579–583 (2008)
13. Zhen, L., Chew, E.P., Lee, L.H.: An integrated model for berth template and yard template planning in transshipment hubs. Transportation Science **45**(4), 483–504 (2011)
14. Zheng, F., Cheng, Y., Liu, M., Xu, Y.: Online interval scheduling on a single machine with finite lookahead. Computers & Operations Research **40**(1), 180–191 (2013)
15. Zheng, F., Qiao, L., Liu, M., Chu, C.: Online integrated allocation for small numbers of berths and quay cranes in container terminals (working paper)

Disjoint Path Allocation with Sublinear Advice

Heidi Gebauer[1], Dennis Komm[2], Rastislav Královič[3], Richard Královič[4], and Jasmin Smula[2](✉)

[1] Institute of Applied Mathematics and Physics, Zurich University of Applied Sciences, Winterthur, Switzerland
geba@zhaw.ch
[2] Department of Computer Science, ETH Zürich, Zürich, Switzerland
{dennis.komm,jasmin.smula}@inf.ethz.c
[3] Department of Computer Science, Comenius University, Bratislava, Slovakia
kralovic@dcs.fmph.uniba.sk
[4] Google Inc., Zürich, Switzerland

Abstract. We study the disjoint path allocation problem. In this setting, a path P of length L is given, and a sequence of subpaths of P arrives online, one in every time step. Each such path requests a permanent connection between its two end-vertices. An online algorithm can admit or reject such a request; in the former case, none of the involved edges can be part of any other connection. We investigate how much additional binary information (called "advice") can help to obtain a good solution. It is known that, with roughly $\log_2 \log_2 L$ advice bits, it can be guaranteed that a $\log_2 L$-competitive solution is computed. In this paper, we prove the surprising result that, with $L^{1-\varepsilon}$ advice bits, it is not possible to obtain a solution with a competitive ratio better than $(\delta \log_2 L)/2$, where $0 < \delta < \varepsilon < 1$. This shows an interesting threshold behavior of the problem. A fairly good competitive ratio, namely $\log_2 L$, can be obtained with very few advice bits. However, any increase of the advice does not help any further until an almost linear number of advice bits is supplied. Then again, it is also known that linear advice allows for optimality.

1 Introduction

The input of an *online problem* arrives piecewise as a sequence of n requests $x_1, \ldots, x_n$ in consecutive time steps. An *online algorithm* Alg computes a sequence of answers $y_1, \ldots, y_n$, where each answer y_i, $1 \leq i \leq n$, must be given in the ith time step while only depending on the requests $x_1, \ldots, x_i$ that are known up to this point. If the given online problem is a maximization problem, we assess the solution quality of Alg by comparing the *gain* of its solution to the one hypothetically reachable if the whole input sequence were known in advance; this is modeled by an offline algorithm Opt that has this knowledge. More formally, Alg is called *c-competitive* if there is a non-negative constant (with respect to the input length) α such that, on any instance $I = (x_1, \ldots, x_n)$ of the given

Partially funded by SNF grant 200021-141089 and VEGA grant 1/0979/12.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 417–429, 2015.
DOI: 10.1007/978-3-319-21398-9_33

online maximization problem, we have gain(Opt(I)) $\leq c \cdot$ gain(Alg(I)) $+ \alpha$; the smallest c for which this holds is called the *competitive ratio* of Alg. If the above inequality even holds with $\alpha = 0$, we call Alg *strictly c-competitive.* This framework, called *competitive analysis*, has been around for three decades now [17] and has become the standard tool to investigate the performance of online algorithms. For a detailed introduction to online algorithms and competitive analysis, we refer the reader to the literature [6]. Similar to the approximation ratio that measures what is lost when computing a solution to a hard offline problem in polynomial time, the competitive ratio tells us what is lost due to incomplete knowledge of the input at hand. However, since a complete absence of knowledge about the input is unrealistic in many real-world environments, it is reasonable to ask to what extent some certain additional information about the yet unrevealed requests can be exploited. While there are many models such as *semi-online* problems [10] where some specific parameters of the input are known, the *advice complexity* of an online problem tries a more general approach. Here, we augment an online algorithm with an *advice tape* that may contain any binary information about the input. We may think of the advice as being prepared by an oracle that sees the whole input in advance. An online algorithm Alg with such an additional resource is called an *online algorithm with advice.* Alg is called *c-competitive with advice complexity b* if, for every input I of the given problem, there is some advice string ϕ such that Alg has a competitive ratio of at most c while never accessing more than the first b bits of ϕ. Note that we assume that the advice string is infinitely long [4,13]. This way, Alg cannot determine itself when the advice "ends," which may carry some additional information. Consequently, online algorithms with advice generalize many other approaches that assume additional information. Here, lower bounds are of particular interest, i. e., statements of the sort that some specific competitive ratio can never be reached with some given amount of additional information, no matter what this information actually is.

In this paper, we continue the study of the *disjoint path allocation problem on paths*, DPA for short. Here, we are given a path P of length L, i. e., with $L + 1$ vertices. A request is equal to a non-empty subpath of L. Alg must answer any such request by either admitting or rejecting it; this decision is final. If the request is admitted, a permanent connection between the two end-vertices of the subpath is established. After that, all involved edges are busy and cannot be part of any other connection. Therefore, a feasible solution corresponds to a set of *edge-disjoint paths*, namely the admitted subpaths of L. We call a request *blocked* if it cannot be admitted as the consequence of an earlier admission; an example is shown in Fig. 1. Note that, if L is known in advance (as a parameter of the problem), we can easily set the constant α of the definition of the competitive ratio to $L - 1$, which implies that every online algorithm that admits at least one request (e. g., a simple greedy algorithm) is indeed 1-competitive; this is, of course, undesirable for a serious analysis. In this paper, we therefore assume that L is given with the first request, and is thus a part of the input.

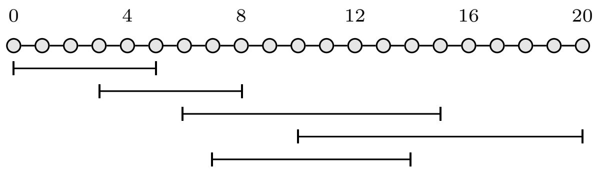


Fig. 1. An example of a DPA instance for $L = 20$. The requests arrive from top to bottom. If, e. g., an online algorithm admits the first one in time step 1, it cannot admit the second one. It is easy to see that no algorithm can admit more than 2 requests.

The advice complexity is usually a function b of the input length n. However, since the competitive ratio of algorithms for DPA is commonly a function of the graph size and we consider paths of length L only, we define b as a function of L.

We now discuss known results, describe our contribution, and put it into context. Due to space constraints, some of the technical details are omitted.

1.1 Related Work

A first model of online computation with advice was given by Dobrev et al. [9]. The model used in this paper was introduced by Hromkovič et al. [13] and first applied by Böckenhauer et al. [4]. There is an alternative model of computing with advice introduced by Emek et al. [11]. In this setting, the advice is not read from a tape, but it is supplied in every time step, and the number of advice bits is the same in every time step. Both models have so far been used to study a large number of problems, including the paging problem [4,14], the k-server problem [3,11,12,16], or metrical task systems [11]. One of the first online problems studied in the model of computing with advice as we use it in this paper was the DPA problem [4]. However, the authors of [4] mainly studied both the advice complexity and the reachable competitive ratio with respect to the input length n. The authors gave a lower bound of roughly $n/(2c)$ advice bits to obtain a strictly c-competitive solution. In this paper, we use the length L of the underlying path as a measurement, which is more consistent with the classical work [6]. With respect to L, the bound presented by Böckenhauer et al. [4] translates to roughly $(\log_2 L)/c$, which is improved exponentially by our main result. Böckenhauer et al. [5] noted that there is a $\log_2 L$-competitive online algorithm with advice that uses $\lceil \log_2 \lceil \log_2 L \rceil \rceil$ advice bits; this is a direct consequence from the "classify-and-randomly-select" algorithm from Awerbuch et al. [1]. Barhum et al. [2] generalized this technique and combined it with advice yielding online algorithms with advice that use a small amount of advice bits and obtain a solution of high quality. Moreover, they showed that $L - 1$ advice bits are both sufficient and necessary to be optimal. So far, no lower bound on the competitive ratio (except for optimality) that is achievable when reading $\omega(\log_2 L)$ advice bits is known.

1.2 Outline, Techniques, and Results

The remainder of this paper is devoted to giving non-trivial lower bounds on the number of advice bits necessary to obtain a certain competitive ratio. The result implies two interesting bounds.

1. For any c smaller than $1/2 \cdot (\log_2 L)/(\log_2 \log_2 L)^{1/4}$, any c-competitive online algorithm with advice needs to read at least $\Omega\big(L/(4^c c^4)\big)$ advice bits. Note that this bound is more general than the one presented by Barhum et al. [2], which only gives a statement for $b \leq \log_2 \log_2 (L/2)$.
2. For any δ, $0 < \delta < 1$, any $(\delta/2 \cdot \log_2 L)$-competitive online algorithm with advice needs to read at least $\omega(L^{1-\varepsilon})$ advice bits, for any constant ε with $\delta < \varepsilon < 1$. This complements the upper bound of $\log_2 L$ using $\lceil \log_2 \lceil \log_2 L \rceil \rceil$ advice bits [4]. The result is particularly surprising as it shows that we need almost double exponentially more advice to be $(\delta/2 \cdot \log_2 L)$-competitive instead of $\log_2 L$-competitive. Indeed, the number of advice bits necessary is almost linear. Then again, with a linear number of advice bits, it is possible to compute an optimal solution [2].

We prove the result by constructing a set $\mathcal{I}$ of instances such that any deterministic online algorithm can achieve the competitive ratio c only on a small fraction of the instances from $\mathcal{I}$. The number of these instances is bounded by some probabilistic arguments. All these instances can be organized as the leaves of a tree, such that paths from the root to some inner vertex v correspond to the instances that are leaves of the subtree rooted at v. Then we show that any online algorithm with advice needs to read many advice bits to achieve a competitive ratio of at most c on all instances from $\mathcal{I}$.

2 The Main Result

We start with some technical preliminaries that we need for the analysis of the given online algorithm with advice. For our calculations, we need Bernoulli's inequality, which states the following [7].

Fact 1. For every $x \in \mathbb{R}^{\geq -1}$ and every $n \in \mathbb{N}^{\geq 0}$, we have $(1+x)^n \geq 1 + nx$. $\square$

The following argumentation involves a random variable with hypergeometric distribution. Therefore, we now establish a result that follows from a well-known bound for the tail of the hypergeometric distribution. First, let us recall that a random variable with hypergeometric distribution with parameters M, N, and n counts the number of black balls drawn from an urn containing N balls, out of which exactly M are black, when drawing n balls uniformly at random without replacement (see, e. g., [15]). The following bound was established by Chvátal [8].

Fact 2. Consider a discrete random variable X with hypergeometric distribution with parameters M, N, and n, i. e.,

$$\Pr(X = i) = \binom{M}{i}\binom{N-M}{n-i}\bigg/\binom{N}{n} .$$

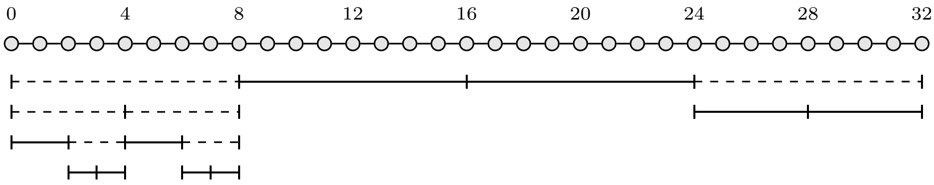


Fig. 2. An example of an instance from the constructed instance set $\mathcal{I}$ for a path of length $L = 32$ and $h = 3$. There are four phases with $L/2^h = 4$ requests each. Open requests are depicted by dashed lines. All other requests are closed (and therefore belong to the optimal solution).

Then, with $\mathrm{e} = 2.71828\ldots$ being Euler's number, we have

$$\mathbb{E}(X) = n \cdot M/N \quad \text{and} \quad P(X \leq \mathbb{E}(X) - tn) \leq \mathrm{e}^{-2t^2 n}, \quad \text{for any } t \geq 0\,. \qquad \square$$

We have to adapt this result slightly for our purposes.

Corollary 1. *Let X be a discrete random variable with hypergeometric distribution with parameters M, N, and n, and let $t \geq 0$. Then, for every $M' \leq M$, we have*

$$\Pr\left(X \leq n \cdot \frac{M'}{N} - tn\right) \leq \mathrm{e}^{-2t^2 n}\,.$$

Now let us describe how to construct the set $\mathcal{I}$ of instances for DPA. For the sake of simplicity, let L be a power of 2. Furthermore, let $h := h(L)$ be a parameter depending on L with

$$h \in \mathbb{N}^{\geq 1} \quad \text{and} \quad h \leq \log_2 L - 1\,. \tag{1}$$

Then the requests are presented to the algorithm in $h + 1$ phases. In each phase i, with $1 \leq i \leq h+1$, the algorithm is given $L/2^h$ edge-disjoint requests of length 2^{h-i+1}. Hence, in the first phase, $L/2^h$ edge-disjoint subpaths of length 2^h are presented, whose concatenation forms the complete path P. Half of the requests from phase i, with $1 \leq i \leq h$, are so-called *closed* requests, for which no intersecting requests will be presented anymore, and which hence belong to the optimal solution computed by an optimal algorithm Opt. The other half of these requests are *open*, i. e., they are split into two edge-disjoint requests of length 2^{h-i} each, which are then presented in phase $i+1$. Finally, in phase $h+1$, the algorithm is given $L/2^h$ subpaths of length 1 each, which all belong to the optimal solution; for an example, see Figs. 2 and 3.

Observation 1. The optimal solution on any instance I from $\mathcal{I}$ has a gain of

$$\text{gain}(\textsc{Opt}(I)) = \frac{(h+2)L}{2^{h+1}}\,.$$

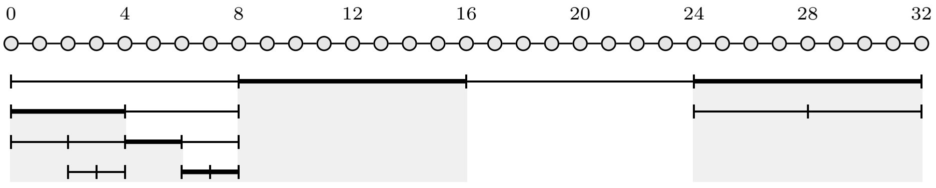


Fig. 3. When given the instance from Fig. 2 as an input, a deterministic algorithm ALG might admit the requests as depicted in this picture. Admitted requests are marked by thick lines. By admitting a request, all requests that intersect with this request become blocked, which is depicted by areas shaded in gray.

Proof. For every phase i with $1 \leq i \leq h$, there are $L/2^{h+1}$ requests that belong to the optimal solution, and additionally, there are $L/2^h$ ones from phase $h+1$, which yields

$$h \cdot \frac{L}{2^{h+1}} + \frac{L}{2^h} = \frac{h\,L}{2^{h+1}} + \frac{2\,L}{2^{h+1}} = \frac{(h+2)L}{2^{h+1}} \,. \qquad \square$$

Let us introduce another parameter $f := f(c)$, such that $f > 0$. Both parameters f and h must be chosen according to the competitive ratio that an algorithm is supposed to achieve. In the remainder of this chapter, we prove the following general theorem.

Theorem 1. *Any online algorithm for DPA with a competitive ratio of*

$$c := \frac{h+2}{2 \cdot \left(1 + \frac{h}{f}\right)}$$

needs to read at least $b := L/(2^h\, f^2) \cdot \log_2 \mathrm{e} - \log_2 h$ *advice bits.*

After that, we choose concrete values for f and h to obtain more tangible lower bounds, which are formulated as corollaries at the end of this section.

We start our argumentation by making the following observation. The set $\mathcal{I}$ of instances can naturally be represented by a $\binom{L/2^h}{L/2^{h+1}}$-ary tree of depth h (i. e., with $h+1$ levels), as depicted in Fig. 4. The root is on level 0 and corresponds to all instances from $\mathcal{I}$. There are $\binom{L/2^h}{L/2^{h+1}}$ instances on level 1, each of them representing all instances with the same particular set of open requests from phase 1. Any vertex on level i represents the set of all instances with the same particular sets of open requests from phases $1, \ldots, i$. Hence, for all instances that are represented by the same vertex on level i, the requests presented in the first $i+1$ phases are exactly the same. Every leaf is located on level h and corresponds to a single instance from $\mathcal{I}$.

Now consider some vertex v on level i with $0 \leq i \leq h$ and some arbitrary instance I_v represented by v. Then, any deterministic algorithm ALG, given I_v as its input, is always in the same state at the beginning of phase $i+1$, independently

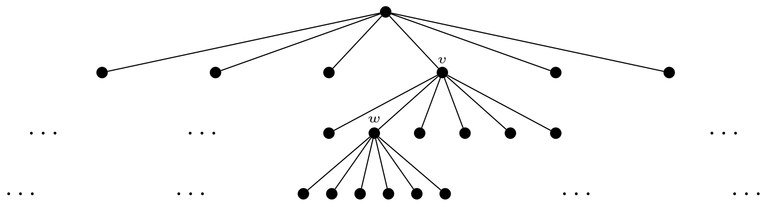


Fig. 4. An example of an instance tree. Figs. 2 and 3 are both placed in a scenario with a path of length 32 and $h+1=4$ phases (hence, with $L/2^h=4$ requests presented per phase). In this scenario, there are 6 possibilities to choose $L/2^{h+1}=2$ out of the 4 requests to be open in every phase. Hence, every inner vertex of the corresponding instance tree has exactly 6 children (most of which are only indicated by dots due to space restrictions). The root represents all instances, the vertex v represents all instances in which the same set of requests from phase 1 are open, and each leaf on level h represents all instances in which the same set of requests from phases $1,\dots,h$ are open, hence, each leaf represents a single instance.

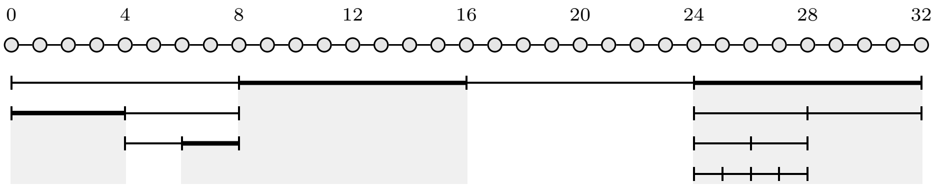


Fig. 5. In the instance tree from Fig. 4, on every level there must be one vertex that contains the instance from Fig. 3. Without loss of generality, on level 1, let this vertex be v. Then, v also contains the instance depicted in this figure. In both instances, the same requests from phase 1 are open, and thus, the requests presented in the first two phases are the same for *all* instances represented by v. Hence, at the beginning of and also throughout phase 2, each fixed deterministic algorithm Alg must be in the same state given any instance corresponding to v as its input, having seen and admitted the exact same requests so far.

of the instance that it gets as its input, i. e., it has seen and admitted the same requests so far on every instance represented by v; see Fig. 5.

From now on, let Alg be an arbitrary, but fixed deterministic algorithm for DPA. For a given vertex v on level i, let $\gamma^{(i)}$ be the gain of Alg on any instance represented by v during phase i, hence, the number of admitted requests during this phase. Moreover, let $\hat{\gamma}^{(i)}$ be the gain of Alg during all phases up to and including phase i, hence, $\hat{\gamma}^{(i)} := \sum_{j=1}^{i}\gamma^{(j)}$.

Let us introduce the following notion of bad phases and vertices. We call a phase i a *bad* phase for Alg if, at the beginning of this phase, at least

$$d_{i-1} := \hat{\gamma}^{(i-1)} - (i-1)\cdot\frac{L}{2^h\cdot f} \tag{2}$$

requests from phase i are already blocked. Furthermore, let us call a vertex v on level $i-1$ *bad* for Alg if, when Alg is given any instance corresponding to v as its input, phase i is bad for Alg. Phases and vertices that are not bad are called *good.* Moreover, let us define the set of requests from phase i that are blocked at the beginning of phase $i+1$ (including those that were admitted in phase i, which are blocking themselves) to be R_i.

Lemma 1. *If at least $d_i/2$ requests from R_i are open, then phase $i+1$ is bad for* Alg.

Proof. Underneath each open request from phase i, two requests appear in phase $i+1$. If $d_i/2$ requests from R_i are open, then at least d_i requests are presented in phase $i+1$ that are already blocked at the beginning of phase $i+1$. This matches the definition of a bad phase. □

Lemma 2. *The fraction of bad vertices on level i is at least*

$$\left(1 - \mathrm{e}^{-L/\left(2^h \cdot f^2\right)}\right)^i .$$

Proof. We prove the claim by induction on i. On level $i = 0$, there is only one vertex, namely the root representing all instances from $\mathcal{I}$. Obviously, the algorithm did not admit any requests before phase 1, and hence, $\hat{\gamma}^{(0)} = 0$. According to its definition, the root is bad if phase 1 is bad for Alg. This, in turn, is the case if at least 0 requests are blocked at the beginning of phase 1 when Alg is processing any instance; see (2). This is obviously true, and hence the base case is covered.

Let us now assume that the claim holds for some level $i-1$. We will show that the claim then also holds for level i. From now on, let v be some bad vertex on level $i-1$. First we prove that the fraction of bad vertices among the children of v is at least $1 - \mathrm{e}^{-L/(2^h \cdot f^2)}$.

Since v is bad, phase i must be bad for Alg, given any instance I_v corresponding to v as its input. Therefore, for each such instance I_v, at least d_{i-1} requests from phase i are already blocked at the beginning of phase i. As Alg admits $\gamma^{(i)}$ further requests in this phase, at least $d_{i-1} + \gamma^{(i)}$ requests from phase i are blocked at the beginning of phase $i+1$, including the admitted requests from phase i. This set of requests corresponds to the set R_i defined earlier. From Lemma 1, we know that, if at least $d_i/2$ requests from R_i are open, then phase $i+1$ is bad, which implies that at the beginning of phase $i+1$, at least d_i requests from phase $i+1$ are already blocked. Since the set of instances that correspond to a child w of v is a subset of the set of instances that correspond to v, and since we just showed that phase $i+1$ is bad for an arbitrary instance I_v, phase $i+1$ is also bad for each instance I_w; Fig. 6 gives an example.

Hence, a sufficient condition for w to be bad is that at least $d_i/2$ requests from R_i are open when giving Alg an instance I_w as its input. Thus, we have the following scenario. There are $N := L/2^h$ requests in phase i, as in every phase. Out of these, $M \geq M' := d_{i-1} + \gamma^{(i)}$ are blocked at the beginning of phase $i+1$. The set of these requests is R_i. Each child w of v corresponds to the set of instances in which the same set of $n := L/2^{h+1}$ requests from phase i

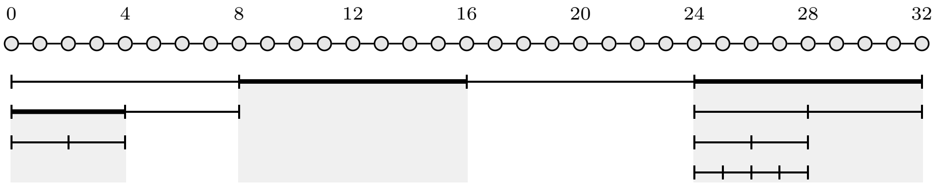


Fig. 6. This instance, like those from Figs. 3 and 5, corresponds to the vertex v from Fig. 4. The deterministic algorithm ALG admits $\hat{\gamma}^{(1)} = 2$ requests in phase 1. Hence, $d_1 = \hat{\gamma}^{(1)} - 1 \cdot L/(2^h \cdot f) = 2 - 32/(2^3 \cdot f) = 2 - 4/f < 2$. Since 2 requests from phase 2 are already blocked at the beginning of this phase, v is a bad vertex. Wlog, let w from Fig. 4 be the vertex on level 2 containing the instance from this picture. The vertex w is bad if, out of all requests from phase 3, at least $d_2 = \hat{\gamma}^{(2)} - 2 \cdot 32/(2^3 \cdot f) = 3 - 8/f < 3$ are blocked at the beginning of phase 3. Hence, in this instance, out of the 3 requests from phase 2 that are blocked after phase 2, at least $d_2/2 < 1.5$ must be open. This is clearly the case, since 2 such requests are open; thus, w is bad.

are open requests. We are interested in the fraction p of children w of v that correspond to instances in which at least $d_i/2$ requests from R_i are open. This is equivalent to the following. We have an urn containing N balls (i. e., requests), out of which $M \geq M'$ are black (i. e., in R_i), we draw n balls (i. e., open n requests) without replacement, and we are interested in the probability that the number of black balls drawn (i. e., open requests from R_i) is at least $d_i/2$.

Let X be a random variable that counts the number of open requests from R_i in this scenario. Note that X has a hypergeometric distribution with parameters $M \geq d_{i-1} + \gamma^{(i)}$, $N = L/2^h$, and $n = L/2^{h+1}$, and we are interested in $\Pr(X \geq d_i/2)$. With

$$\frac{d_i}{2} = \frac{1}{2}\left(\hat{\gamma}^{(i)} - i \cdot \frac{L}{2^h \cdot f}\right) = \frac{d_{i-1}}{2} + \frac{\gamma^{(i)}}{2} - \frac{L}{2^{h+1} \cdot f}$$

we obtain

$$\Pr\left(X \geq \frac{d_i}{2}\right) \geq \Pr\left(X > \frac{d_i}{2}\right) = 1 - \Pr\left(X \leq \frac{d_{i-1}}{2} + \frac{\gamma^{(i)}}{2} - \frac{L}{2^{h+1} \cdot f}\right). \quad (3)$$

Corollary 1 gives us a means to bound

$$\Pr\left(X \leq n \cdot \frac{M'}{N} - t \cdot n\right) = \Pr\left(X \leq \frac{L}{2^{h+1}} \cdot \frac{d_{i-1} + \gamma^{(i)}}{\frac{L}{2^h}} - t \cdot \frac{L}{2^{h+1}}\right)$$
$$= \Pr\left(X \leq \frac{d_{i-1} + \gamma^{(i)}}{2} - t \cdot \frac{L}{2^{h+1}}\right)$$

from above for any $t \geq 0$. Hence, choosing $t := 1/f$ yields

$$\Pr\left(X \leq \frac{d_{i-1} + \gamma^{(i)}}{2} - t \cdot \frac{L}{2^{h+1}}\right) = \Pr\left(X \leq \frac{d_{i-1}}{2} + \frac{\gamma^{(i)}}{2} - \frac{L}{2^{h+1} \cdot f}\right).$$

Then, according to Corollary 1, we get

$$\Pr\left(X \le \frac{d_{i-1} + \gamma^{(i)}}{2} - \frac{L}{2^{h+1} f}\right) \le \mathrm{e}^{-L/(2^h f^2)} . \tag{4}$$

Finally, combining (3) and (4), we obtain

$$\Pr\left(X \ge \frac{d_i}{2}\right) \ge 1 - \Pr\left(X \le \frac{d_{i-1}}{2} + \frac{\gamma^{(i)}}{2} - \frac{L}{2^{h+1} \cdot f}\right) \ge 1 - \mathrm{e}^{-L/(2^h f^2)} .$$

Hence, we have now shown that, for each bad vertex v on level $i-1$, the fraction of bad vertices among its children is at least $1 - \mathrm{e}^{-L/(2^h f^2)}$.

At this point, we are almost done. The only thing that remains to do is to exhibit a connection to the number of bad vertices on level i. All vertices on level $i-1$ have the same number of children and due to the induction hypothesis, for every bad vertex on level $i-1$, a fraction of at least $1 - \mathrm{e}^{-L/(2^h f^2)}$ of its children is bad. Hence, the fraction of bad vertices on level i is at least

$$\left(1 - \mathrm{e}^{-L/(2^h f^2)}\right)^{i-1} \cdot \left(1 - \mathrm{e}^{-L/(2^h f^2)}\right) = \left(1 - \mathrm{e}^{-L/(2^h f^2)}\right)^{i} . \qquad \square$$

A direct consequence from this result that many vertices are bad is that many instances are bad for Alg.

Corollary 2. *For any deterministic online algorithm* Alg*, the fraction of instances in $\mathcal{I}$ which are bad for* Alg *is at least*

$$\left(1 - \mathrm{e}^{-L/(2^h f^2)}\right)^{h} .$$

Proof. Every single instance corresponds to a leaf in the instance tree, and is thus located at level h. Plugging in the result of Lemma 2 proves the statement. $\square$

We have now shown that there are many bad instances for a given deterministic algorithm Alg for DPA. What we will show next is that the choice of the term "bad" was indeed justified for these instances, i. e., that Alg can actually only admit few requests on any bad instance.

Lemma 3. *Let* Alg *be an arbitrary but fixed deterministic algorithm for DPA, and let $I \in \mathcal{I}$ be a bad instance for* Alg*. Then, the gain of* Alg *on I is at most*

$$\mathrm{gain}(\textsc{Alg}(I)) \le \frac{L}{2^h}\left(1 + \frac{h}{f}\right) .$$

Proof. According to the definition of bad vertices (2), an instance (corresponding to a vertex on level h of the instance tree) is bad if there are at least $d_h = \hat{\gamma}^{(h)} - h \cdot L/(2^h f)$ requests from phase $h+1$ that are already blocked at the beginning of phase $h+1$. In this last phase, Alg is presented $L/2^h$ requests, and thus, the number of requests Alg can admit in this phase is

$$\gamma^{(h+1)} \leq \frac{L}{2^h} - \left(\hat{\gamma}^{(h)} - h \cdot \frac{L}{2^h f}\right) = \frac{L}{2^h} \cdot \left(1 + \frac{h}{f}\right) - \hat{\gamma}^{(h)} .$$

For the number of admitted intervals at the end of the computation and thus, for the total gain of Alg on any bad instance I, we obtain

$$\hat{\gamma}^{(h)} + \gamma^{(h+1)} \leq \hat{\gamma}^{(h)} + \frac{L}{2^h} \cdot \left(1 + \frac{h}{f}\right) - \hat{\gamma}^{(h)} = \frac{L}{2^h} \cdot \left(1 + \frac{h}{f}\right) . \qquad \square$$

All in all, we have shown that, for a fixed deterministic algorithm Alg, there are many instances on which Alg has only small gain. We now combine these results to prove Theorem 1.

Proof of Theorem 1. Consider an arbitrary but fixed deterministic algorithm Alg for DPA. The competitive ratio of Alg on an arbitrary bad instance I is, according to Lemma 3 and Observation 1,

$$c = \frac{\text{gain}(\text{Opt}(I))}{\text{gain}(\text{Alg}(I))} \geq \frac{\frac{(h+2)L}{2^{h+1}}}{\frac{L}{2^h} \cdot \left(1 + \frac{h}{f}\right)} = \frac{2^h}{2^{h+1}} \cdot \frac{h+2}{\left(1 + \frac{h}{f}\right)} = \frac{h+2}{2\left(1 + \frac{h}{f}\right)} .$$

Now consider an arbitrary online algorithm $\mathcal{A}$ with advice for DPA that reads b advice bits. We can interpret $\mathcal{A}$ in the usual way as a set of 2^b deterministic algorithms, $\mathcal{A} = \{\text{Alg}_1, \ldots, \text{Alg}_{2^b}\}$ [3,4]. From Corollary 2, we know that, for every such deterministic algorithm Alg_i, the fraction of good instances from $\mathcal{I}$, and hence the fraction of instances on which Alg_i has a competitive ratio of at most $(h+2)/(2(1+h/f))$, is at most

$$1 - \left(1 - \frac{1}{e^{L/(2^h f^2)}}\right)^h \leq \frac{h}{e^{L/(2^h f^2)}} ,$$

where we used Bernoulli's inequality (Fact 1), plugging in the values $n := h$ and $x := -1/e^{L/(2^h f^2)}$. Note that this is legitimate as long as $2^h > 0$ and $f > 0$, since then $L/(2^h f^2) \geq 0$ and hence $x \geq -1$.

Obviously, the best case for $\mathcal{A}$ is met if the good instances of all Alg_i's, $1 \leq i \leq 2^b$ are pairwise disjoint. Therefore, the number of deterministic algorithms that are necessary to guarantee a competitive ratio of at most $(h+2)/(2(1+h/f))$ for every instance from $\mathcal{I}$ is at least $(e^{L/(2^h f^2)})/h$.

To be able to distinguish this many different deterministic strategies, the number of advice bits the online algorithm Alg has to read is at least

$$\log_2\left(\frac{e^{L/(2^h f^2)}}{h}\right) = \frac{L}{2^h f^2} \cdot \log_2 e - \log_2 h . \qquad \square$$

Now that we have established a general lower bound that gives a minimum number of advice bits necessary to achieve a specific competitive ratio, we use Theorem 1 to get two concrete lower bounds by choosing concrete values for h and f that are in accordance with (1).

Corollary 3. *For any $c = c(L)$ with $1 < c \leq 1/2 \cdot (\log_2 L)/(\log_2 \log_2 L)^{1/4}$, any online algorithm for DPA that achieves a competitive ratio of c needs to read at least $\Omega\big(L/(4^c\, c^4)\big)$ advice bits.*

Finally, from Theorem 1 we can also derive a more concrete result on the number of advice bits necessary to achieve competitive ratios in the order of $\log_2 L$.

Corollary 4. *Let δ be an arbitrary constant with $0 < \delta < 1$. Any online algorithm for DPA that achieves a competitive ratio of $\delta/2 \cdot \log_2 L$ needs to read at least $\omega(L^{1-\varepsilon})$ advice bits, for any constant ε with $\delta < \varepsilon < 1$.*

Acknowledgments. The authors would like to thank Hans-Joachim Böckenhauer and Juraj Hromkovič for very valuable discussions.

References

1. Awerbuch, B., Bartal, Y., Fiat, A., Rosén, A.: Competitive non-preemptive call control. In: Proc. of SODA 1994, pp. 312–320. SIAM (1994)
2. Barhum, K., Böckenhauer, H.-J., Forišek, M., Gebauer, H., Hromkovič, J., Krug, S., Smula, J., Steffen, B.: On the Power of Advice and Randomization for the Disjoint Path Allocation Problem. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 89–101. Springer, Heidelberg (2014)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
4. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the Advice Complexity of Online Problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
5. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: Online algorithms with advice. Technical Report 614, Department of Computer Science. ETH Zurich (2009)
6. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
7. Carothers, N.L.: Real analysis. Cambridge University Press (2000)
8. Chvátal, V.: The tail of the hypergeometric distribution. Discrete Mathematics **25**(3), 285–287 (1979)
9. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. Theoretical Informatics and Applications (RAIRO) **43**(3), 585–613 (2009)
10. Epstein, L., Favrholdt, L.M.: Optimal preemptive semi-online scheduling to minimize makespan on two related machines. Operations Research Letters **30**(4), 269–275 (2002)

11. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoretical Computer Science **412**(24), 2642–2656 (2011)
12. Gupta, S., Kamali, S., López-Ortiz, A.: On Advice Complexity of the k-server Problem under Sparse Metrics. In: Moscibroda, T., Rescigno, A.A. (eds.) SIROCCO 2013. LNCS, vol. 8179, pp. 55–67. Springer, Heidelberg (2013)
13. Hromkovič, J., Královič, R., Královič, R.: Information Complexity of Online Problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
14. Komm, D., Královič, R.: Advice complexity and barely random algorithms. Theoretical Informatics and Applications (RAIRO) **45**(2), 249–267 (2011)
15. Rice, J. A.: Mathematical Statistics and Data Analysis. Duxbury Press, 3rd edn. (2007)
16. Renault, M.P., Rosén, A.: On Online Algorithms with Advice for the k-Server Problem. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 198–210. Springer, Heidelberg (2012)
17. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM **28**(2), 202–208 (1985)

Graph Algorithms III

Dynamic Tree Shortcut with Constant Degree

T.-H. Hubert Chan, Xiaowei Wu, Chenzi Zhang(✉), and Zhichao Zhao

The University of Hong Kong, Hong Kong, China
{hubert,xwwu,czzhang,zczhao}@cs.hku.hk

Abstract. Given a rooted tree with n nodes, the tree shortcut problem is to add a set of shortcut edges to the tree such that the shortest path from each node to any of its ancestors is of length $O(\log n)$ and the degree increment of each node is constant. We consider in this paper the dynamic version of the problem, which supports node insertion and deletion. For insertion, a node can be inserted as a leaf node or an internal node by sub-dividing an existing edge. For deletion, a leaf node can be deleted, or an internal node can be merged with its single child. We propose an algorithm that maintains a set of shortcut edges in $O(\log n)$ time for an insertion or deletion.

1 Introduction

The problem of adding a set of shortcut edges S to a tree $T(V, E)$ to reduce the *hop-diameter* of the resulting graph $G(V, E \cup S)$ has been studied for many years [1,4,5,11,14,15]. We call a simple path P from node u to v in G *straight* if the sequence of nodes in P is a sub-sequence of the unique path from u to v in T. The hop-diameter of the graph is the maximum number of edges in the shortest straight path between any two nodes in the graph. Given a rooted tree with n nodes, the hop-diameter of the tree can be as large as $\Theta(n)$. In applications like index-based search or broadcast in tree-networks, the hop-diameter (or the height) of the tree is crucial to the performance. To improve the efficiency, researchers have been analysing how to add shortcut edges between ancestor and descendant such that the hop-diameter of the resulting graph is small (i.e., $\log n$). Note that the hop-diameter can be easily reduced to $O(1)$ by adding a shortcut edge between each pair of ancestor and descendant in the tree. However, the resulting graph has maximum degree $\Omega(n)$. Hence another objective on shortcutting a tree is the *degree increment* on each node $u \in V$, which is the number of edges in S incident to u. We refer to the problem of shortcutting a tree as the tree-shortcut problem, which has application to spanners [6,10].

A natural extension of the tree-shortcut problem is to support operations on the tree [9]. In the dynamic setting, the tree structure will be changed in each iteration (by one operation) and we need to maintain the set of shortcut edges such that the hop-diameter and the degree increment on each node are

This research is partially funded by a grant from Hong Kong RGC under the contract HKU17200214E.

still bounded. We refer to the dynamic version of the tree-shortcut problem as the dynamic-tree-shortcut problem. We consider in this paper the dynamic-tree-shortcut problem that supports two kinds of operations: node insertion and deletion. The insertion operation inserts a node to the tree as a leaf node or as an internal node by sub-dividing an existing edge. The deletion operation deletes a leaf node (together with the incident edge) or an internal node that has a single child (the internal node is merged with its child after the deletion).

1.1 Related Works

Yao [15] studied a special version of tree-shortcut problem that has application to range query, in which the given tree is a single chain with n nodes. It is shown in [15] that we can add $\Theta(n)$ shortcut edges to guarantee $O(\alpha(n))$ hop-diameter, where $\alpha(n)$ is the inverse Ackermann function introduced by Tarjan [12]. Note that the maximum degree increment on each node in their construction is $\Omega(n^{\frac{1}{\alpha(n)}})$. Their hop-diameter is proved to be asymptotically optimal, if the total number of shortcut edges is $\Theta(n)$. Their result was later generalized to arbitrary trees by Chazelle [5] and Thorup [14] such that $\Theta(n)$ shortcut edges are added to a tree with n nodes to obtain $O(\alpha(n))$ hop-diameter and $\Omega(n^{\frac{1}{\alpha(n)}})$ degree increment. Conjecture on a generalized version of the tree-shortcut problem on directed graphs were also made [13], but it was later disproved [7,8]. Solomon and Elkin [11] considered the trade-off between degree increment and hop-diameter of the tree-shortcut problem, and proposed an algorithm that guarantees $O(k)$ degree increment on each node and $O(\log_k n + \alpha(k))$ hop-diameter by adding $O(n)$ shortcut edges to the tree, where k is any integer less than n. Note that [11] obtained, for the first time, constant degree increment and $O(\log n)$ hop-diameter for the tree-shortcut problem. However, operations such as node insertion and deletion are not supported by their data structure.

As a parallel line of research, the dynamic-tree-shortcut problem is also extensively studied. Sleator and Tarjan [9] derived a dynamic data structure that can be used to shortcut a tree such that the degree increment on each node and the hop-diameter are both bounded by $O(\log n)$. Moreover, they showed that the data structure can be maintained in $O(\log n)$ time against operations such as cutting a tree into two by removing one edge, or linking two trees by adding an edge. Note that node insertion and deletion are special cases of the above operations and hence are also supported by [9]. A centroid decomposition of the tree and a biased search tree [3] on each centroid path were maintained in [9]. We adopt similar ideas on constructing shortcut edges for the dynamic-tree-shortcut problem and improve their result by reducing the degree increment to a constant.

1.2 Our Contribution

Given a rooted tree with n nodes, we construct in this paper a dynamic data structure for shortcut edges that guarantees $O(1)$ degree increment on each node and $O(\log n)$ hop-diameter. Moreover, we show that our data structure can be maintained against node insertions and deletions in $O(\log n)$ time by adding or deleting $O(\log n)$ shortcut edges after each operation. We summarize and compare some related results as follows.

	Degree-increment	Hop-diameter	Update time for node insertion/deletion
[5,14]	$\Omega(n^{1/\alpha(n)})$	$O(\alpha(n))$	Not supported
[11]	$O(k)$	$O(\log_k n + \alpha(k))$	Not supported
[9]	$O(\log n)$	$O(\log n)$	$O(\log n)$
This paper	$O(1)$	$O(\log n)$	$O(\log n)$

In the table, n is the number of nodes in the tree (before node insertion/deletion), k is any integer less than n and $\alpha(n)$ is the inverse Ackermann function of n.

2 Preliminaries

Given a rooted tree $T(V, E)$, the *distance* $d(u, v)$ between two nodes $u, v \in V$ is the number of edges in the unique path from u to v in the tree. For each internal node $u \in V$, let Child(u) be the set of children of u in the tree. We use $[n]$ to denote $\{1, 2, \ldots, n\}$ for any positive integer n and $\log n$ to denote $\log_2 n$.

Definition 1 (Operation). *There are two kinds of operations on the tree.*

- **Insertion.** *Operation $(x, insertion)$ inserts a new node x to the tree $T(V, E)$. The new node x can be inserted as a child of an existing node or an internal node that subdivides an existing tree edge. Note that x indicates a position in the new tree.*
- **Deletion.** *Operation $(x, deletion)$ deletes an existing node x, which is either a leaf node or an internal node with only one child, from the tree $T(V, E)$. If x is a leaf node, then the edge incident to x is also deleted; otherwise x is merged with its single child. Note that x specifies a node in the original tree.*

Definition 2 (Shortcut Edge). *In a rooted tree, a* shortcut edge *is an edge between a node and one of its descendants.*

Let S be a set of shortcut edges built on $T(V, E)$. Let $G(V, E \cup S)$ be the graph obtained by adding the shortcut edges in S to the tree $T(V, E)$. In the dynamic setting, we always use G and T to denote the current graph and tree (after the operations and updates).

For each edge $e = (u, v) \in E \cup S$, let $l_e = d(u, v)$ be its *length*. Note that $l_e = 1$ for all $e \in E$. A path (from node u_1 to node u_q) is denoted by a sequence of nodes $P = (u_1, u_2, \ldots, u_q)$. The *length* of path $P = (u_1, u_2, \ldots, u_q)$ is the sum of lengths of the edges in the path, which is $\sum_{i=1}^{q-1} l_{(u_i, u_{i+1})} = \sum_{i=1}^{q-1} d(u_i, u_{i+1})$. With a slight abuse of notation, we use $|P|$ to denote the number of nodes in P and $u \in P$ to denote that $u \in V$ is in the path.

Definition 3 (Straight Path). *A path $P = (u_1, u_2, \ldots, u_q)$ is* straight *iff its length equals the distance $d(u_1, u_q)$ between u_1 and u_q.*

Definition 4 (Hop-distance, Hop-diameter, Degree). *The* hop-distance *$h_G(u,v)$ between two nodes $u,v \in V$ in G is the minimum number of edges in a straight path from u to v. The* hop-diameter *$\Delta_G = \max_{u,v\in V}\{h_G(u,v)\}$ of G is the maximum hop-distance between any two nodes. Let $\mathsf{deg}_S(u)$ be the number of shortcut edges incident to $u \in V$. We call $\mathsf{deg}_S(u)$ the* degree *of u. The maximum degree in G is denoted by $\mathsf{deg}_S := \max_{u\in V}\{\mathsf{deg}_S(u)\}$.*

Theorem 1. *A set of shortcut edges S for a rooted tree $T(V,E)$ of n nodes can be found such that graph $G(V, E\cup S)$ ensures $\Delta_G = O(\log n)$ and $\mathsf{deg}_S = O(1)$. For each operation on T, shortcut edges S can be maintained in $O(\log n)$ time, where n is the number of nodes before the operation.*

We prove Theorem 1 in the rest of this paper by providing a data structure in Section 3, and an algorithm in Section 4 to maintain the data structure against operations. The following definitions are important for the construction and the update algorithm. Let T_u be the sub-tree rooted at node u. We denote by $|T_u|$ the number of nodes in T_u.

Definition 5 (Heavy Child Mapping). *For any internal node u, a child x of u is* heavy *iff $4|T_x| \geq \max_{y\in Child(u)}\{|T_y|\}$. A mapping f that maps each internal node u to one of its children is called a* heavy child mapping *iff $f(u)$ is a heavy child of u for all internal node $u \in V$.*

Note that the heavy child mapping may be non-unique. For each internal node u we set $f(u) = \arg\max_{x\in\mathsf{Child}(u)}\{|T_x|\}$ when the tree $T(V,E)$ is given at the beginning. We may change the mapping f in the update algorithm. However, we always maintain f as a heavy child mapping after each update. Based on the heavy child mapping $f(u)$, let $\hat{w}_u = |T_u| - |T_{f(u)}|$ for each internal node u and $\hat{w}_u = 1$ for each leaf node u.

Definition 6 (Proper Weighting). *A weighting function w that assigns an integer weight w_u to each node $u \in V$ is a* proper weighting *iff $w_u \in [\frac{\hat{w}_u}{4}, 4\hat{w}_u]$.*

Note that a proper weighting is based on a heavy child mapping. The proper weighting on all nodes is also not unique. We set $w_u = \hat{w}_u$ for each node $u \in V$ when the tree $T(V,E)$ is given at the beginning. We may change the weights of nodes in the update algorithm. However, we always maintain the proper weighting w after each update. Given a path $P = (u_1, u_2, \ldots, u_q)$ and a proper weighting w, let $w(P) = \sum_{i=1}^{q} w_{u_i}$. Unless otherwise specified, we assume that a heavy child mapping f and a proper weighting w are maintained throughout this paper.

3 The Structure of Shortcut Edges

To build shortcut edges on the tree, we first break the tree into *centroid paths* and then build shortcut edges within each centroid path. We further break a centroid path into *buckets*, each of which contains $O(\log n)$ consecutive nodes. We build Static-Path-Shortcut (Section 3.1) on each bucket and Dynamic-Path-Shortcut (Section 3.2) between buckets using biased-skip-list. Due to page limit, missing proofs are provided in the full version.

Centroid Paths. For each internal node $u \in V$, we call $(u, f(u))$ a *joint*. Let $E_f = \{(u, f(u)) | u \text{ is an internal node in } V\}$ be the set of all joints. We call the edges in $E \backslash E_f$ *links*. Given a rooted tree $T(V, E)$ and a heavy child mapping f, we partition the tree into paths $P_1, P_2, \ldots, P_r$ by removing all *links*. We call each of those paths a *centroid path*, and the set of centroid paths the *Tree-Decomposition* of T. The set of centroid paths can be induced by $T_f(V, E_f)$. For all $i \in [r]$, we have $P_i = (x, f(x), f(f(x)), \ldots)$ for some $x \in V$. We call x the *top-node* of P_i.

Claim 1. *The nodes in the path P from any node $u \in V$ to the root of the tree are contained in at most $O(\log n)$ centroid paths.*

Proof. Note that P consists of joints and links. For each link (x, y) in P such that $x \in \text{Child}(y)$, we have $x \neq f(y)$. By definition of heavy child mapping f, we have $|T_y| \geq \frac{5}{4}|T_x|$, which means that the number of links in P is $O(\log n)$. Since all consecutive joints in P are contained in the same centroid path, the nodes in P are contained in at most $O(\log n)$ centroid paths. $\square$

3.1 Static Path Shortcut

In this section, we consider how to add shortcut edges to a path. Given P and w, the Static-Path-Shortcut defined as follows is a set of shortcut edges and must be unique. We can run Alg. 1 as $\text{SPS}(P, 1, q, t)$ to construct it.

Definition 7 (Static-Path-Shortcut). *Given a path $P = (u_1, u_2, \ldots, u_q)$ and a proper weighting w, let $t_k = \sum_{i \in [k]} w_{u_i}$ for all $k \in [q]$ and $t_0 = 0$. If $q \leq 2$, then the* Static-Path-Shortcut *of P is an empty set; otherwise the Static-Path-Shortcut of P contains the edge (u_1, u_m), where $t_{m-1} < \frac{t_q}{2} \leq t_m$, and the edges in the Static-Path-Shortcuts of sub-paths $(u_2, \ldots, u_{m-1})$ and $(u_{m+1}, \ldots, u_q)$. A sub-path $(u_i, \ldots, u_j)$ is empty if $j < i$.*

Lemma 1. *Given a path $P = (u_1, u_2, \ldots, u_q)$ and proper weighting w, its Static-Path-Shortcut S can be constructed in $O(q)$ time such that $\mathsf{deg}_S = O(1)$ and $h_G(u_1, u_i) = O(\log \frac{w(P)}{w_{u_i}})$ for $i \in [q]$, where we consider P as a tree rooted at u_1.*

Algorithm 1. $\text{SPS}(P, i, j, t)$:

```
Input: Path $P = (u_1, u_2, \ldots, u_q)$, positions $1 \leq i < j \leq q$ and cumulative weights $t$.
1: if $i + 1 < j$ then
2:     $\tau \leftarrow \frac{t_j + t_{i-1}}{2}$;
3:     find integer $l \geq 0$ such that $t_{i+2^l-1} \leq \tau \leq t_{i+2^{l+1}-1}$ or $t_{j-2^{l+1}+1} \leq \tau \leq t_{j-2^l+1}$.
4:     find $m$ such that $t_{m-1} < \tau \leq t_m$ in the above range using binary search.
5:     return $\{(u_i, u_m)\} \cup \text{SPS}(P, i+1, m-1, t) \cup \text{SPS}(P, m+1, j, t)$
```

3.2 Dynamic Path Shortcut

Definition 8 (Path-Decomposition). *Given a path $P = (u_1, u_2, \ldots, u_q)$, a* Path-Decomposition *of P is a sequence of* buckets $B_1, B_2, \ldots, B_k$ *such that $B_i = (u_{b_{i-1}+1}, u_{b_{i-1}+2}, \ldots, u_{b_i})$ for all $i \in [k]$, where $0 = b_0 < b_1 < b_2 < \ldots < b_k = q$. The number of nodes in each bucket B_i, denoted by $|B_i| = b_i - b_{i-1}$, satisfies $2\log(4n) \le |B_i| \le 10\log(4n)$. For the case when $|P| < 2\log(4n)$, we set $k = 1$ and the bucket size requirement is removed.*

Note that the Path-Decomposition of a path P may be non-unique. We ensure $4\log(4n) \le |B_i| \le 8\log(4n)$ when the tree is given for the first time but only maintain $2\log(4n) \le |B_i| \le 10\log(4n)$ in the update algorithms. If $|P| \le 8\log(4n)$, we use one bucket to contain all nodes. We consider a bucket as a sub-path of the path P. We use $u \in B_i$ to denote that u is in bucket B_i. Given a proper weighting w, let $w(B_k) = 1$ and $w(B_i) = \sum_{u \in B_{i+1}} w_u$ be the *weight* of B_i, for $i \in [k-1]$. Note that by definition, the weight of a bucket is the total weight of nodes in the *next* bucket. The reason behind this definition will be clear in the proof of Lemma 2.

Definition 9 (Biased-Skip-List [2]). *Given a sequence of buckets $B_1, B_2 \ldots, B_k$ and a proper weighting w, a* biased-skip-list *h assigns an integer* height *h_i to each bucket B_i such that (1) $\lfloor \log w(B_i) \rfloor \le h_i \le \log(4n)$ for all $i \in [k]$; (2) $h_1 = h_k = h_{\max} = \max_{i \in [k]}\{h_i\}$.*

Definition 10 (Successor Pointer). *Given a biased-skip-list h, for each height $t \le h_{\max}$, buckets of height at least t are kept in sorted order in a doubly linked list called t-list. The* successor pointer *$s_{i,t}$ points to the successor of B_i in t-list.*

Recall the following algorithms [2] of direct search and finger search.

Algorithm 2. Direct_Search(h, s, x):

```
Input: biased-skip-list $h$, successor pointers $s$ and target bucket $B_x$.
1: $i \leftarrow 1$; $j \leftarrow h_{\max}$;
2: while $i \ne x$ do
3:     if $s_{i,j} \le x$ then
4:         $i \leftarrow s_{i,j}$;
5:     else if $i < x$ then
6:         $j \leftarrow j - 1$;
```

Algorithm 3. Finger_Search(h, s, x, y):

```
Input: biased-skip-list $h$, successor pointers $s$, start bucket $B_x$ and target bucket $B_y$.
1: $i \leftarrow x$; $j \leftarrow h_x$;
2: while $i \ne y$ do
3:     if $j < h_i$ AND $s_{i,j+1} \le y$ then                ▷ Up phase.
4:         $i \leftarrow s_{i,j+1}$; $j \leftarrow j + 1$;
5:     else if $s_{i,j} \le y$ then
6:         $i \leftarrow s_{i,j}$;
7:     else if $i < y$ then                                  ▷ Down phase.
8:         $j \leftarrow j - 1$;
```

Fact 1 ([2]). *Given a biased-skip-list, the time complexity for direct search of bucket B_x is $O(\log \frac{w(P)}{w(B_x)})$ and the time complexity for finger search is $O(\log w(P))$. The time complexity to maintain the biased-skip-list(s) is $O(\log W)$ for the following operations, where $W = \max\{W_a, W_b\}$, W_a and W_b are the total weights of buckets before and after the operation, respectively: (1) adding, deleting or re-weighting one bucket. (2) splitting a biased-skip-list into two. (3) merging two biased-skip-lists into one.*

Definition 11 (Dynamic-Path-Shortcut). *Given a Path-Decomposition $B_1, B_2, \ldots, B_k$ of path $P = (u_1, u_2, \ldots, u_q)$ and a biased-skip-list h on the buckets, the* Dynamic-Path-Shortcut *is a set of shortcut edges that contains*
- *edges in the Static-Path-Shortcut of each bucket B_i,*
- *$(u_{b_{i-1}+1}, u_{b_i-h_i+1})$ and $(u_{b_i-h_i+1}, u_{b_i})$ for all $i \in [k]$,*
- *$(u_{b_i-t+1}, u_{b_j-t+1})$ and $(u_{b_i-t+2}, u_{b_j-t+1})$ for all i, j, t such that $2 \le t \le h_i$ and $B_j = s_{i,t}$,*
- *$(u_{b_i}, u_{b_{i+1}})$ for all $i \in [k-1]$.*

Lemma 2. *Given a path P with proper weighting w, its Dynamic-Path-Shortcut S guarantees $\mathsf{deg}_S = O(1)$, $h_G(u_1, u_i) = O(\log \frac{w(P)}{w_{u_i}})$ and $h_G(u_i, u_j) = O(\log(4n))$ for all $u_i, u_j \in P$.*

Lemma 3. *To maintain the Dynamic-Path-Shortcut(s) of path P, the time complexity for the update algorithms is $O(\log(4n))$ for the following operations.*
- *Adding, removing or re-weighting (while maintaining a proper weighting) a node $u \in P$.*
- *Splitting the path P into two paths by deleting one edge in P.*
- *Concatenate a path P', with its Dynamic-Path-Shortcut, to the end of P.*

3.3 Dynamic Tree Shortcut

Definition 12 (Dynamic-Tree-Shortcut). *Given a rooted tree $T(V, E)$, a heavy child mapping f and a proper weighting w, the* Dynamic-Path-Shortcut *of T contains all edges in the Dynamic-Path-Shortcut of each centroid path P_i in the Tree-Decomposition $P_1, P_2, \ldots, P_r$ of T.*

Lemma 4. *Given a rooted tree T of n nodes, the Dynamic-Path-Shortcut S guarantees $\mathsf{deg}_S = O(1)$ and $h_G(u, v) = O(\log n)$ for each ancestor-descendant pair (u, v).*

Proof. First observe that $\mathsf{deg}_S = O(1)$ by Lemma 2 and the disjunction of centroid paths.

Let P be the tree path from node v to its ancestor u, which is a sub-path of the tree path from v to the root. By Claim 1, nodes in P are contained in $k = O(\log n)$ centroid paths. Let $P'_1, P'_2, \ldots, P'_k$ be those centroid paths such that $u \in P'_1$, $v \in P'_k$ and the parent of the top-node of P'_{i+1} is contained in P'_i, for all $i \in [k-1]$.

Let x_i be the top-node of P'_i, for all $i \in [k]$. For all $i \in [k-1]$, let $y_i \in P'_i$ be the parent of x_{i+1}. Let $y_k = v$. Hence P follows $v = y_k \rightsquigarrow (x_k, y_{k-1}) \rightsquigarrow (x_{k-1}, y_{k-2}) \rightsquigarrow \ldots \rightsquigarrow (x_2, y_1) \rightsquigarrow u$. By definition of proper weighting, for all $1 < i \le k$ we have

$$w(P'_i) \le 4|T_{x_i}| \le 4(|T_{y_{i-1}}| - |T_{f(y_{i-1})}|) = 4\hat{w}_{y_{i-1}} \le 16 w_{y_{i-1}}.$$

Instead of following tree path P, by Lemma 3, there exists a constant c such that we can use the Dynamic-Path-Shortcut of each centroid path to reach x_i from y_i by a straight path of length $c \log \frac{w(P'_i)}{w_{y_i}}$ for all $1 < i \le k$, and reach u from y_1 by a straight path of length $O(\log n)$.

Hence the hop-distance between v and u using Dynamic-Path-Shortcut is

$$\begin{aligned} h_G(u,v) &\le O(\log n) + \sum_{i=2}^{k} (c \log \frac{w(P'_i)}{w_{y_i}} + 1) \le O(\log n) + k + c \sum_{i=2}^{k} (\log \frac{16 w_{y_{i-1}}}{w_{y_i}}) \\ &\le O(\log n) + (4c+1)k + c \log \frac{w_{y_1}}{w_v} = O(\log n). \end{aligned}$$ □

4 Update Algorithm for Operations

We have introduced how to build shortcut edges on a given tree such that the degree increment and the diameter are bounded after the construction. In this section, we will describe how to update the Dynamic-Tree-Shortcut after each operation (x, insertion/deletion) by giving update algorithms. The real challenge for maintaining the Dynamic-Tree-Shortcut is that the heavy child mapping f and proper weighting w, if not updated accordingly, may not hold due to the change of the number of nodes. Throughout this section, we assume the current number of nodes n in the tree is large enough.

By Definition 12, as long as the heavy child mapping f, the proper weighting w and the Dynamic-Path-Shortcut of each centroid path are maintained, the Dynamic-Tree-Shortcut is maintained as Lemma 4. For the maintainance of Dynamic-Path-Shortcut, the most crucial part is to maintain the buckets. We define three statuses of an object: *intact*, *safe* and *risky*, where an object is a bucket or a node or a joint. We call an object *maintained* iff it is in one of those three statuses. Note that the status of an object may change after each operation.

	intact	safe	risky
Bucket B	$\frac{\lvert B\rvert}{\log(4n)} \in [4,8]$	$\frac{\lvert B\rvert}{\log(4n)} \in [3,4) \cup (8,9]$	$\frac{\lvert B\rvert}{\log(4n)} \in [2,3) \cup (9,10]$
Node u	$\frac{w_u}{\hat{w}_u} = 1$	$\frac{w_u}{\hat{w}_u} \in [\frac{1}{2},1) \cup (1,2]$	$\frac{w_u}{\hat{w}_u} \in [\frac{1}{4},\frac{1}{2}) \cup (2,4]$
Joint $(u, f(u))$	$\frac{\lvert T_{f(u)}\rvert}{\max_{v\in \text{Child}(u)} \lvert T_v\rvert} = 1$	$\frac{\lvert T_{f(u)}\rvert}{\max_{v\in \text{Child}(u)} \lvert T_v\rvert} \in [\frac{1}{2},1)$	$\frac{\lvert T_{f(u)}\rvert}{\max_{v\in \text{Child}(u)} \lvert T_v\rvert} \in [\frac{1}{4},\frac{1}{2})$

For the case when a centroid path P is of size $|P| \le 10\log(4n)$, since one bucket B is sufficient to contain the whole path, we call the bucket B intact if $|B| \le 8\log(4n)$; safe if $8\log(4n) < |B| \le 9\log(4n)$ and risky if

$9\log(4n) < |B| \leq 10\log(4n)$. By the above definitions, at the beginning when the tree is given, the Dynamic-Tree-Shortcut we construct in Section 3 ensures that all buckets, joints and nodes are **intact**.

Due to page limit, the update algorithms for the reconstruction of risky buckets, risky joints and risky nodes are deferred to the full version. All those algorithms rebuild a risky object and turn it intact.

We give the following algorithm to update the Dynamic-Tree-Shortcut for each operation $(x, \text{insertion/deletion})$. Given an operation $(x, \text{insertion/}$ $\text{deletion})$, we call a joint $(u, f(u))$ *touched* if u is an ancestor of x; a node u *touched* if u is in the path P from x to the root and $f(u) \notin P$. Let $k_1 = 18$ and $k_2 = 36$ be constant parameters.

Algorithm 4. Update(x, insertion/deletion)

```
1: insert or delete a node at position $x$
2: update the Dynamic-Path-Shortcut of the centroid path containing $x$ ▷ Lemma 3
3: $n \leftarrow n + 1$ if $x$ is inserted; $n \leftarrow n - 1$ if $x$ is deleted
4: rebuild an arbitrary **risky bucket** (if exist)
5: rebuild $k_1$ touched **risky joints** that are closest to $x$ (if exist)
6: rebuild $k_2$ touched **risky nodes** that are closest to $x$ (if exist)
```

Lemma 5. *Given a Dynamic-Tree-Shortcut of a rooted tree such that all buckets, joints and nodes are intact, Alg. 4 keeps all buckets, joints and nodes maintained for any sequence of operations.*

We prove Lemma 5 by analysing the buckets, joints and nodes one by one as follows. The analysis of maintaining nodes is in the full version.

4.1 Maintaining the Buckets

Lemma 6. *Alg. 4 keeps all buckets maintained for any sequence of operations, if initially all buckets are intact.*

Proof. First we show that it takes more than $\frac{n}{2}$ operations to turn a bucket from intact to risky, or from safe to not-maintained, where n is number of nodes in the tree before those operations. Notice that if the size of a bucket is changed after operation $(x, \text{insertion/deletion})$, then it must be in the centroid path containing x. By Alg. 4 line 2 and Lemma 3, the bucket will be reconstructed and become intact. Hence the only case a bucket B changes from intact to risky is due to the change of the number of nodes in the tree, while the size of the bucket remains unchanged. Let n' be the number of nodes in the tree when bucket B becomes risky. Then we have either $4\log(4n) \leq |B| < 3\log(4n')$ or $9\log(4n') < |B| \leq 8\log(4n)$. In the first case we have $n' - n > n^{4/3} - n = n(n^{1/3} - 1) > n > \frac{n}{2}$ and in the second case we have $n - n' > n - n^{8/9} = n(1 - n^{-1/9}) > \frac{n}{2}$. Since n is changed by 1 after each operation, the number of operations needed to change a bucket from intact to risky is more than $\frac{n}{2}$. Similar argument can be applied to show that the number of operations needed to change a bucket from safe to not-maintained is also more than $\frac{n}{2}$.

Assume the contrary of Lemma 6 and let bucket B be the first bucket that is not maintained. Consider the last moment t when B is safe and let n^* be the number of nodes in the tree at moment t. Since each safe or risky bucket at moment t must be of size at least $2\log(4n^*)$, the number of safe or risky buckets is at most $\frac{n^*}{2\log(4n^*)}$. Consider moment $t+\frac{n^*}{2}$, after $\frac{n^*}{2}$ operations. Note that at this moment, all intact buckets at moment t and buckets created after moment t must not be risky. Moreover, bucket B must be risky at this moment, by the above analysis. However, since B is risky between moment t and $t+\frac{n^*}{2}$, a risky bucket other than B must be rebuilt after each of the past $\frac{n^*}{2}$ operations, which is a contradiction since there are at most $\frac{n^*}{2\log(4n^*)}$ risky buckets between moment t and moment $t+\frac{n^*}{2}$. □

4.2 Maintaining the Joints

Note that the status of a joint $(u, f(u))$ will not change if the sizes of all subtrees of u remain unchanged. Hence we only need to maintain the joints that are touched, after each operation. For each joint $(u, f(u))$, between its two consecutive rebuilds, we call the last moment before $(u, f(u))$ is changed from intact to safe the *marginal moment* of $(u, f(u))$.

Definition 13 (Class). *We put joints into* classes C_i, $i \in [\log n]$, *right after their marginal moments. A joint* $(u, f(u))$ *is put into* C_i *iff* $|T_{f(u)}| \in [2^i, 2^{i+1})$. *A joint* $(u, f(u)) \in C_i$ *is removed from* C_i *if it is touched by* 2^{i-1} *operations after its marginal moment.*

We show that joints in classes are either safe or intact. Since risky joints may be rebuilt, each joint can be put into and removed from classes multiple times.

Lemma 7. *For all* $i \in [\log n]$, *each joint* $(u, f(u)) \in C_i$ *is either safe or intact.*

Proof. It suffices to show that $(u, f(u)) \in C_i$ will become risky only after being touched by more than 2^{i-1} operations. Consider the marginal moment of $(u, f(u))$. Since $(u, f(u))$ is intact at this moment, we have $|T_{f(u)}| \geq |T_x|$ for all $x \in \text{Child}(u)$. Let $t = |T_{f(u)}| \in [2^i, 2^{i+1})$ at this marginal moment.

Now consider the time when $(u, f(u))$ becomes risky. Let $t' = |T_{f(u)}|$ at this moment. Since $(u, f(u))$ is risky, there exists $y \in \text{Child}(u)$ such that $y \neq f(u)$ and $|T_y| > 2t'$. Note that at the marginal moment of $(u, f(u))$, we have $|T_y| \leq t$, as argued above. Since only one node is inserted or deleted in each operation, the number of operations between the above two moments, is more than $|t - t'| + |t - 2t'| = t(|1 - \frac{t'}{t}| + |1 - 2\frac{t'}{t}|) \geq \frac{t}{2} \geq 2^{i-1}$. □

Lemma 8. *Each operation touches at most* 4 *joints in each class.*

Proof. Fix one operation and one class C_i. Consider any joint $(u, f(u)) \in C_i$. Similar to the proof of Lemma 7, let $t = |T_{f(u)}| \in [2^i, 2^{i+1})$ at the marginal moment of $(u, f(u))$. Note that at the marginal moment, u has another child $v \neq f(u)$ such that $|T_v| = t$, since $(u, f(u))$ becomes safe after the next operation. Since $(u, f(u))$ can only be touched by $2^{i-1} \leq \frac{t}{2}$ operations, at any moment when $(u, f(u)) \in C_i$ we have $\max_{y\in \text{Child}(u)}\{|T_y|\} \leq 2\min\{|T_{f(u)}|, |T_v|\}$, as argued in Lemma 7. Hence for all joint $(u, f(u)) \in C_i$, we have $\max_{y\in \text{Child}(u)}\{|T_y|\} \leq \frac{2}{3}|T_u|$. Let $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_l, f(x_l))$ be all joints in C_i touched by this operation such that x_{j+1} is a descendant of x_j for all $j \in [l-1]$, we have

$$2^i - 2^{i-1} \leq |T_{f(x_l)}| \leq \frac{2}{3}|T_{x_l}| \leq (\frac{2}{3})^{l-1}|T_{x_2}| \leq (\frac{2}{3})^{l-1}(2^{i+1}+2^{i-1}) = 5(\frac{2}{3})^{l-1}2^{i-1},$$

which implies that $l \leq 1 + \left\lfloor \frac{\log 5}{\log \frac{3}{2}} \right\rfloor = 4$. □

Lemma 9. *Alg. 4 keeps all joints maintained for any sequence of operations, if initially all joints are intact.*

Proof. Assume the contrary and let $(u, f(u))$ be the first joint that is not maintained such that all other joints in T_u are maintained. Consider the last moment t when $(u, f(u))$ is safe and the moment t' when $(u, f(u))$ becomes not maintained. Let $r = |T_{f(u)}|$ at moment t and $r' = |T_{f(u)}|$ at moment t'. Let $v \in \text{Child}(u)$ such that $|T_v| > 4r'$ at moment t'. Note that $|T_v| \leq 2r$ at moment t since $(u, f(u))$ is safe at moment t. Let m be the number of operations that touch $(f(u), f(f(u)))$ or $(v, f(v))$ between moment t and t'. Similar to the proof of Lemma 7, we have $m \geq |r - r'| + |2r - 4r'| \geq \frac{r}{2}$.

Now consider all joints in $T_{f(u)}$ and T_v at moment t. Note that there are at most $3r$ joints. We count the number of reconstructions of risky joints between moment t and t'. A risky joint may become risky again after being rebuilt and hence a joint may be counted multiple times. Note that the classes do not contain the touched joints $(u, f(u))$ such that $|T_{f(u)}| = 1$. However, at most three such joints are touched by each operation. The total number of reconstructions of risky joints between moment t and t' can be upper bounded by $3r + 3m+$the number of joints that are removed from the node-classes, which by Lemma 8, is $3r + 3m + \sum_{i\in[\log n]} \frac{4m}{2^{i-1}} \leq 3r + 3m + 8m \leq 17m$.

Since by assumption $(u, f(u))$ is not maintained at moment t', by Alg. 4 line 5, $k_1 = 18$ risky joints in $T_{f(u)}$ or T_v are rebuilt after each of those m operations, which implies that $k_1 m = 18m \leq 17m$ and is a contradiction. □

4.3 Time Complexity Analysis

Lemma 10. *The time complexity of Algorithm 4 is $O(\log n)$.*

Proof. As argued in Lemma 3, updating the Dynamic-Path-Shortcut after inserting or deleting one node can be done in $O(\log n)$ time. As proved in the full version, rebuilding a risky bucket/joint/node can be done in $O(\log n)$ time. Hence we only need to find the risky bucket/joints/nodes in $O(\log n)$ time.

If we use a max-heap to store the sizes of all buckets and a min-heap to store the sizes of the buckets that contain only a fraction of some centroid path, then it takes $O(\log n)$ time to find the bucket with maximum size and the bucket with minimum size such that does not contain a whole centroid path. Note that if there exist any risky buckets, then one of those two buckets must be risky. We prove the following fact in the full version.

Claim 2. *It takes $O(\log n)$ time to find the k_1 touched risky joints that are closest to x after each operation $(x, insertion/deletion)$.*

As proved in Lemma 1, at most $O(\log n)$ nodes can be touched by each operation and hence $O(\log n)$ time suffices to update their weights and to identify the risky nodes. □

References

1. Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product queries. Technical report (1987)
2. Bagchi, A., Buchsbaum, A.L., Goodrich, M.T.: Biased skip lists. Algorithmica **42**(1), 31–48 (2005)
3. Samuel, W.: Bent, Daniel D Sleator, and Robert E Tarjan. Biased search trees. SIAM Journal on Computing **14**(3), 545–568 (1985)
4. Bodlaender, H.L., Tel, G., Santoro, N.: Trade-offs in non-reversing diameter. Nord. J. Comput. **1**(1), 111–134 (1994)
5. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. Algorithmica **2**, 337–361 (1987)
6. Elkin, M., Solomon, S.: Optimal euclidean spanners: really short, thin and lanky. In: STOC, pp. 645–654 (2013)
7. Hesse, W.: Directed graphs requiring large numbers of shortcuts. In: SODA, pp. 665–669 (2003)
8. Raskhodnikova, S.: Transitive-Closure Spanners: A Survey. In: Goldreich, O. (ed.) Property Testing. LNCS, vol. 6390, pp. 167–196. Springer, Heidelberg (2010)
9. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)
10. Solomon, S.: From hierarchical partitions to hierarchical covers: optimal fault-tolerant spanners for doubling metrics. In: STOC, pp. 363–372 (2014)
11. Solomon, S., Elkin, M.: Balancing Degree, Diameter and Weight in Euclidean Spanners. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 48–59. Springer, Heidelberg (2010)
12. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975)
13. Thorup, M.: On shortcutting digraphs. In: Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 1992, London, UK, pp. 205–211. Springer (1993)
14. Thorup, M.: Parallel shortcutting of rooted trees. J. Algorithms **23**(1), 139–159 (1997)
15. Yao, A.C.-C.: Space-time tradeoff for answering range queries (extended abstract). In: STOC, pp. 128–136 (1982)

The Rectilinear Steiner Tree Problem with Given Topology and Length Restrictions

Jens Maßberg(✉)

Institut für Optimierung und Operations Research, Universität Ulm, Ulm, Germany
jens.massberg@uni-ulm.de

Abstract. We consider the problem of embedding the Steiner points of a Steiner tree with given topology into the rectilinear plane. Thereby, the length of the path between a distinguished terminal and each other terminal must not exceed given length restrictions. We want to minimize the total length of the tree.

The problem can be formulated as a linear program and therefore it is solvable in polynomial time. In this paper we analyze the structure of feasible embeddings and give a combinatorial polynomial time algorithm for the problem. Our algorithm combines a dynamic programming approach and binary search and relies on the total unimodularity of a matrix appearing in a sub-problem.

Keywords: Steiner trees with given topology · Rectilinear Steiner trees · Dynamic programming · Totally unimodular · Shallow light Steiner trees

1 Introduction

The Rectilinear Steiner Tree Problem With Given Topology And Length Restrictions can be stated as follows. The input (S, T, r, p, l) consists of a set of terminals T with positions $p : T \to \mathbb{R}^2$, a tree S with $T \subseteq V(S)$, a distinguished terminal $r \in T$ - called the *root* of the tree - and length restrictions $l_t \in \mathbb{R}_{\geq 0}$ for all $t \in T$.

The task is to find an embedding $\pi : V(S) \to \mathbb{R}^2$ of the vertices of the tree into the plane with $\pi(t) = p(t)$ for all $t \in T$, such that for all $t \in T$ the length $d_\pi(t)$ of the unique path from r to t in S with edge set $E_S[r, t]$ has length at most l_t, that is,

$$d_\pi(t) = \sum_{\{v,w\} \in E_S[r,t]} ||\pi(v) - \pi(w)||_1 \leq l_t \tag{1}$$

and the total length

$$c(\pi) := \sum_{\{v,w\} \in E(S)} ||\pi(v) - \pi(w)||_1 \tag{2}$$

of the tree is minimized. The tree S is called *Steiner tree* and the vertices in $V(S) \setminus S$ *Steiner points*. Throughout this paper we assume w.l.o.g. that the root is placed at the origin, that is, $p(r) = (0,0)$. By adding Steiner points and edges of length zero we can assume that the terminals are leaves of S and that all Steiner points have degree 3. Moreover, we denote by $\pi_x(v)$ and $\pi_y(v)$ the x- and y-coordinate, respectively, of $\pi(v)$ for an embedding π and a vertex $v \in V(S)$.

A further generalization of the problem is to extend it to other metrics or to consider length restrictions between any pair of terminals. In this paper we restrict ourselves to the ℓ_1 metric and length restrictions between one distinguished vertex and all other terminals, as this case has a strong application in practice.

Our problem is motivated by an application arising in VLSI design, where one of the main challenges is to build so-called repeater trees. These are tree-like structures consisting of wires and possibly so-called repeater circuits and their task is to distribute a signal from a source circuit to several sink circuits. Thereby, the signal is delayed. In order to guarantee, that the chip works on the desired speed, timing constraints are given, that is, the signal has to arrive at each sink circuit not later than a given individual time bound. There are several heuristics to build such repeater trees (see e.g. [1]).

A repeater tree can be modeled as a Steiner tree connecting the source and the sinks and containing repeater circuits at some of the Steiner points. The length of a repeater tree corresponds to its power consumption. So the question arises, if the length of a given tree can be reduced by moving the positions of the Steiner points. Bartoschek et al. [1] have shown, that by adding repeater circuits at appropriate positions the delay of a signal on a path from the source to a sink is approximately proportional to the length of the path. Thus the timing constraints directly yield length restrictions on root-terminal paths. It turns out, that the *Rectilinear Steiner Tree Problem with given Topology and Length restrictions* is a good model for the task to minimize the power consumption of given repeater trees without changing their topology.

If we are allowed to change the topology of the tree, the problem becomes NP-hard, as it contains the *Rectilinear Steiner Tree Problem* [4]. If, additionally, $l_t = ||p(s) - p(t)||_1$ for all $t \in T$, that is, all root-terminal paths are shortest paths, we end at the *Rectilinear Steiner Arborescence Problem*, which is also NP-hard ([8,9]). In the case where the length restrictions are the same for all terminals we have the case of *Shallow Light Steiner Trees* (see e.g. [7]).

However, if we have to keep the topology, but do not have any length restrictions, an optimal embedding can be computed in linear time using dynamic programming (see e.g. [6]). To our knowledge, the problem of embedding a Steiner tree with a given topology satisfying length restrictions has not been considered yet. In this paper we present the first combinatorial polynomial time algorithm that computes an optimal embedding.

Figure 1 (i) shows an instance with seven terminals drawn as black squares and 5 Steiner points drawn as white circles. Figure (ii) shows an optimal solution if there are no length restrictions. In Figure (iii) an optimal solution is shown, if we have length restrictions $l_{t_1} = 5$, $l_{t_2} = 6$ and $l_s = \infty$ otherwise. If there are no length restrictions, then there always exists an optimal solution where the Steiner points are positioned at the so called Hanan grid on T (see [5]). With length restrictions, this is no longer true. Nevertheless, we prove that if the positions of the terminals and the length restrictions are integral, then there always exists an solution on half-integral positions.

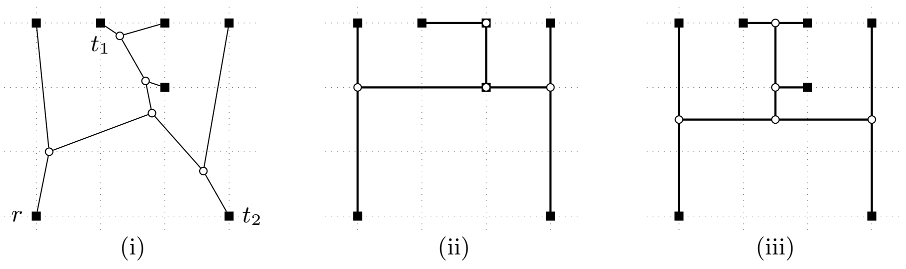


Fig. 1. Instance (i), optimal embedding without length restrictions (ii) and optimal embedding with length restrictions $l_{t_1} = 5$ and $l_{t_2} = 6$ (iii). The regular dotted grid has a lattice spacing of 1.

The problem can be formulated as a linear program by extending the LPs presented in [2] [6]. Therefore it can be solved in polynomial time by non-combinatorial algorithms. Nevertheless, we are interested in a combinatorial algorithm for the problem.

After introducing several definitions concerning the movement of components of the tree in Section 2, we present our main observations in Section 3. Among others, we prove that there always exists an optimal embedding where the Steiner points are on half-integral positions. Based on this observation, we introduce in Section 4 a dynamic programming algorithm which is the main ingredient to achieve a pseudo-polynomial time algorithm. Refining this algorithm we finally gain a polynomial time algorithm in Section 5.

2 Moving Components

Before we come to the main observations of the paper we examine how the movements of Steiner points of a given embedding influence the total length of the tree and the length of root-terminal paths. First we start with several definitions that we need throughout this paper.

If π is an embedding, then an *x-component* C at *position* $x(C)$ with respect to π, $x(C) \in \mathbb{R}$, is a connected subtree C of T such that all vertices in C have x-coordinate $x(C)$. An x-component C is called *maximal* if there does not exist any x-component C' with $C \subsetneq C'$. A component always depends on the embedding π. In the following, we omit π in the notation if it is clear from the context. In an analogous way we define a y-component C at position $y(C)$. In the remainder of the paper we introduce several definitions and state lemmata concerning x-components. By symmetry, these definitions and lemmata also hold for y-components.

Let $\Gamma(V(C))$ be the neighbors of the vertices of C. For an x-component C we define

$$\Gamma^{\pi}_{<}(C) := \{v \in \Gamma(V(C)) : \pi_x(v) < x(C)\} \text{ and} \tag{3}$$
$$\Gamma^{\pi}_{>}(C) := \{v \in \Gamma(V(C)) : \pi_x(v) > x(C)\}. \tag{4}$$

In an analogous way we define $\Gamma^{\pi}_{<}(C)$ and $\Gamma^{\pi}_{>}(C)$ for a y-components C. If C is a component not containing r, then the *predecessor* of C is the unique vertex $v \in \Gamma^{\pi}_{>}(C) \cup \Gamma^{\pi}_{<}(C)$ such that v is on the root-w path for all $w \in V(C)$. For simplicity of notation we define

$$\operatorname{sign}(C) = \begin{cases} 1 & \text{if the predecessor of } C \text{ is in } \Gamma_{<}(C) \\ -1 & \text{otherwise.} \end{cases} \tag{5}$$

If C is an x-component with respect to some embedding π then we say that we *move C by δ* if we replace π by the embedding π' defined by

$$\pi'(v) := \begin{cases} \pi(v) + (\delta, 0) & \text{for all } v \in V(C) \setminus T, \\ \pi(v) & \text{otherwise.} \end{cases} \tag{6}$$

We say, that we move C *towards its predecessor* if $\delta \cdot \operatorname{sign}(C) < 0$.

If C is a maximal component containing no terminals, then we define $R(C)$ to be the set of terminals t such that the unique root-t path P passes C, that is, $V(P) \cap V(C) \neq \emptyset$ and the path enters and leaves C at the same side, that is, we have either $|V(P) \cap \Gamma_{>}(C)| = 2$ or $|V(P) \cap \Gamma_{<}(C)| = 2$. If we choose $\delta \in \mathbb{R}$ with $|\delta|$ small enough and move C by δ, then the length of all root-t paths with $t \in R(C)$ change by $2\operatorname{sign}(C)\delta$. The length of any other root-terminal path does not change.

Figure 2 illustrates some of the definitions.

If π and π' are two embedding, then we say that π' *preserves the local order of* π if for every edge $(v, w) \in E(S)$ we have

$$(\pi_x(v) \leq \pi_x(w)) \quad \Rightarrow \quad (\pi'_x(v) \leq \pi'_x(w)) \text{ and} \tag{7}$$
$$(\pi_y(v) \leq \pi_y(w)) \quad \Rightarrow \quad (\pi'_y(v) \leq \pi'_y(w)). \tag{8}$$

Note, that by (7) if $\pi_x(v) = \pi_x(w)$ then $\pi'_x(v) = \pi'_x(w)$ and analogously for y. This implies that each component with respect to π is also a component with respect to π' (but not necessarily the other way round!). Moreover, if v is a vertex of an x-component that contains terminals, we have $\pi_x(v) = \pi'_x(v)$.

Now we can analyze how the length of the embedding and of root-terminal paths change if we move maximal components simultaneously and the local order is preserved.

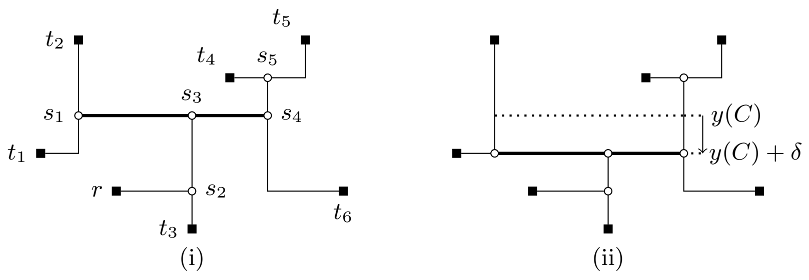


Fig. 2. (i) An embedding π with a maximal y-component C with $V(C) = \{s_1, s_3, s_4\}$, predecessor s_2, $\Gamma_>(C) = \{t_2, s_5\}$, $\Gamma_<(C) = \{t_1, s_2, t_6\}$, $\mathrm{sign}(C) = 1$ and $R(C) = \{t_1, t_6\}$. (ii) Embedding obtained by moving C by $\delta < 0$. The new embedding preserves the local positions of π. The length of all root-t with $t \in R(C)$ changed by $2\mathrm{sign}(C)\delta = 2\delta$ (< 0).

Lemma 1. *Let π be an embedding, Δ be the set of all maximal x- and y-components, and $\delta_C \in \mathbb{R}$ for $C \in \Delta$. Denote by π' the embedding we obtain by moving each component $C \in \Delta$ by δ_C. If π' preserves the local order of π then*

$$c(\pi') = c(\pi) + \sum_{C\in\Delta} \delta_C \left(|\Gamma^\pi_<(C)| - |\Gamma^\pi_>(C)|\right). \tag{9}$$

Moreover we have for all $t \in T$:

$$d_{\pi'}(t) = d_\pi(t) + \sum_{C\in\Delta : t\in R(C)} 2sign(C) \cdot \delta_C. \tag{10}$$

Proof. Consider an x-component $C \in \Delta_x$. If we move C, then only the length of edges $\{v, w\} \in E(S)$ with $v \in V(C)$ and $w \notin V(C)$ are changed. Let $\{v, w\}$ be such an edge and assume $w \in \Gamma^\pi_<(C)$, that is $\pi_x(w) < \pi_x(v)$. As the local order is preserved, we have $\pi'_x(w) \leq \pi'_x(v)$. But then moving C by δ increases the length of the edge $\{v, w\}$ by δ. In an analogous way we see that the length of the edge decreases by δ if $w \in \Gamma^\pi_>(C)$. Summing up the changes over all components we obtain (9).

Now consider a terminal $t \in T$. Again, as the local order is preserved by π', the length of the root-t path is only influenced by components C with $t \in R(C)$. Consider such a component C. If we move C by $|\delta_C|$ towards the predecessor of C, the length of the path is reduced by $2|\delta_C|$. On the other hand, if we move C in the other direction by $|\delta_C|$, then the length is increased by $2|\delta_C|$. In total, the length changes by $\mathrm{sign}(C)2 \cdot \delta_C$. Summing up over all such components, we obtain (10). □

The following observation is crucial in order to prove that there exist optimal solutions that are half-integral.

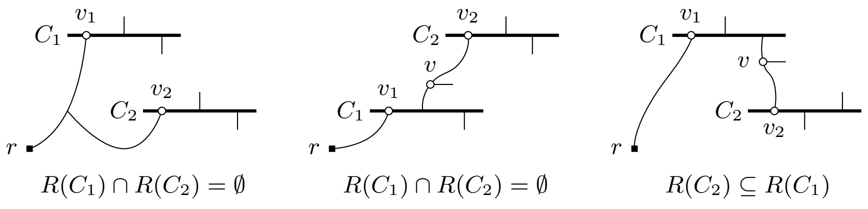

C_1
v_1
C_2
v_2
r
v
$R(C_1) \cap R(C_2) = \emptyset$
$R(C_1) \cap R(C_2) = \emptyset$
$R(C_2) \subseteq R(C_1)$

Fig. 3. The possible positions of two maximal y-components within a tree.

Lemma 2. *If Δ is a set of maximal x-components that do not contain terminals, then $\{R(C)\}_{C\in\Delta}$ is a laminar family.*

Proof. Let $C_1, C_2 \in \Delta$. By definition $V(C_1) \cap V(C_2) = \emptyset$. For $i \in \{1,2\}$ let v_i be the vertex of $V(C_i)$ that is adjacent to the predecessor of C_i. Note that v_i is on the unique r-t path for every $t \in R(C_i)$ (see Figure 3). Now assume that neither v_1 is on the r-v_2 path nor v_2 is on the r-v_1 path. Then $R(C_1) \cap R(C_2) = \emptyset$. Otherwise, assume w.l.o.g. that v_1 is on the r-v_2 path. In this case there exists a unique vertex v on the v_1-v_2 path satisfying $v \in \Gamma^{\pi}_{>}(C_1) \cup \Gamma^{\pi}_{<}(C_1)$. Now note that for all $t \in R(C_2)$ the length of the r-t path changes when moving C_1 if and only if the length of the r-v path changes when moving C_1. Hence, $R(C_2) \subseteq R(C_1)$ or $R(C_2) \cap R(C_1) = \emptyset$. This implies the desired result. □

Before we continue with the main result we make another simple observation:

Proposition 1. *If π is an embedding and there exists a vertex $t \in T$ such that $d_\pi(t) > ||p(t) - p(r)||_1$, then there exists a component C such that moving C towards its predecessor decreases the length of the root-t path.*

3 Main Section

In this section we prove that if all terminals are on integral coordinates and all length restrictions are integral, then there exists an optimal half-integral embedding. More precisely we prove that for any given feasible embedding π there exists a feasible half-integral embedding σ of at most the same cost such that the ℓ_∞ distance between the positions of a vertex in both embeddings is at most 0.5. To this end we consider a sub problem that can be formulated as a linear program based on a totally unimodular matrix.

We start with some observations on half-integral embeddings.

Proposition 2. *Every half-integral embedding has half-integral cost.*

Proof. Obviously, all edges in such an embedding have half-integral lengths and thus the total length is also half-integral. □

Proposition 3. *In every half-integral embedding π the length of every root-terminal path has integral length.*

Proof. Let $t \in T$ and denote by P the unique root-t path in S. If P is a shortest path, then the length of P is $||\pi(r) - \pi(t)||_1$, which is integral. If P is not a shortest path, then by Proposition 1 there exists a component C such that moving C towards its predecessor decreases the length of P. As π is half-integral, we can move C by 0.5 towards its predecessor, reducing the length of P by 1 and obtaining a new half-integral embedding π'. Then by induction the length of P must be integral. □

The main theorem of this section is the following.

Theorem 1. *If π is an embedding for an instance (S, T, r, p, l) where all positions and length restrictions are integral, then there exists an half-integral embedding σ with $\max_{v\in V} ||\pi(v) - \sigma(v)||_\infty \le 0.5$ and $c(\sigma) \le c(\pi)$.*

Proof. For $x \in \mathbb{R}$ we denote by $I(x)$ the smallest interval in $\mathbb{R}$ with half-integral boundaries such that x is in the interior of the interval, that is,

$$I(x) := [\lceil 2x - 1\rceil/2, \lfloor 2x + 1\rfloor/2]\,. \tag{11}$$

For a point $(x, y) \in \mathbb{R}^2$ we set $I((x, y)) := I(x) \times I(y)$. We show that there exists an half-integral embedding σ with

$$\sigma(v) \in I(\pi(v)) \text{ for all } v \in V \tag{12}$$

such that $c(\sigma) \le c(\pi)$.

Let σ be a feasible embedding for (S, T, r, p, l) of minimum cost satisfying (12). If there are several such embeddings we choose one with a minimal number of components that are not on half-integral coordinates. We denote this number by $N(\sigma)$ and prove that $N(\sigma) = 0$. Suppose that this is not the case. The idea is to move maximal components such that $N(\sigma)$ gets smaller without increasing $c(\sigma)$. As π trivially satisfies (12), we have $c(\sigma) \le c(\pi)$.

Let Δ_x and Δ_y be the sets of maximal x- and y-components, respectively, with respect to σ that are not on half-integral coordinates and set $\Delta := \Delta_x \dot\cup \Delta_y$. Then $N(\sigma) = |\Delta|$. For $C \in \Delta$ we set

$$z^*_C := \begin{cases} x(C) - \lfloor 2x(C)\rfloor/2 & C \in \Delta_x, \\ y(C) - \lfloor 2y(C)\rfloor/2 & C \in \Delta_y. \end{cases} \tag{13}$$

Consider a vector $z \in [0, 0.5]^\Delta$. Starting with the embedding σ and moving each component $C \in \Delta$ by $z_C - z^*_C$ in x- or y-direction, depending if $C \in \Delta_x$ or $C \in \Delta_y$, respectively, we obtain a new embedding $\tau(z)$. Note that by the definition of z^*_C this embedding is half-integral if and only if $z \in \{0, 0.5\}^\Delta$. Observe that $\tau(0)$ is half-integral, but it does not necessarily satisfy the length restrictions. Since by construction $\tau(0)$ preserves the local order of σ we can apply Lemma 1 and conclude that for all $t \in T$ the length of the root-t path with respect to $\tau(0)$ is

$$d_{\tau(0)}(t) = d_\sigma(t) + \sum_{C\in\Delta : t\in R(C)} 2\text{sign}(C) \cdot (-z^*_C). \tag{14}$$

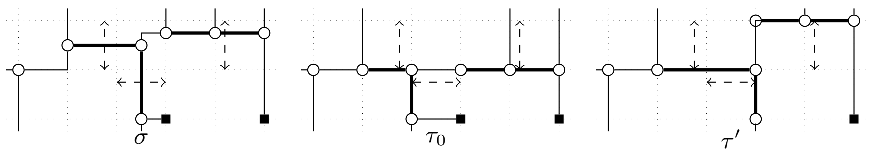


Fig. 4. Detail of an embedding σ with three maximal components not on half-integral positions. The embeddings τ_0 and τ' preserve the local order of σ.

As $\tau(0)$ is integral, this length is also integral by Proposition 3.

Using z as a variable we can formulate a linear program reflecting the new cost of the embedding $\tau(z)$ and the length restrictions, under the assumption that $\tau(z)$ preserves the local oder of σ:

$$\min c(\sigma) + \sum_{C \in \Delta} (z_C - z_C^*) \cdot (|\Gamma^\pi_<(C)| - |\Gamma^\pi_>(C)|),$$

$$\text{s.t. } d_\sigma(t) + \sum_{C \in \Delta : t \in R(C)} 2\text{sign}(C)(z_C - z_C^*) \le l_t \qquad \forall t \in T \tag{15}$$

$$\text{and } 0 \le 2z_C \le 1 \qquad \forall C \in \Delta. \tag{16}$$

As $z = z^*$ is a feasible solution the linear program has an optimal solution (see also Figure 4). Substituting $2z_C$ by z'_C for all $C \in \Delta$ and using (14) we obtain the modified linear program (P'):

$$\min \sum_{C \in \Delta} z'_C/2 \cdot (|\Gamma^\pi_<(C)| - |\Gamma^\pi_>(C)|),$$

$$\text{s.t. } \sum_{C \in \Delta : t \in R(C)} \text{sign}(C) z'_C \le l_t - d_{\tau(0)}(t) \qquad \forall t \in T \tag{17}$$

$$\text{and } 0 \le z'_C \le 1 \qquad \forall C \in \Delta. \tag{18}$$

We show that the matrix A defined by the left side of the inequalities (17) is totally unimodular. Note that all entries of a column of A are either non-negative or non-positive. Thus multiplying all rows with non-positive entries by -1 we obtain a non-negative matrix where each column correspond to the characteristic vectors of $\{R(C)\}_{C \in \Delta} = \{R(C)\}_{C \in \Delta_x} \dot\cup \{R(C)\}_{C \in \Delta_y}$. Recall, that by Lemma 2 the sets $\{R(C)\}_{C \in \Delta_x}$ and $\{R(C)\}_{C \in \Delta_y}$ are laminar families. We conclude that the rows of A correspond to the characteristic vectors of the union of two laminar families. Edmonds [3] proved, that such matrices are totally unimodular.

Consequently, as the right hand side of (17) is integral, the constraints in (18) are integral and A is totally unimodular, there exists an optimal solution for (P') that is integral which further implies that the original LP has an half-integral optimal solution $\hat{z}$. But then $\tau(\hat{z})$ is also half integral and satisfies (12) and $c(\tau(\hat{z})) \le c(\sigma)$.

If $\tau(\hat{z})$ preserves the local order of σ, then $\tau(\hat{z})$ is the embedding we are looking for and we are done. Otherwise choose $\lambda > 0$ minimal such that τ^λ defined by $\tau^\lambda(v) = \lambda\sigma(v) + (1-\lambda)\tau(\hat{z})(v)$ for all $v \in V$ preserves the local order of σ. Note that λ is well defined as the set of all embeddings preserving the local order of σ is closed. As the cost and length functions are convex, τ^λ is a feasible embedding and $c(\tau^\lambda) \leq \lambda c(\sigma) + (1-\lambda)c(\tau(\hat{z})) \leq c(\sigma)$. Moreover, every maximal x- or y-component of σ is also an x- or y-component of τ^λ, respectively, implying $N(\tau^\lambda) \leq N(\sigma)$. As $\tau^{\lambda-\delta}$ is not preserving the local order of σ for all $\delta < 0$ but for $\delta \geq 0$ there must be an edge (v,w) with $(\sigma_x(v) < \sigma_x(w)$ and $\tau_x^\lambda(v) = \tau_x^\lambda(w))$ or $(\sigma_y(v) < \sigma_y(w)$ and $\tau_y^\lambda(v) = \tau_y^\lambda(w))$. As the only components that are moved are not on half-integral position with respect to σ, we must have $N(\tau^\lambda) < N(\sigma)$ contradicting the choice of σ. This finishes the proof. □

Conclusion 2 *If all positions and length restrictions are integral, then there exists an optimal embedding that is half-integral.*

4 Dynamic Programming

A consequence of the previous section is, that any non-optimal half-integral embedding can be improved by small half-integral movements of the Steiner points.

Lemma 3. *If π is an half-integral embedding that is not optimal, then there exists an half-integral embedding π' with $\pi(v) - \pi'(v) \in \{-0.5, 0, 0.5\}^2$ for all $v \in V(S)$ and $c(\pi') \leq c(\pi) - 0.5$.*

Proof. Let σ be an optimal half-integral embedding. For $\lambda \in (0,1)$ we define π_λ by $\pi_\lambda(v) = \lambda\pi(v) + (1-\lambda)\sigma(v)$ for all $v \in V(S)$. As π is not optimal and by the convexity of the length function, π_λ is a feasible embedding and we have $c(\pi_\lambda) \leq \lambda c(\pi) + (1-\lambda)c(\sigma) \leq c(\pi)$. Choose λ small enough such that $\max_{v \in V(S)} ||\pi(v) - \pi_\lambda(v)||_\infty < 0.5$. Now Theorem 1 yields an half-integral embedding π' satisfying $\max_{v\in V} ||\pi(v) - \pi'(v)||_\infty \leq \max_{v \in V} ||\pi(v) - \pi_\lambda(v)||_\infty + ||\pi_\lambda(v) - \pi'(v)||_\infty < 1$ and $c(\pi') \leq c(\pi_\lambda) < c(\pi)$. The claim follows by observing that π' and π are half-integral. □

This lemma gives a direct idea for an algorithm based on dynamic programming to improve a non-optimal half-integral embedding. In the following, we interpret S as an arborescence rooted at r and denote by $\Gamma^+(v)$ the children of a vertex $v \in V(S)$. For simplicity of notation we set $\pi_\delta(v) := \pi(v) + \delta$ for $\delta \in \{-0.5, 0, 0.5\}$. Moreover, we expand the definition of length restrictions to Steiner points: Initially we set $l_t^\pi := l_t$ for all $t \in T$. For each vertex $v \in V(S)$ whose children have a length restriction, we recursively set

$$l_v^\pi = \min_{w \in \Gamma^+(v)} l_w^\pi - ||\pi(v) - \pi(w)||_1.$$

Given an half-integral embedding π we want to compute an half-integral embedding π' with $\pi(v) - \pi'(v) \in \{-0.5, 0, 0.5\}^2$ and $c(\pi')$ minimal. Note, that in this case the length of every root-terminal path changes by at most $2n$. As, additionally, π' is half-integral, $l_v^{\pi'}$ is half-integral and $|l_v^{\pi'} - l_v^{\pi}| \leq 2n$ for all $v \in V(S)$.

Thus it is sufficient to compute for every vertex $v \in V(S)$, every translation $\delta \in \{-0.5, 0, 0.5\}^2$ and every possible length restriction $l \in \{l_v^{\pi} - 2n, l_v^{\pi} - 2n + 0.5, \ldots, l_v^{\pi} - 2n + 2n - 0.5, l_v^{\pi} + 2n\}$ the minimum length $\gamma(v, \delta, l)$ of an embedding of the arborescence rooted at v such that v is positioned at $\pi_\delta(v)$ and v satisfies the length restriction l. For a terminal t we have $\gamma(t, \delta, l) = 0$ if $\delta = (0, 0)$ and $l \leq l_t$. Otherwise, we set $\gamma(t, \delta, l) = \infty$. For all other vertices $v \in V(T)$ we obviously have $\gamma(v, \delta, l) =$

$$\sum_{w \in \Gamma^+(v)} \min_{\delta' \in \{-0.5,0,0.5\}^2} \gamma\left(w, \delta', l - ||\pi_\delta(v) - \pi_{\delta'}(w)||_1\right) + ||\pi_\delta(v) - \pi_{\delta'}(w)||_1.$$

It follows, that the length of an optimal embedding π' with $\pi(v) - \pi'(v) \in \{-0.5, 0.0.5\}^2$ is $\gamma(r, (0, 0), 0)$. This number can be computed in $O(n^2)$ time: There are $O(n^2)$ different triples (v, δ, l) for which $\gamma(v, \delta, l)$ has to be computed and each of these computations can be done in constant time.

To compute a global optimal solution, we start with the trivial embedding, where all Steiner points are positioned at the root. This solution has cost $C = \sum_{t \in T} ||p(t)||_1$. Then we apply the dynamic programming approach as long as the cost of the newly computed embedding decreases. As the cost is reduced by at least 0.5 in every round, we must obtain an optimal embedding after $2C$ iterations. Thus our algorithm has a pseudo polynomial running time of $O(Cn^2)$. In the next section we show how to refine this approach in order to achieve a polynomial running time.

5 An Optimal Polynomial Time Algorithm

We refine the ideas of the previous sections in order to obtain a polynomial time algorithm for our problem. In the first algorithm the Steiner points are moved by at most 0.5 in each direction in every call of the dynamic programming. The idea of the refined algorithm is to move the Steiner points by 2^k for a suitable $k \in \mathbb{Z}$ in the first rounds. As soon as no improvements can be obtained by moving Steiner points by 2^k, we reduce the moving distance to 2^{k-1} and continue applying the dynamic programming (see Algorithm 1). Repeating this procedure we finally move the Steiner points by 0.5, obtaining an optimal embedding.

Due to space limitation and as the proofs are very technical without giving much more insight into the problem we omit a detailed description.

Theorem 3. *The rectilinear Steiner tree embedding problem with length restrictions can be solved in polynomial time by a combinatorial algorithm.*

Proof (outline). Consider Algorithm 1. In the last iteration of the main loop, the dynamic programming is applied with step width 0.5 until no further improvements are made. Therefore, by Lemma 3 the computed solution is optimal.

```
   Input: An integral instance $(S, T, r, p, l)$ with $p(r) = (0, 0)$.
   Output: An optimal embedding $\pi : V(G) \setminus T \to \mathbb{R}^2$.
1  Set $m \leftarrow \min\{m' \in \mathbb{N} : |p_x(v)| \le 2^{m'}$ and $|p_y(v)| \le 2^{m'}\ \forall v \in V(S)\}$;
2  Set $\pi(s) \leftarrow (0, 0)$ for all Steiner points $s \in V(S) \setminus T$;
3  $k \leftarrow m$;
4  while $k \ge -1$ do
5      Improve embedding $\pi$ by applying the dynamic programming with step
       width $2^k$ until no further length reduction is obtained;
6      $k \leftarrow k - 1$;
7  end
8  return $\pi$;
```

Algorithm 1. Optimal polynomial time algorithm.

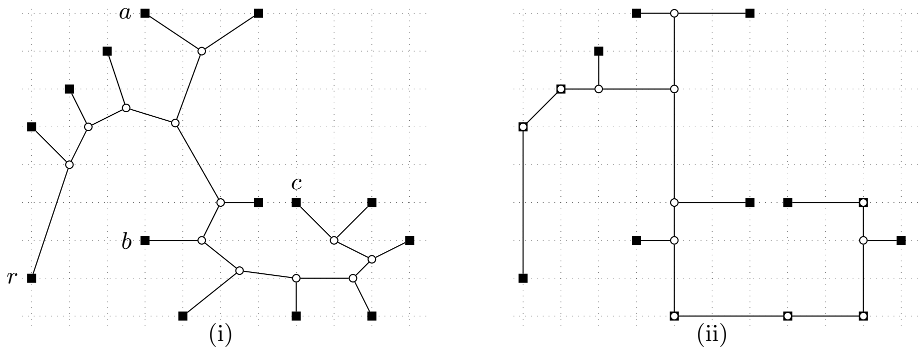


Fig. 5. Instance for the Steiner tree embedding problem (i) and an optimal embedding if there are no length restrictions (ii)

It can be shown, that for each k, the dynamic programming is called at most $O(n)$ times. As each call of the dynamic programming can be implemented to run in time $O(n^2)$, the total running time is $O(mn^3)$ where $m = \lceil \log_2(\max\{|p_x(t)|, |p_y(t)| : t \in T\}) \rceil$. Finally observe, that m is polynomially bounded in the size of the instance. $\square$

Figures 5 and 6 show how the algorithm works on an example. Figure 5 (i) shows the instance and Figure 5 (ii) an optimal embedding of length 35 if there are no length restrictions. In Figure 6 the embeddings computed by our algorithm are shown. As input we used the instance from Figure 5 with length restrictions $l_a = 10$, $l_b = 11$ and $l_c = 20$. As $\max\{|p_x(t)|, |p_y(t)| : t \in T\} = 10$ we have $m = 4$. Thus the algorithm begins with an embedding where all Steiner points are 2^{m-1}-integral (Figure 6 (i)). The last one is the final optimal embedding of length 37.5. For each k the dynamic programming is called at most twice, the first time the length is reduced, the second time an embedding of the same cost is computed, proving that it is an optimal one.

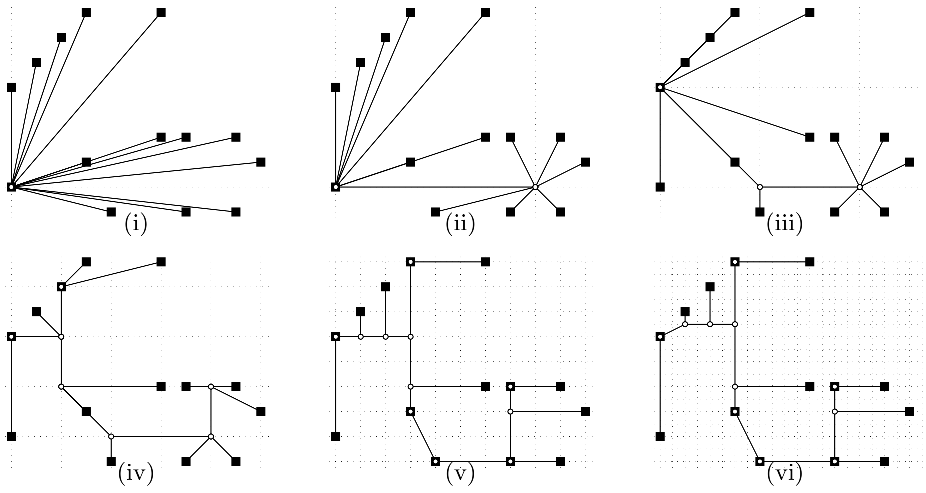


Fig. 6. Run of the algorithm on the instance shown in Figure 5 (i) with length restrictions $l_a = 10$, $l_b = 11$ and $l_c = 20$. Figure (vi) shows the final optimal solution.

References

1. Bartoschek, C., Held, S., Maßberg, J., Rautenbach, D., Vygen, J.: The repeater tree construction problem. Information Processing Letters **110**(24), 1079–1083 (2010)
2. Victor Cabot, A., Francis, R.L., Stary, M.A.: A network flow solution to a rectilinear distance facility location problem. AIIE Transactions **2**(2) 132–141 (1970)
3. Edmonds, J.: Submodular functions, matroids and certain polyhedra. In: Gordon, Breach (eds.) Combinatorial Structures and Their Applications, New York, pp. 68–87 (1970)
4. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete. SIAM Journal on Applied Mathematics **32**(4), 826–834 (1977)
5. Hanan, M.: On Steiner's problem with rectilinear distance. SIAM Journal on Applied Mathematics **14**(2), 255–265 (1966)
6. Jiang, T., Wang, L.: Computing shortest networks with fixed topologies. Advances in Steiner Trees. vol. 6. Combinatorial Optimization, pp. 39–62. Springer, US (2000)
7. Kortsarz, G., Peleg, D.: Approximating the weight of shallow Steiner trees. Discrete Applied Mathematics **93**(2–3), 265–285 (1999)
8. Rao, S.K., Sadayappan, P., Hwang, F.K., Shor, P.W.: The rectilinear Steiner arborescence problem. Algorithmica **7**(1–6), 277–288 (1992)
9. Shi, W., Chen, S.: The rectilinear Steiner arborescence problem is NP-complete. SIAM Journal on Computation **35**(3), 729–740 (2005)

Compact Monotone Drawing of Trees

Xin He(✉) and Dayu He

Department of Computer Science and Engineering,
State University of New York at Buffalo, Buffalo, NY 14260, USA
{xinhe,dayuhe}@buffalo.edu

Abstract. A monotone drawing of a graph G is a straight-line drawing of G such that, for every pair of vertices u, w in G, there exists a path P_{uw} in G that is monotone in some direction l. (Namely, the order of the orthogonal projections of the vertices of P_{uw} on l is the same as the order they appear in P_{uw}.)

The problem of finding monotone drawing for trees has been studied in several recent papers. The main focus is to reduce the size of the drawing. Currently, the smallest drawing size is $O(n^{1.5}) \times O(n^{1.5})$. In this paper, we present a linear time algorithm for constructing monotone drawing of trees on a grid of size at most $O(n^{1.205}) \times O(n^{1.205})$. This is the first result achieving $o(n^3)$ drawing area for solving this problem.

1 Introduction

A *straight-line drawing* of a plane graph G is a drawing Γ in which each vertex of G is drawn as a distinct point on the plane and each edge of G is drawn as a line segment connecting two end vertices without any edge crossing. A path P in a straight-line drawing Γ is *monotone* if there exists a line l such that the orthogonal projections of the vertices of P on l appear along l in the order they appear in P. We call l a *monotone line* (or *monotone direction*) of P. Γ is called a *monotone drawing* of G if it contains at least one monotone path P_{uw} between every pair of vertices u, w of G. We call the monotone direction l_{uw} of P_{uw} the monotone direction for u, w.

The monotone drawings are introduced by Angelini et al. as a new visualization paradigm in [1]. Consider the example described in [1]: a traveler uses a road map to find a route from a town u to a town w. He would like to easily spot a path connecting u and w. This task is harder if each path from u to w on the map has legs moving away from u. The traveler rotates the map to better perceive its content. Hence, even if in the original map orientation all the paths from u to w have annoying back and forth legs, the traveler might be happy to find one map orientation where a path from u to w smoothly goes from left to right. This approach is also motivated by human subject experiments: it was shown the "geodesic tendency" (paths following a given direction) is important in understanding the structure of the underlying graphs [10].

X. He—Research supported in part by NSF Grant CCR-1319732.

The monotone drawing is also closely related to several other important graph drawing problems. In a monotone drawing, each monotone path is monotone with respect to a different line. In an *upward drawing* [6,7], every directed path is monotone with respect to the positive y direction. Even more related to the monotone drawings are the *greedy drawings* [2,12,13]. In a greedy drawing, for any two vertices u, v, there exists a path P_{uv} from u to v such that the Euclidean distance from an intermediate vertex of P_{uv} to the destination v decreases at each step. In a monotone drawing, for any two vertices u, v, there exists a path P_{uv} from u to v and a line l_{uv} such that the Euclidean distance from the projection of an intermediate vertex of P_{uv} on l to the projection of the destination v on l decreases at each step.

Related Works: Angelini et al. [1] showed that every tree of n vertices has a monotone drawing of size $O(n^2) \times O(n)$ (using a DFS-based algorithm), or $O(n^{\log_2 3}) \times O(n^{\log_2 3}) = O(n^{1.58}) \times O(n^{1.58})$ (using a BFS-based algorithm). It was also shown that every biconnected planar graph of n vertices has a monotone drawing in real coordinate space. Several papers have been published after [1]. The focus of the research is to identify the graph classes having monotone drawings and, if so, to find their monotone drawings with size as small as possible. It was shown in [3] that every planar graph has a monotone drawing of size $O(n) \times O(n^2)$. However, the drawing presented in [3] is not straight line. It may need up to $4n - 10$ bends in the drawing. Recently Hossain and Rahman showed that every planar graph has a monotone drawing of size $O(n^2) \times O(n)$ [9].

The monotone drawing problem for trees is particularly important. Any drawing result for trees can be applied to any connected graphs G: First we construct a spanning tree T for G, then find a monotone drawing Γ for T. Γ is automatically a monotone drawing for G (although not necessarily planar).

Both the DFS- and BFS-based tree drawing algorithms in [1] use the so-called Stern-Brocot tree to generate a set of $n - 1$ primitive vectors (will be defined later) in increasing order of slope. Then both algorithms do a post-order traversal of the input tree, and assign each edge a corresponding primitive vector as its slope. Such drawings of trees are called *slope-disjoint*. Kindermann et al. in [11] proposed another version of slope-disjoint algorithm, but using a different set of primitive vectors (based on *Farey sequence*), which slightly decreases the grid size to $O(n^{1.5}) \times O(n^{1.5})$. It remains an open problem to find a tree monotone drawing algorithm with area $o(n^3)$.

Our Results: In this paper, we show that every n-vertex tree admits a monotone drawing on a grid of size at most $O(n^{1.205)} \times O(n^{1.205})$. To the best knowledge of the authors, this is the first monotone drawing with $o(n^3)$ area for trees. Our drawings are slope-disjoint, and use Farey sequence to generate the primitive vectors as in [11]. However we use a set of more compact primitive vectors by introducing a *tree trimming* operation (see Lemma 2).

The paper is organized as follows. Section 2 introduces definitions and preliminary results on monotone drawings. In Section 3, we give our algorithm for constructing monotone drawing of trees. Section 4 concludes the paper and discusses future works.

2 Preliminaries

Let p be a point in the plane and l be a half-line with p as its starting point. The slope of l, denote by *slope(l)*, is the angle spanned by a ccw (we abbreviate the words "counterclockwise" and "clockwise" as ccw and cw respectively) rotation that brings the positive x-axis to overlap with l.

In this paper, we only consider *straight line drawings* (i.e. each edge of G is drawn as a straight line segment between its end vertices.) Let Γ be such a drawing of G and let $e = (u, w)$ be an edge of G. The *direction* of e, denoted by $d(u, w)$ or $d(e)$, is the half-line starting at u and passing through w. The slope of an edge (u, w), denoted by $\mathit{slope}(u, w)$, is the slope of $d(u, w)$. Observe that $\mathit{slope}(u, w) = \mathit{slope}(w, u) - 180°$. When comparing directions and their slopes, we assume that they are applied at the origin of the axes.

Let $P(u_1, u_k) = (u_1, \ldots, u_k)$ be a path of G. We also use $P(u_1, u_k)$ to denote the drawing of the path in Γ. $P(u_1, u_k)$ is *monotone with respect to a direction* l if the orthogonal projections of the vertices $u_1, \ldots, u_k$ on l appear in the same order as they appear on the path. $P(u_1, u_k)$ is *monotone* if it is monotone with respect to some direction. A drawing Γ is *monotone* if there exists a monotone path $P(u, w)$ for every pair of vertices u, w in G.

Let e_1 and e_2 be the edges in $P(u_1, u_k)$ with the smallest and the largest slope, respectively. e_1 and e_2 are called the *extremal edges* of $P(u_1, u_k)$. The closed interval $[\mathit{slope}(e_1), \mathit{slope}(e_2)]$ is called the *range* of $P(u_1, u_k)$ and denoted by $\mathit{range}(P(u_1, u_k))$. Note $\mathit{slope}(u_i, u_{i+1}) \in \mathit{range}(P(u_1, u_k))$ for all edges (u_i, u_{i+1}) $(i = 1, \ldots, k - 1)$ in $P(u_1, u_k)$.

The following property is proved in [1].

Property 1. A path $P(u_1, u_k)$ with range $[\mathit{slope}(e_1), \mathit{slope}(e_2)]$ is monotone if and only if $\mathit{slope}(e_2) - \mathit{slope}(e_1) < 180°$.

3 Monotone Drawings of Trees

Let T be a tree rooted at r. For each vertex u in T, $T(u)$ denotes the subtree of T rooted at u. $|T|$ denotes the number of vertices in T. T is called a *full tree* if every internal vertex (except the root r) of T has at least two children. The following fact is trivial.

Fact 1. *Let T be a full tree with n vertices. Let n_1 be the number of internal vertices and n_2 the number of leaves in T. Then (1) $n_1 \leq n_2$; and (2) $n = n_1 + n_2 \leq 2n_2$.*

The set of *primitive vectors of size d* is defined as:

$$P_d = \{(x, y) \mid x \text{ and } y \text{ are integers, } \gcd(x, y) \in \{1, d\}, 1 \leq x \leq y \leq d\}$$

If we consider each entry $(x, y) \in P_d$ to be the rational number y/x and order them by value, we get the so called *Farey sequence* $\mathcal{F}_d$ (see [8]). The property of

Farey sequence is well understood. It is known $|\mathcal{F}_d| = 3d^2/\pi^2 + O(d \log d)$ ([8], Thm 331). Thus $|P_d| = |\mathcal{F}_d| \geq 3d^2/\pi^2$. Let P'_d be the set of vectors that are the reflections of the vectors in P_d through the line $x = y$. Define:

$$\overline{P_d} = P_d \cup P'_d = \{(x, y) \mid x \text{ and } y \text{ are integers, } \gcd(x, y) \in \{1, d\}, 1 \leq x, y \leq d\}$$

Fig 1 (1) shows the vectors in $\overline{P_3}$. We will refer the members of P_d either as vectors or fractions. We have $|\overline{P_d}| \geq 6d^2/\pi^2$. Moreover, the members of $\overline{P_d}$ can be enumerated in $O(|\overline{P_d}|)$ time [11]. The *Stern-Brocot tree* is an infinite binary tree whose nodes are labeled by the rational numbers $\{y/x \mid (x, y) \in \cup_{d=1}^{\infty} \overline{P_d}\}$ in an elegant way [5,14].

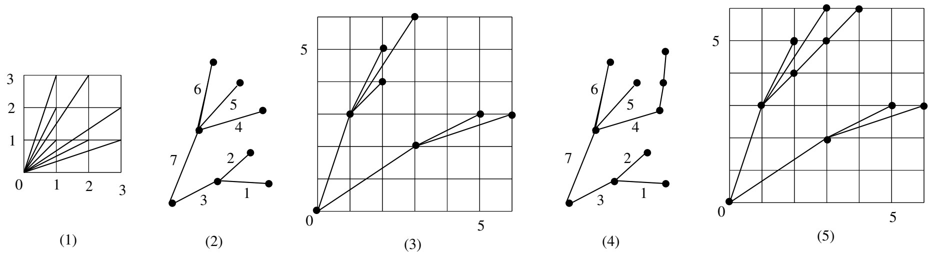


Fig. 1. Examples

Next, we outline the algorithm in [1] for monotone drawing of trees.

Definition 1. *[1] A* slope-disjoint drawing *of a tree T is such that:*

1. *For each vertex u in T, there exist two angles $\alpha_1(u)$ and $\alpha_2(u)$, with $0 < \alpha_1(u) < \alpha_2(u) < 180°$ such that, for every edge e that is either in $T(u)$ or that connects u with its parent, it holds that $\alpha_1(u) <$ slope$(e) < \alpha_2(u)$;*
2. *for any vertex u in T and a child v of u, it holds that $\alpha_1(u) < \alpha_1(v) < \alpha_2(v) < \alpha_2(u)$;*
3. *for every two vertices v_1, v_2 with the same parent, it holds that either $\alpha_1(v_1) < \alpha_2(v_1) < \alpha_1(v_2) < \alpha_2(v_2)$ or $\alpha_1(v_2) < \alpha_2(v_2) < \alpha_1(v_1) < \alpha_2(v_1)$.*

Based on Property 1, the following theorem was proved in [1].

Theorem 1. *Every slope-disjoint drawing of a tree is monotone.*

Remark 1. By Theorem 1, as long as the slopes of the edges in a drawing of a tree T guarantee the slope-disjoint property, one can *arbitrarily* assign lengths to the edges always obtaining a monotone drawing of T.

Algorithm 1 produces a monotone drawing for full trees. It was described in [1]. (But the presentation here is slightly modified).

Algorithm 1. Monotone-Full-Tree

```
Input: A full tree $T = (V, E)$ with $n$ vertices.
1. Take any set $\mathcal{V} = \{(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ of $n-1$ distinct primitive vectors,
   sorted by increasing $y_i/x_i$ value.
2. Assign the vectors in $\mathcal{V}$ to the edges of $T$ in ccw post-order. (Namely, recursively
   visit the subtrees of $T$ rooted at the children of the root $r$ in ccw order; then visit
   the root).
3. Draw the root $r$ of $T$ at the point $(0, 0)$. Then draw other vertices of $T$ in ccw
   pre-order as follows:
3.1 Let $w$ be the vertex to be drawn next; let $u$ be the parent of $w$ which has been
    drawn at a location $(x(u), y(u))$.
3.2 Let $(x_i, y_i)$ be the primitive vector assigned to the edge $(u, w)$ in step 2. Draw $w$
    at the location $(x(w), y(w))$ where $x(w) = x(u) + x_i$ and $y(w) = y(u) + y_i$.
```

Fig 1 (2) shows a full tree T. The numbers next to the edges indicate the order they are assigned the vectors in $\mathcal{V} = \overline{P_3}$. Fig 1 (3) shows the drawing of T produced by Algorithm 1.

It was shown in [1] that the drawing obtained in Algorithm 1 is slope-disjoint and hence monotone. Two versions of Algorithm 1 were given in [1]. Both use the Stern-Brocot tree $\mathcal{T}$ to generate the vector set $\mathcal{V}$ needed in step 1. The BFS version of the algorithm collects the vectors from $\mathcal{T}$ in a breath-first-search fashion. This leads to a drawing of size $O(n^{\log_2 3}) \times O(n^{\log_2 3}) = O(n^{1.58}) \times O(n^{1.58})$. The DFS version of the algorithm collects the vectors from $\mathcal{T}$ in a depth-first-search fashion. This leads to a drawing of size $O(n) \times O(n^2)$. The algorithm in [11] for finding monotone drawing of trees is essentially another version of Algorithm 1. It uses the vectors in $\overline{P_d}$ (with $d = 4\sqrt{n}$) for the set $\mathcal{V}$ in step 1. This leads to a monotone drawing of size $O(n^{1.5}) \times O(n^{1.5})$.

Next, we modify Algorithm 1 so that it can handle general (not necessarily full) trees. Let T be such a tree. Let w be an internal vertex in T with just one child. In a monotone drawing of T, we draw the two edges incident to w with the same slope (so the two angles at either side of w are 180°). This motivates the following definition.

Definition 2. *A* branch *of T is a path $P = (u_1, \ldots, u_k)$ ($k \geq 3$) in T such that:*

- *u_i is the child of u_{i+1} for $1 \leq i \leq k-1$;*
- *For each $i = 2, \ldots, k-1$, u_i has exactly one child;*
- *u_1 is either a leaf, or has at least two children;*
- *u_k is either the root, or has at least two children.*

Definition 3. *Let T be a tree. By replacing each branch $P = (u_1, \ldots, u_k)$ of T with a single edge (u_1, u_k), we obtain a full tree, which is called the* skeleton tree *of T and denoted by T^s.*

The tree in Fig 1 (2) is the skeleton tree of the tree shown in Fig 1 (4).

Algorithm 2 finds a monotone drawing of a general tree T.

```
Algorithm 2. Monotone-General-Tree
Input: A (general) tree $T = (V, E)$ with $n$ vertices.
0. Construct the skeleton tree $T^s$ of $T$. Let $n^s$ be the number of vertices in $T^s$.
1. Take any set $\mathcal{V}^s = \{(x_1, y_1), \dots, (x_{n^s-1}, y_{n^s-1})\}$ of $n^s - 1$ distinct primitive vectors,
   sorted by increasing $y_i/x_i$ value.
2. Assign the vectors in $\mathcal{V}^s$ to the edges of $T^s$ in ccw post order.
3.0. Let $(u, w)$ be an edge in $T^s$ corresponding to a branch $P(u_1, u_k) = (u = u_1, u_2, \dots, u_k = w)$ in $T$.
   Let $(x, y)$ be the vector assigned to the edge $(u, w)$ in step 2. Assign the vector $(x, y)$ to all edges
   $(u_i, u_{i+1})$ $(1 \le i \le k-1)$ in $P(u_1, u_k)$. Do the same for every branch in $T$.
   (Every edge in $T$ is assigned a vector in $\mathcal{V}^s$ by now).
3. Draw the vertices of $T$ as in step 3 of Algorithm 1.
```

Fig 1 (5) shows the drawing of the tree in Fig 1 (4) produced by Algorithm 2. By Remark 1, it can be shown Algorithm 2 produces a monotone drawing of a general tree T.

Let $\alpha = \frac{1+\sqrt{5}}{2} \simeq 1.618$ and $\bar{\alpha} = \frac{1-\sqrt{5}}{2} \simeq -0.618$ be the two roots of the equation $x^2 - x - 1 = 0$. Let $\beta = \log_2 \alpha \simeq 0.694$.

Let $\{F_1 = 1, F_2 = 2, F_3 = 3, F_4 = 5, \dots, F_k = F_{k-1} + F_{k-2}, \dots\}$ be the Fibonacci sequence. It is known that $F_k = \frac{\alpha^{k+1}}{\sqrt{5}} - \frac{\bar{\alpha}^{k+1}}{\sqrt{5}}$. It is easy to show $F_k \le \frac{\alpha^{k+1.5}}{\sqrt{5}}$ for all $k \ge 1$.

Let d and D be two positive integers with $d < D$. (Their values will be determined later). The vectors in $\overline{P_d}$ are called *short vectors*. The vectors in $\overline{P_D} - \overline{P_d}$ are called *long vectors*. The idea of our algorithm is as follows: First we "trim" the input tree T by deleting some subtrees from T such that the height of each deleted subtree is small. The resulting tree T_1, with reduced size, can be drawn by using short vectors. Then we draw the deleted subtrees by using long vectors. Because the heights of the deleted subtrees are small, this will not increase the drawing size too much. To make this idea work, we must make sure there are enough long vectors between any two consecutive short vectors (ordered by their slopes), as shown in the following:

Lemma 1. *Let $0 < \delta < 1$ and $\lambda > 0$ be two constants. Define $\Delta = \delta + \lambda\beta$, $d = 2n^\delta$ and $D = 3n^\Delta$. Let y_1/x_1 and y_2/x_2 be any two consecutive short vectors in $\overline{P_d}$ with $y_1/x_1 < y_2/x_2$. Then there exist at least $n^\lambda - 1$ long vectors y/x in $\overline{P_D} - \overline{P_d}$ such that $y_1/x_1 < y/x < y_2/x_2$.*

Proof. Since y_1/x_1 and y_2/x_2 are consecutive fractions in $\overline{P_d}$, we have $y_2x_1 - y_1x_2 = 1$ ([8], Theorem 28) and $x_1 + x_2 > d$ ([8], Theorem 30).

Define an operator $\odot$ of two fractions as follows:

$$\frac{y_1}{x_1} \odot \frac{y_2}{x_2} = \frac{y_1 + y_2}{x_1 + x_2}$$

Let $y_3/x_3 = \frac{y_1}{x_1} \odot \frac{y_2}{x_2}$. It is easy to show y_3/x_3 is a fraction strictly between y_1/x_1 and y_2/x_2. Similarly, let $y_4/x_4 = y_1/x_1 \odot y_3/x_3$ and $y_5/x_5 = y_3/x_3 \odot y_2/x_2$, we will have three fractions $y_4/x_4 < y_3/x_3 < y_5/x_5$ strictly between y_1/x_1 and y_2/x_2.

Repeat this process, we can generate all fractions between y_1/x_1 and y_2/x_2 in the form of a binary tree, called the Stern-Brocot tree for y_1/x_1 and y_2/x_2, denoted by $\mathcal{T}(y_1/x_1, y_2/x_2)$, as follows. (The original Stern-Brocot tree defined in [5,14] is for the fractions $y_1/x_1 = 0/1$ and $y_2/x_2 = 1/0$).

$\mathcal{T}(y_1/x_1, y_2/x_2)$ has two nodes y_1/x_1 and y_2/x_2 in level 0. The level 1 contains a single node r labeled by the fraction $y_3/x_3 = y_1/x_1 \odot y_2/x_2$, which is the right child of y_1/x_1, and is the left child of y_2/x_2. An infinite ordered binary tree rooted at y_3/x_3 is constructed as follows. Consider a node y/x of the tree. The left child of y/x is $y/x \odot y'/x'$ where y'/x' is the ancestor of y/x that is closest to y/x (in terms of graph-theoretical distance in $\mathcal{T}(y_1/x_1, y_2/x_2)$) and that has y/x in its right subtree. The right child of y/x is $y/x \odot y''/x''$ where y''/x'' is the ancestor of y/x that is closest to y/x and that has y/x in its left subtree. (Fig 2 shows a portion of the Stern-Brocot tree $\mathcal{T}(4/5, 5/6)$. The leftmost column indicate the level numbers).

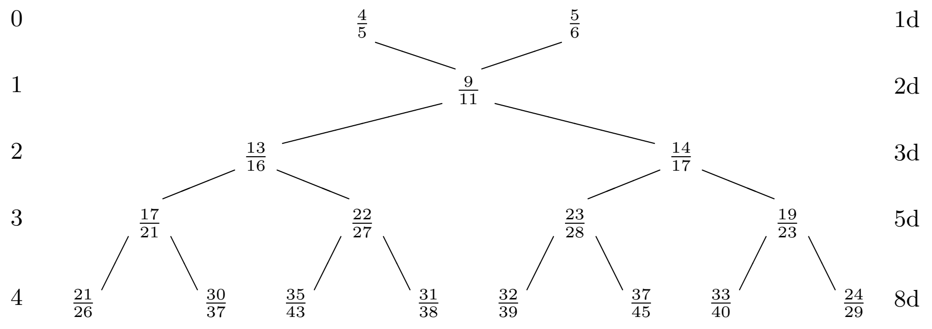


Fig. 2. The first 5 levels of the Stern-Brocot tree $\mathcal{T}(4/5, 5/6)$

All fractions in $\mathcal{T}(y_1/x_1, y_2/x_2)$ are distinct and strictly between y_1/x_1 and y_2/x_2. The following facts are either directly from the definition of $\mathcal{T}(y_1/x_1, y_2/x_2)$ or can be shown by easy induction:

1. Each node at level k is the result of the operator $\odot$ applied to a node in level $k-1$ and a node in level $\leq k-2$.
2. For each level k, there exists a node that is the result of the operator $\odot$ applied to a node in level $k-1$ and a node in level $k-2$.
3. Let $\mathcal{V}_k$ be the set of the fractions contained in $\mathcal{T}(y_1/x_1, y_2/x_2)$ from level 1 through level k. Then $|\mathcal{V}_k| = 2^k - 1$.
4. For each node y/x with the left child y'/x' and the right child y''/x'', we have $y'/x' < y/x < y''/x''$. So the in-order traversal of $\mathcal{T}(y_1/x_1, y_2/x_2)$ lists the fractions in $\mathcal{V}_k$ in increasing order.

5. Define the *size* of a node y/x to be $\max\{x, y\}$. The size of the nodes in level 0 (i.e. the two nodes y_1/x_1 and y_2/x_2) is bounded by $1 \cdot d = F_1 \cdot d$ (because both y_1/x_1 and y_2/x_2 are fractions in $\overline{P_d}$).
6. The size of the node in level 1 (i.e. the node y_3/x_3) is bounded by $2 \cdot d = F_2 \cdot d$ (because $x_3 = x_1 + x_2 \leq 2d$ and $y_3 = y_1 + y_2 \leq 2d$).
7. For each $q \geq 2$, the size of level q nodes is bounded by $F_{q+1} \cdot d$. (The last column in Fig 2 show the upper bounds of the size of the level q fractions.)

Set $k = \lambda \log_2 n$. Then $|\mathcal{V}_k| = 2^{\lambda \log_2 n} - 1 = n^\lambda - 1$.
The size of any fraction in $\mathcal{V}_k$ is bounded by:

$$F_{k+1} \cdot d \leq \frac{\alpha^{k+2.5}}{\sqrt{5}} \cdot d = \frac{\alpha^{2.5}}{\sqrt{5}} \alpha^{\lambda \log_2 n} \cdot d = \frac{\alpha^{2.5}}{\sqrt{5}} n^{\lambda \log_2 \alpha} \cdot 2n^\delta < 3n^{\delta + \lambda\beta} = D$$

Since there are no short vectors between y_1/x_1 and y_2/x_2, all members in $\mathcal{V}_k$ are long vectors. Thus the lemma holds. □

Next we describe how to "trim" a tree T with n vertices. To simplify notation, we use $L = \{1, 2, \ldots, p\}$ to denote the set of the leaves in T ordered from left to right. Let P_k ($1 \leq k \leq p$) be the path in T from the leaf k to the root r. $T_{|k}$ denotes the subtree of T consisting of the vertices to the left of P_k (including the vertices in P_k).

For each pair i, j ($1 \leq i < j \leq p$), $P(i, j)$ denotes the path in T from the leaf i to the leaf j. When walking along $P(i, j)$ from i to j, we encounter some subtrees of T on the right side of $P(i, j)$. Denote by $\mathcal{D}(i, j)$ the set of these subtrees. For notation convenience, $\mathcal{D}(0, k)$ denotes the set of the subtrees of T located to the left of the path P_k, and $\mathcal{D}(k, p+1)$ denotes the set of the subtrees of T located to the right of the path P_k.

Let T_1 be a subtree obtained from T by deleting some subtrees from T. T_1 is called a *trimmed subtree* of T if every leaf in T_1 is also a leaf in T. Consider a trimmed subtree T_1 of T. Let $L_1 = \{i_1, \ldots, i_q\} \subseteq L$ be the set of the leaves in T_1. For each k ($0 \leq k \leq q$), the subtrees in $\mathcal{D}(i_k, i_{k+1})$ are deleted from T when constructing T_1. (Here $\mathcal{D}(i_0, i_1) = \mathcal{D}(0, i_1)$ and $\mathcal{D}(i_q, i_{q+1}) = \mathcal{D}(i_q, p+1)$.) The subtrees in $\mathcal{D}(i_k, i_{k+1})$ are called the *kth trimmed set* of T with respect to T_1.

Definition 4. *Let $0 < \gamma < 1$ be a constant. A trimmed subtree T_1 of T is* γ-well-trimmed *if the following conditions hold:*

1. *T_1 has $\leq n^\gamma$ leaves.*
2. *Every trimmed set of T with respect to T_1 contains $< n^{1-\gamma}$ vertices.*

Lemma 2. [Trimming Lemma]: *Every tree T has a γ-well-trimmed subtree T_1 for any $0 < \gamma < 1$.*

Proof. Let $L = \{1, 2, \ldots, p\}$ be the set of the leaves in T ordered from left to right. If $p \leq n^\gamma$, then T itself is a γ-well-trimmed subtree and we are done. Now suppose $p > n^\gamma$. We need to select a subset $L_1 = \{i_1, \ldots, i_q\} \subseteq L$ ($1 \leq i_1 < i_2 < \ldots < i_q \leq p$) as the leaf set of T_1, and trim all leaves in $L - L_1$ from T.

Temporarily add a dummy child 0 of the root as the left most leaf of T. Initially, set $i_0 = 0$. Suppose that we have selected the leaf i_k. We describe how to select the next leaf $i_{k+1} \in L_1$. Let $t > i_k$ be the leaf in L with the smallest index such that (see Fig 3 (1)):

$$|T_{|t}| - |T_{|i_k}| \geq n^{1-\gamma} \tag{1}$$

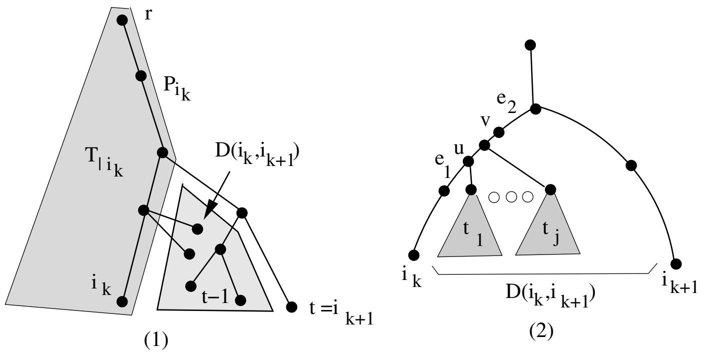


Fig. 3. Examples

We set $i_{k+1} = t$, add i_{k+1} into L_1 and delete all subtrees in $\mathcal{D}(i_k, i_{k+1})$ from T. If no such leaf t exists, then i_k is the last leaf selected in L_1. We delete all subtrees in $\mathcal{D}(i_k, p+1)$ from T. The process stops and the resulting subtree is T_1. By the choice of t, we have:

$$|T_{|t-1}| - |T_{|i_k}| < n^{1-\gamma} \tag{2}$$

Clearly $\mathcal{D}(i_k, i_{k+1}) \subseteq T_{|t-1} - T_{|i_k}$ (see Fig 3 (1)). This implies: $|\mathcal{D}(i_k, i_{k+1})| \leq |T_{|t-1}| - |T_{|i_k}| < n^{1-\gamma}$.

By using (1) repeatedly, we have: $|T_{|i_k}| \geq k \cdot n^{1-\gamma}$. Let i_q be the last selected leaf in L_1, we have:

$$n \geq |T_{|i_q}| \geq q \cdot n^{1-\gamma}$$

This implies $q \leq n^{\gamma}$. Thus T_1 is a γ-well-trimmed subtree of T. □

Now we can describe Algorithm 3 for constructing compact monotone drawing of trees.

Theorem 2. *Algorithm 3 produces a monotone drawing of size at most $O(n^l) \times O(n^l)$ where $l = \max\{1 + \delta, \delta + (1 - 2\delta)(1 + \beta)\}$.*

Proof. We analyze the drawing size step by step.

Step 3: Since T_1^s has at most $n^{2\delta}$ leaves, $|T_1^s| \leq 2n^{2\delta}$ by Fact 1. So T_1^s has at most $2n^{2\delta} - 1$ edges. The set $\overline{P_d}$ contains at least $6 \cdot (2n^{\delta})^2/\pi^2 \geq 2n^{2\delta}$ primary

Algorithm 3. δ-Monotone-Drawing

```
Input: A tree $T = (V, E)$ with $n$ vertices, and a constant $0 < \delta < 1/2$.
1. Construct a $2\delta$-well-trimmed subtree $T_1$ of $T$.
2. Construct the skeleton subtree $T_1^s$ of $T_1$.
3. Let $\mathcal{V}$ be the set of short primary vectors in $\overline{P_d}$ (where $d = 2n^\delta$).
4. Draw $T_1^s$ by using the vectors in $\mathcal{V}$ as in Algorithm 1.
5. Modify the drawing of $T_1^s$ to the drawing of $T_1$ as in Algorithm 2, still using vectors
   in $\mathcal{V}$.
6. Let $i_1, \dots, i_q$ be the leaves of $T_1$. For each $k$ $(0 \le k \le q)$, draw the vertices in the
   $k$th trimmed set $\mathcal{D}(i_k, i_{k+1})$, using the long vectors in $\overline{P_D} - \overline{P_d}$ where $D = 3n^\Delta$
   with $\Delta = \delta + (1 - 2\delta)\beta$
```

vectors. Thus, there are enough vectors in $\mathcal{V}$ in step 3 to complete the drawing in step 4.

Step 5: The length of the longest path in T_1 is at most $n - 1$. Since we use only short vectors (the length of whose projection on x- and y-axis is at most $2n^\delta$), the size of the drawing of T_1 obtained in step 5 is at most $2n^{1+\delta} \times 2n^{1+\delta}$.

Step 6: We need to describe the drawing of the subtrees in each $\mathcal{D}(i_k, i_{k+1})$. Consider any pair of edges e_1 and e_2 on $P(i_k, i_{k+1})$. Suppose that when walking from the leaf i_k to the leaf i_{k+1} along the path $P(i_i, i_{k+1})$, we pass e_1 before e_2. Let u be the second end vertex of e_1 and v be the first end vertex of e_2 we encounter. Consider each edge pair e_1 and e_2 such that the following hold (see Fig 3 (2)):

- e_1 and e_2 are assigned distinct consecutive vectors in $\mathcal{V}$;
- some subtrees in $\mathcal{D}(i_k, i_{k+1})$, (say $t_1, \dots t_j$), are attached to a vertex in $P(i_k, i_{k+1})$ between u and v. (Namely the root of each t_i $(1 \le i \le j)$ is a child of a vertex in $P(i_k, i_{k+1})$ that is between u and v, inclusive).

By Definition 4, each $\mathcal{D}(i, j)$ has less than $n^{1-2\delta}$ vertices. So there are $< n^{1-2\delta}$ vertices and edges in $\cup_{i=1}^{j} t_i$. Let (x_1, y_1) and (x_2, y_2) be the short vectors assigned to e_1 and e_2 respectively. Let $\Delta = \delta + (1 - 2\delta)\beta$ and $D = 3n^\Delta$. By Lemma 1, there are at least $n^{1-2\delta} - 1$ long vectors in $\overline{P_D} - \overline{P_d}$ whose slopes are strictly between y_1/x_1 and y_2/x_2. So we can assign these long primary vectors to the edges in $\cup_{i=1}^{j} t_i$ in ccw post-order. (If t_i contains branches, the edges in the same branch are assigned the same vector as in Algorithm 2). Such assignment guarantees the slope-disjoint property. All subtrees in $\mathcal{D}(i_k, i_{k+1})$ are drawn this way. Thus the drawing of T obtained in step 6 is monotone by Theorem 1.

The length of the longest path in each t_i $(1 \le i \le j)$ is at most $n^{1-2\delta}$. Note that the x- and y-projections of the long vectors used to draw the edges in t_i are bounded by $D = 3n^{\delta+(1-2\delta)\beta}$. So the drawing of each t_i adds at most $3n^{\delta+(1-2\delta)\beta+(1-2\delta)}$ to the x- and y-directions of the drawing of T_1. Thus the size of the drawing for T is at most $O(n^l) \times O(n^l)$ where $l = \max\{1 + \delta, \delta + (1 - 2\delta)(1 + \beta)\}$. □

For example, when $\delta = 1/4$, we have $1 + \delta > \delta + (1 - 2\delta)(1 + \beta)$. So Algorithm 3 produces a drawing of size $O(n^{5/4}) \times O(n^{5/4})$.

Lemma 3. *Algorithm 3 takes $O(n)$ time.*

Proof. Steps 2 - 5 of Algorithm 3 are modifications of Algorithm 1 and 2. By results in [1,11], these two algorithms take $O(n)$ time. The implementation of Step 6 is straightforward. So we just need to describe the details of Step 1: how to find a well-trimmed subtree T_1 of T.

Let $L = \{1, \ldots p\}$ be the leaf set of T. We need to identify a subset $L_1 = \{i_1, \ldots, i_q\} \subseteq L$ of leaves in T_1.

For each leaf i $(1 \leq i \leq p)$ of T, let v_i be the lowest common ancestor of the leaf $i - 1$ and the leaf i in T. Let p_i be the path from i to v_i. It is easy to see:

$$|T_{|i}| - |T_{|i-1}| = |p_i| - 1$$

By performing a ccw post-order traversal on T, we can easily calculate $|p_i|$ for each leaf $i \in L$ in linear time. Recall that the leaf i_{k+1} is the smallest index such that

$$|T_{|i_{k+1}}| - |T_{|i_k}| \geq n^{1-2\delta}$$

Thus, knowing $|p_i|$ for each leaf $i \in L$, we can identify the leaf set $L_1 = \{i_1, \ldots, i_q\}$ of T_1 in linear time. $\square$

When $\delta = \frac{\beta}{2(1+\beta)} \simeq 0.205$, we have $1 + \delta = \delta + (1 - 2\delta)(1 + \beta)$. Then $l = \max\{1 + \delta, \delta + (1 - 2\delta)(1 + \beta)\} = 1 + \delta = 1 + \frac{\beta}{2(1+\beta)}$. From Theorem 2 and Lemma 3, we obtain our main result:

Theorem 3. *Every tree T with n vertices has a monotone drawing of size at most $O(n^l) \times O(n^l)$, where $l = 1 + \frac{\beta}{2(1+\beta)} \simeq 1.205$. The drawing can be constructed in $O(n)$ time.*

4 Conclusion

In this paper, we showed that any n-vertex tree has a monotone drawing on an $O(n^{1.205}) \times O(n^{1.205})$ grid, and such drawing can be constructed in $O(n)$ time.

Finding a monotone drawing of trees with even smaller grid size appears to be an interesting challenge. We mention a possible way for improving our result:

The 2δ-well-trimmed subtree T_1 has at most $n^{2\delta}$ leaves. In the analysis, we assume the worst case that the length of the longest path in T_1 is $n - 1$. In this case, there are many long branches in T_1. Is it possible to draw such a special tree (n vertices, but at most $n^{2\delta}$ leaves) on a grid smaller than $O(n^{1+\delta}) \times O(n^{1+\delta})$? A positive answer will reduce the drawing size of general trees.

References

1. Angelini, P., Colasante, E., Di Battista, G., Frati, F., Patrignani, M.: Monotone Drawings of Graphs. Journal of Graph Algorithms and Applications **16**(1), 5–35 (2012)
2. Angelini, P., Frati, F., Grilli, L.: An Algorithm to Construct Greedy Drawings of Triangulations. Journal of Graph Algorithms and Applications **14**(1), 19–51 (2010)
3. Angelini, P., Didimo, W., Kobourov, S., Mchedlidze, T., Roselli, V., Symvonis, A., Wismath, S.: Monotone Drawings of Graphs with Fixed Embedding. Algorithmica (2013). doi:10.1007/s00453-013-9790-3
4. Arkin, E.M., Connelly, R., Mitchell, J.S.: On Monotone Paths among Obstacles with Applications to Planning Assemblies. In: SoCG 1989, pp. 334–343 (1989)
5. Brocot, A.: Calcul des Rouages par Approximation. Nouvelle Methode, Revue Chronometrique **6**, 186–194 (1860)
6. Di Battista, G., Tamassia, R.: Algorithms for Plane Representations of Acyclic Digraphs. Theor. Comput. Sci. **61**, 175–198 (1988)
7. Garg, A., Tammassia, R.: On the Computational Complexity of Upward and Rectilinear Planarity Testing. SIAM J. Comp. **31**(2), 601–625 (2001)
8. Hardy, G., Write, E.M.: An Introduction to the Theory of Numbers, 5th edn. Oxford University Press (1989)
9. Hossain, M.I., Rahman, M.S.: Monotone Grid Drawings of Planar Graphs. In: Chen, J., Hopcroft, J.E., Wang, J. (eds.) FAW 2014. LNCS, vol. 8497, pp. 105–116. Springer, Heidelberg (2014)
10. Huang, W., Eades, P., Hong, S.-H.: A Graph Reading Behavior: Geodesic-Path Tendency. In Proceedings of IEEE Pacific Visualization Symposium, pp. 137–144 (2009)
11. Kindermann, P., Schulz, A., Spoerhase, J., Wolff, A.: On Monotone Drawings of Trees. In: Duncan, C., Symvonis, A. (eds.) GD 2014. LNCS, vol. 8871, pp. 488–500. Springer, Heidelberg (2014)
12. Moitra, A., Leighton, T.: Some Results on Greedy Embeddings in Metric Spaces. In: Proceedings FOCS 2008, pp. 337–346 (2008)
13. Papadimitriou, C.H., Ratajczak, D.: On a Conjecture Related to Geometric Routing. Theoretical Computer Science **344**(1), 3–14 (2005)
14. Stern, M.A.: Ueber eine Zahlentheoretische Funktion. Journal fur die reine und angewandte Mathematik **55**, 193–220 (1958)

A Measure and Conquer Approach for the Parameterized Bounded Degree-One Vertex Deletion

Bang Ye Wu(✉)

National Chung Cheng University, Chiayi 621, Taiwan, R.O.C
bangye@cs.ccu.edu.tw

Abstract. Measure & Conquer is an approach helpful for designing branching algorithms. A key point in the approach is how to design the measure. Given a graph $G = (V, E)$ and an integer k, the Bounded Degree-one Deletion problem asks for if there exists a subset D of at most k vertices such that the degree of any vertex in $G[V \setminus D]$ is upper bounded by one. Combining the parameter with a potential as the measure in Measure & Conquer, where the potential is a lower bound of the decrement of the parameter, we design a branching algorithm running in polynomial space and $O(1.882^k + |V||E|)$ time, which improves the current best parameterized complexity $O^*(2^k)$ of the problem.

Keywords: Parameterized algorithm · Measure and conquer · Branch and reduce · Bounded degree-one deletion · Vertex cover P_3

1 Introduction

Designing parameterized and exact exponential algorithms is an important issue to cope with NP-hard problems and attracts many researchers to devote themselves to this area [6,7,10]. *Branch & Reduce* is one of the major techniques in designing such algorithms, in which one applies some reduction rules, branches the problem into two or more subproblems, and then solves the subproblems recursively. For a branching algorithm, its time complexity is usually determined by a set of recurrences which come from the branching rules. To analyze the time complexity, a traditional method finds the largest root (*branching factor*) of each recurrence and then obtains an upper bound of the worst-case.

As an example, consider the well-known parameterized Vertex Cover, in which one looks for a vertex subset of size at most k covering all the edges of the input graph, i.e., the removal of a vertex cover leaves an independent set. A clear and crucial observation is that if we determine not to select a vertex v into a solution, then all its neighbors must be selected. This branch rule gives us a recurrence $T(k) = T(k-1) + T(k-d(v))$, where $T(k)$ is the parameterized complexity and $d(v)$ is the degree of v. Now consider the following similar but more difficult problem, and it is the problem we consider in this paper. By $G[V \setminus D]$, we denote the subgraph induced by $V \setminus D$.

Bounded-Degree-one Deletion (1-BDD)
Instance: A graph $G = (V, E)$ and a positive integer k.
Question: Does there exist $D \subseteq V$ of size at most k such that the maximum degree of $G[V \setminus D]$ is at most 1?

If one uses the above branching rule on this problem, then a difficulty arises: Since we cannot immediately determine which of the neighbors should be selected, there is no decrement on the parameter. However, once a vertex is determined not to be selected, we indeed make some progress. The difficulty comes from that it is hard to reflect the progress by using the traditional analysis method. Although usually in such a case one may still design a branching algorithm by exploring more detailed local structure or consider two or more branching stages, it leads to a minute and complicated algorithm.

Measure & Conquer [11] is an approach for designing branching algorithms and can be used to cope with such situations. In addition to balancing a set of recurrences to get a better bound, one of the key points in Measure & Conquer is to design a *measure* to reflect the progress. A potential function estimates a lower bound of the decrement of the parameter. In this paper we demonstrate that using a potential function and the Measure & Conquer approach is helpful for designing parameterized algorithms. For the 1-BDD problem, once we decide not to select a vertex v, at most one of its neighbors can be left, and therefore we can increase the potential by $d(v) - 1$. With the intention of utilizing the potential and introducing the potential as an additional parameter, it is easier to design an efficient Branch & Reduce algorithm, which is the main result we show in this paper.

Related Work. In the literature, the 1-BDD problem is also known as the Vertex Cover P_3 problem, since a 1-BDD set covers all P_3 (path of three vertices), i.e., for each P_3 in the input graph, at least one of the three vertices is chosen. Both Vertex Cover and 1-BDD problem are special cases of the δ-BDD problem, in which the maximum degree of the remaining graph must be at most δ. Since a 1-BDD set cover all P_3 in the graph, by the kernelization results for 3-Hitting Set [1], it is easy to show that the 1-BDD problem is fixed-parameter tractable. Chen *et al.* [12] gave a kernelization algorithm to obtain a kernel of size $6k$. Several fixed-parameter algorithms were developed for the δ-BDD problem [9,13,15]. For 1-BDD, Moser *et. al.* [14] gave an algorithm with time complexity $O(2.31^k + kn)$ based on bounded search trees and developed an $O(2^k \cdot k^2 + kn)$-time algorithm by the *iterated-compression* technique. An $O^*(2^k)$-time algorithm was also given with the name Vertex Cover P_3 [17]. Currently the best parameterized complexity with solution size as the parameter is $O^*(2^k)$.

There are also some results of exact algorithms and approximation algorithms in the literature. Kardoš *et al.* [8] gave an exact algorithm for the Vertex Cover P_3 problem running in time $O^*(1.5171^n)$, and Chang *et al.* improved the complexity to $O^*(1.4658^n)$ [3]. A randomized approximation was given with an expected approximation ratio 23/11 for Vertex Cover P_3 [8].

A 2-approximation algorithm by using linear programming approaches was given by Tu and Zhou [19]. Later, they also gave a greedy algorithm and showed that it is a 2-approximation algorithm [18].

Betzler *et al.* [2] showed that when parameterized by the treewidth of the input graph and δ is unbounded, δ-BDD is W[1]-hard. In the same paper, they showed that it becomes fixed-parameter tractable with respect to the combined parameters of the treewidth and the solution size.

New Results. Instead of solving only the 1-BDD problem, we consider the following more general problem.

> Red-White BDD (RW-BDD) problem
> Instance: A graph $G = (V, E)$ and an integer $k \geq 0$, where $V = R \cup W$ is partitioned into a red subset R and a white subset W.
> Question: Is there a vertex subset D with size at most k such that in $G[V \setminus D]$ each red vertex has degree 0 and each white vertex has degree at most one?

When all vertices are red, Red-White BDD is the same as Vertex Cover; and if $V = W$, it degenerates to the 1-BDD problem. The intuition of transforming the 1-BDD problem to Red-White BDD is that we color a vertex red when one of its neighbors is decided not to be selected into the 1-BDD set. In this paper, we design reduction and branching rules and obtain an algorithm with polynomial space and time complexity $O^*(1.882^k)$ for the Red-White BDD problem, and thus improve the current best result $O^*(2^k)$ for the 1-BDD problem.

Organization. The remaining paragraphs are organized as follows. We introduce some notation in Section 2. In Section 3, we show some properties used in the algorithm, and the branching algorithm is in Section 4. Finally concluding remarks are given in Section 5.

2 Preliminaries

The following notation will be used. For two sets X and Y, the difference is denoted by $X \setminus Y$. Let $G = (V, E)$ be a graph with vertex set V and edge set E. In the remaining paragraphs, $G = (V, E)$ is always the input graph. Each vertex is either red or white, and let R and W denote the sets of red and white vertices, respectively. A red/white path is a path consisting of only red/white vertices. The similar definitions are used for red/white edges, cycles, components and cliques.

For $S \subseteq V$, $G[S]$ denotes the subgraph induced by S. A vertex subset D is a *red-white bounded degree deletion* set, RW-BDD for short, if the degree of any red vertex in $G[V \setminus D]$ is 0 and the degree of any white vertex in $G[V \setminus D]$ is at most one. That is, after removing a RW-BDD set, the remaining subgraph is a collection of isolated vertices and white edges.

For $v \in V$, let $N[v]$ and $N(v)$ denote the closed and open neighborhoods of v in G, respectively, i.e., $N(v) = \{u \mid (u, v) \in E\}$ and $N[v] = \{v\} \cup N_G(v)$. For a vertex subset U, let $N[U] = \bigcup_{v \in U} N_G[v]$ and $N(U) = N[U] \setminus U$. Let $d(v) = |N(v)|$ denote the degree of v. For a vertex set S, $N_S(v) = N(v) \cap S$ and $d_S(v) = |N_S(v)|$. Particularly $N_R(v)$ and $N_W(v)$ denote the red and white neighborhoods of v, respectively. Thus, $d_R(v)$ and $d_W(v)$ are the numbers of red and white neighbors, respectively.

As in the literature, we define a parameterized problem as a decision problem. A branching algorithm, also known as search-tree algorithm, solves a problem by branching into two or more cases and solving the subproblems recursively until some terminal cases. Reduction rules may be used in the recursive procedure to reduce the instance. A reduction rule must be sound. That is, it does not change the yes/no-answer.

Since an RW-BDD set must cover all white P_3 and all edges with one or two red endpoints in the graph, by the kernelization results for 3-Hitting Set [1], a kernel of $O(k^2)$ vertices can be obtained in $O(mn)$ time, where n and m denote the numbers of vertices and edges of the input graph, respectively.

Let $\mathcal{P}(I, k)$ be a parameterized problem, where I and k are the instance and the parameter, respectively. A function f mapping instances to nonnegative real numbers is a *potential function* if $f(I)$ is a lower bound of the parameter for instance I, i.e., the answer of $\mathcal{P}(I, k)$ is no whenever $k < f(I)$.

3 Some Observations

While recursively searching the solution, some vertices are *selected*, and some are *discarded*. By "selecting a vertex", we mean to put it into the RW-BDD set. By "discarding a vertex", we mean that we decide to exclude the vertex from the RW-BDD set. Any vertex which has been selected or discarded is removed from the instance. A white vertex is colored red when one of its neighbors is discarded. In the following, we list some properties. Since they can be easily obtained by definitions or come from previous results for Vertex Cover, we omit the proofs.

- When a red vertex v is discarded, all vertices in $N(v)$ must be selected, i.e., put into the RW-BDD set.
- When a white vertex v is discarded, all its red neighbors must be selected and at most one of its white neighbors can be not selected. Let G' be the graph constructed from G by removing $N_R[v]$ and making $N_W(v)$ a red clique. Then, (G, k) has a solution discarding v if and only if $(G', k - d_R(v))$ has a solution.
- For $v \in W$ and $d_W(v) = 0$, we can color v red without changing the yes/no answer.
- (Red Deg-1 rule) If $v \in R$ and $N(v) = \{u\}$, then there is an optimal solution selecting u and discarding v.

- (White Deg-1 rule) If $v \in W$ and $N(v) = \{u\}$, then there is an optimal solution discarding v. Therefore we can have the two subcases: If $u \in R$, then we select u and discard v. If $u \in W$, then we color u red and discard v.
- Solving the RW-BDD problem on an isolated red component is equivalent to solving the VERTEX COVER on it.
- For an isolated white cycle, a minimum RW-BDD set can be found in polynomial time.
- (Red Deg-2 folding rule, [4]). Suppose that $N(v) = \{u, w\}$ and v, u, w are all red. If u and w are adjacent, then there is an optimal solution selecting $\{u, w\}$ and discarding v. If u and w are not adjacent, we can merge u and w, discard v, and then reduce the parameter k by one. Note that this rule can be also applied if $v \in W$ and $u, w \in R$ (v can be colored red since it has no white neighbor.)

An induced white path $(s, v_r, v_{r-1}, \ldots, v_1)$, $r \geq 0$, is a *white tail* if (1) $d_W(s) \leq 1$, $d_W(v_1) = 1$ and $d_W(v_i) = 2$ for all $1 < i \leq r$; (2) $1 \leq d_R(s) \leq 2$ and $d_R(v_i) = 0$ for all $1 \leq i \leq r$. When $r = 0$, a white tail is in fact only a white vertex with one or two red neighbors, and it can be reduced trivially. For simplicity, we also regard it as a white tail.

Lemma 1 (Tail reduction). *Let $(s, v_r, v_{r-1}, \ldots, v_1)$ be a white tail. By applying a series of Red/White Deg-1 and possibly Red Deg-2 folding rules, the white tail can be resolved in polynomial time. If $r < 2$, then at least one red vertex is selected; otherwise at least one white vertex is selected.*

Proof. When $r = 0$, $d_W(s) = 0$, and we can color it red. Then either a Red Deg-1 or Red Deg-2 folding can be applied, and we select a red vertex. We consider the following cases.

- $r = 1$. We can discard v_1 and color s red. Then, Red Deg-1 or Red Deg-2 folding rule can be applied on s according to $d_R(s) = 1$ or 2. The effect is equivalent to selecting a red vertex.
- $r = 2$. We can discard v_1 and color v_2 red. Then v_2 is a degree-1 red vertex, and therefore we select s. Thus, we select a white vertex.
- $r = 3$. We can discard v_1 and color v_2 red. Then v_2 is a degree-1 red vertex, and therefore we select v_3. After v_3 is selected, it becomes the case of $r = 0$.

When $r > 3$, there must be an optimal solution selecting v_3 and discarding $\{v_1, v_2\}$. The process repeats every three vertices. □

We use the following simple scheme to calculate the potential p.

- Initially set $p = 0$.
- When $R = \emptyset$, set $p = 0$.
- When a set of white vertex C becomes a red clique, $p \leftarrow p + |C| - 1$.
- When a red vertex v is selected, $p \leftarrow p - 1$.

In a red clique, all but at most one vertex must be selected. It is easy to see that p is a lower bound since it is increased only when a set of white vertices becomes a red clique. The following property is immediate and we omit the proof.

Lemma 2. *The size of any vertex cover of $G[R]$ is at least p.*

4 Branching Algorithm

In this section we shall show a branching algorithm for the RW-BDD problem. The input of the branching algorithm is a 6-tuple (G, R, W, k, p), where k is the parameter of solution size and p is the potential. The branching algorithm begins with a kernelization, and then exhaustively applies a set of reduction rules and check the terminal cases. Finally it performs the branching rules. In the next two subsections, we first give the details of the algorithm, and then the time complexity is shown.

4.1 Reduction and Branching Rules

The following reduction rules R1–R6 are used in the branching algorithm. The reduction rules come from the observation in the previous section.

R1: Discard all isolated vertices and isolated white edges.
R2: For $v \in R$, if $d(v) = 1$, then remove $N[v]$ and $k \leftarrow k - 1$.
R3: For $v \in W$, if $d_W(v) = 0$, then color v red.
R4: For $v \in W$ and $N(v) = \{u\}$, if $u \in W$, then remove v and color u red; else remove u, v and $k \leftarrow k - 1$.
R5: For any red component, check if the size of its minimum vertex cover is at most k. If not, then no solution; else remove the component and reduce k by the size of its minimum vertex cover.
R6: If there is an isolated white cycle, then check if the size of its minimum RW-BDD set is at most k. If not, then no solution; else remove the component and reduce k by the size of its minimum RW-BDD set.

Lemma 3. *When $p = k$, if D is a k-vertex RW-BDD, then $D \cap W = \emptyset$, and the problem can be solved in $O(1.274^k)$ time.*

Proof. If $v \in W \cap D$, then $D \setminus \{v\}$ would be a vertex cover of $G[R]$, a contradiction to Lemma 2. Thus, $D \cap W = \emptyset$, which means that any solution discards all white vertices. If there is a white vertex v with $d_W(v) = 2$, there is no solution, since there is a white P_3. Otherwise, we discard all white vertices and select all their red neighbors $N(W)$. Finally, we check if there is a $(k - |N(W)|)$-size vertex cover of the remaining (red) graph, which can be solved by a fixed-parameter vertex-cover algorithm in $O(1.274^k)$ time [5]. □

There are three terminal cases D1–D3. Note that the terminal conditions are check in this order and after the reduction rules are exhaustively applied. We also remark that in the literature a reduction rule is sometimes defined as a polynomial process, while R5 employs a fixed-parameter vertex-cover algorithm with running time $O(1.274^k)$ [5]. One way to meet such a definition is to keep all red components and resolve them in a terminal condition. However, it is equivalent.

D1: If $k \geq 0$ and $R = W = \emptyset$, then output "yes" and stop.
D2: If $k \leq 0$ or $k < p$, then no solution.
D3: If $k = p$, then solve the problem according to Lemma 3.

After the reduction rules are exhaustively applied, we can conclude the following properties.

> Each vertex has degree at least two. Furthermore, if $R \neq \emptyset$, then $W \cap N(R) \neq \emptyset$; and if $R = \emptyset$, then there must be a vertex of degree at least three.

The following branching rules are performed in this order. In each branching rule, we choose a white vertex and branch into selecting and discarding it.

1. If there exists $v \in W$ such that $d_W(v) \geq 3$, then branch into discarding and selecting v. (W3-branch)
2. If there exists $v \in W$ such that $d_R(v) \geq 3$, then branch into discarding and selecting v. (R3-branch)
3. If there exists $s \in W$ such that $d_W(s) = 1$, then perform the following *path branch*. Note that in this case, $0 < d_R(s) < 3$ and $d_W(v) < 3$ for any $v \in W$. Find a white path $(s, v_1, v_2, \ldots, v_r, \ldots)$ such that $d_R(v_i) = 0$ for all $1 \leq i < r$ and $d_R(v_r) > 0$. If $d_W(v_r) = 2$ or $d_R(v_r) \geq d_R(s)$, then choose v_r and branch into discarding and selecting v_r. Otherwise, v_r is the other endpoint of this white path and $d_R(v_r) < d_R(s)$, and we branch into discarding and selecting s in this case.
4. If all the above branch rules cannot be applied, then $d_W(v) = 2$ for any $v \in W$, i.e., $G[W]$ is a collection of cycles. Note that for any white cycle C, $N_R(C) \neq \emptyset$. Then, perform the following *cycle branch*: Pick a white vertex s with maximal $d_R(s)$ and branch into discarding and selecting s.

4.2 Time Complexity

Similar to the branching factor method, to find the time complexity of the branching algorithm, we establish a set of recurrences, but there are two variables k and p. To meet the convention that the time complexity is an increasing function, we use a variable substitution. Let $T(k, q)$ be the time complexity with parameter k and potential $p = k - q$. By the technique of *iterative kernelization* [16], we can ignore the polynomial terms in the recurrences for simplicity.

By the terminal cases and Lemma 3, the boundary conditions are as follows.

$$T(k, q) = \begin{cases} O(1) & \text{if } k = 0 \text{ or terminal cases D1 and D2;} \\ O(1.274^k) & \text{if } q = 0 \text{ at terminal case D3.} \end{cases} \tag{1}$$

Next, we consider the four branching rules. Note that after we select a white vertex, the complexity becomes $T(k-1, q-1)$ since the potential is not reduced. But, after selecting a red vertex, the complexity becomes $T(k-1, q)$.

Lemma 4. *If a W3-branch is applied,*

$$T(k, q) \leq T(k-1, q-1) + T(k, q-2). \tag{2}$$

Proof. In a W3-branch, we pick a white vertex v with $d_W(v) \geq 3$. If v is selected, $k \leftarrow k-1$ and p does not change, and the time complexity is $T(k-1, q-1)$. If v is discarded, we construct G' from G by making $N_W(v)$ a red clique and removing v and $N_R(v)$. The parameter $k \leftarrow k - d_R(v)$, and the potential $p \leftarrow p + d_W(v) - 1 - d_R(v)$. Thus, $q \leftarrow q - d_W(v) + 1$. In the worst case, $d_R(v) = 0$ and $d_W(v) = 3$, and therefore the time complexity is $T(k, q-2)$. □

Lemma 5. *If an R3-branch is applied,*

$$T(k,q) = T(k-1, q-1) + T(k-3, q). \tag{3}$$

Proof. In an R3-branch, we pick a white vertex v with $d_R(v) \geq 3$. If v is selected, the time complexity is $T(k-1, q-1)$. If v is discarded, at least three red vertices are selected. Both k and p are reduced by three in the worst case, and therefore the time complexity is $T(k-3, q)$. □

The path-branch is somewhat more complicated, and the analysis of cycle-branch is based on path-branch. The path-branch is divided into three cases for the sake of obtaining a better result for cycle-branch.

Lemma 6 (R0-path branch). *If $(s, v_1, v_2, \ldots, v_r, t)$ is a white path in which $d_W(s) = d_R(s) = d_W(t) = d_R(t) = 1$ and for each $1 \leq i \leq r$, $d_R(v_i) = 0$ and $d_W(v_i) = 2$, then we can make a path-branch with time complexity $T(k,q) = 2T(k-2, q-1)$.*

Proof. Let $N_R(s) = \{u\}$ and $N_R(t) = \{w\}$. First, if $r = 0$, at least two vertices in $\{s, t, u, w\}$ must be selected, and it is easy to verify that there is an optimal solution selecting the two red vertices and discarding the white ones. In this case we do not need to branch.

In case of $r \geq 1$, we branch on selecting or discarding s. After selecting s, we can perform a tail reduction, and by Lemma 1 at least one vertex will be selected. The complexity is $T(k-2, q-1)$. If we discard s, its red neighbor u must be selected and v_1 changes to red. After v_1 is colored red, a series of reductions similar to the tail reduction can also be applied, and at least one white vertex will be selected. Thus the time complexity is also $T(k-2, q-1)$. □

Lemma 7 (R1-path branch). *Suppose that $d_W(v) \leq 2$ and $d_R(v) \leq 2$ for each $v \in W$. If $d_W(s) = d_R(s) = 1$ for some $s \in W$, then we can make a path-branch with time complexity $T(k,q) = T(k-2, q-1) + \max\{T(k-2, q-1), T(k-3, q)\}$.*

Proof. By the degree-one reduction, we can assume that $d_R(v) > 0$ for any degree-1 white vertex v. Since s must be an endpoint of a white path ($s = v_0, v_1, v_2, \ldots$), let v_r be the first vertex on the white path such that $d_R(v_r) > 0$. We branch into selecting or discarding v_r. There are two cases: $d_W(v_r) = 2$ or $d_W(v_r) = 1$.

Suppose that $d_W(v_r) = 1$, i.e., v_r is the other endpoint of this white path. If $d_R(v_r) = d_R(s) = 1$, then by Lemma 6 we have $T(k,q) = 2T(k-2, q-1)$. Otherwise, according to the path-branch rule, we have that $d_R(v_r) = 2$. If we select v_r, a tail reduction can be followed, and the complexity is $T(k-2, q-1)$. If we discard v_r, its two red neighbors must be selected and a tail reduction can also be applied. The time complexity is $T(k-3, q)$. So, in this case, the total time complexity is $T(k-2, q-1) + \max\{T(k-2, q-1), T(k-3, q)\}$.

Suppose that $d_W(v_r) = 2$. For the case of selecting v_r, there must be a tail reduction can be performed on the subpath $(s, \ldots, v_{r-1})$, and the complexity is $T(k-2, q-1)$. If we discard v_r, its red neighbors must be selected. Further, its two white neighbors change to red and edge (v_{r-1}, v_{r+1}) is added, which raises the potential by one. Since $d_R(s) = 1$, a tail reduction or a Red Deg-2 folding can also be applied on the subpath. The complexity is therefore $T(k-2, q-1)$. The overall time complexity in this case is $2T(k-2, q-1)$.

In summary, $T(k, q) = T(k-2, q-1) + \max\{T(k-3, q), T(k-2, q-1)\}$. □

Lemma 8 (R2-path branch). *Suppose that $d_W(v) \leq 2$ and $d_R(v) \leq 2$ for each $v \in W$. If $d_W(s) = 1$ and $d_R(s) = 2$ for some $s \in W$, then we can make a path-branch with time complexity $T(k, q) = T(k-2, q-1) + \max\{T(k-3, q), T(k-1, q-1)\}$.*

Proof. The proof is similar to Lemma 7 except for the case that $d_W(v_r) = 2$ and we discard v_r. In this case, since $d_R(s) = 2$, maybe neither tail reduction nor Red Deg-2 folding can be applied. It leads to a worse time complexity of $T(k-1, q-1) + T(k-2, q-1)$. The overall time complexity is $T(k-2, q-1) + \max\{T(k-3, q), T(k-1, q-1)\}$. □

The following corollary summaries Lemmas 6–8.

Corollary 1. *If a path-branch is applied,*

$$T(k, q) = T(k-2, q-1) + \max\{T(k-3, q), T(k-1, q-1)\} \tag{4}$$

In Lemma 9, we shall show that Eq. (5) and (6) describe the time complexity when a cycle-branch is applied.

$$\begin{aligned} &T(k, q) \\ &= T(k-3, q-2) + \max\left\{\begin{matrix} T(k-4, q-1) \\ T(k-3, q-2) \end{matrix}\right\} \\ &\quad + \max\left\{\begin{matrix} T(k-2, q-1) \\ T(k-3, q-2) + T(k-4, q-1) \\ 2T(k-3, q-2) \end{matrix}\right\}. \end{aligned} \tag{5}$$

$$\begin{aligned} &T(k, q) \\ &= T(k-2, q-1) + T(k-3, q-2) + \max\left\{\begin{matrix} T(k-4, q-1) \\ T(k-2, q-2) \end{matrix}\right\}. \end{aligned} \tag{6}$$

Lemma 9. *If a cycle-branch is applied, the time complexity is given by Eq. (5) and (6).*

Proof. When a cycle-branch is applied, all white vertices have white degree two. That is, each component in $G[W]$ is a cycle. Let C be such a white cycle. A vertex in C is a *port* if it has a red neighbor.

If C has only one port, it is not hard to verify that there is an optimal solution selecting the port. Suppose that C has exactly two ports s and t. After selecting s, we can perform a tail reduction, which give the time complexity $T(k-2, q-1)$. If s is discarded, the red neighbors of s must be selected. In addition, similar to the tail reduction, at least one vertex on C can be determined to select. So for the case of two ports, the time complexity is $T(k, q) = 2T(k-2, q-1)$.

Now we consider the case of more than two ports. Let s be the port with maximal red neighbors. We further divide into two subcases:

- $d_R(s) = 1$. Since all vertices on C has at most one red neighbor, after selecting s, an R1-path branch can be applied. The time complexity is $T(k-3, q-2) + \max\{T(k-4, q-1), T(k-3, q-2)\}$. In case of discarding s, we first select its red neighbor, and then make its two white neighbors a red clique (edge). Following this change, either a Red Deg-2 folding or an R1-path branch can be performed, which give the time complexity $T(k-2, q-1)$ or $T(k-3, q-2) + \max\{T(k-4, q-1), T(k-3, q-2)\}$. The overall time complexity in this case is given by (5).
- $d_R(s) = 2$. When selecting s, an R1-path or R2-path branch can be applied. If an R1-path branch is applied, it is better than the case of $d_R(s) = 1$. Therefore it is sufficient to consider the case that an R2-path branch is applied. By Lemma 8, the time complexity is $T(k-3, q-2) + \max\{T(k-4, q-1), T(k-2, q-2)\}$. When discarding s, two red neighbors of s can be selected, and then again we make its two white neighbors a red clique (edge). However, following this change, maybe no reduction can be applied, and the time complexity is $T(k-2, q-1)$. The overall time complexity in this case is given by (6).

□

Theorem 1. *The RW-BDD problem can be solved in $O(1.882^k + mn)$ time and polynomial space, where n and m denote the numbers of vertices and edges of the input graph, respectively.*

Proof. The height of the search tree is apparently polynomial, and therefore it takes polynomial space. Initially we perform the kernelization in $O(mn)$ time. We need to show the time complexity of the recursive part. By the technique of *iterative kernelization* [16], the polynomial terms in the recurrences can be ignored. To find an upper bound of $T(k, q)$, one looks for a function $f(k, q) = x^k y^q$ satisfying the worst case of the recurrences. First, the boundary condition $T(k, 0) \in O(1.274^k)$ yield a constraint $x \geq 1.274$.

It is clear that the running time of each reduction rule is a polynomial in k, except for R5, in which we find a vertex cover to resolve a red component. If R5 is applied, we find a vertex cover of size $i \leq k$ in $O(1.274^i)$ time, and the recurrence is

$$T(k, q) = \max_{0<i\leq k}\{T(k-i, q) + O(1.274^i)\}. \qquad (7)$$

The recurrence yields the same constraint $x \geq 1.274$ as the boundary conditions. For the branching rules, by Eq. (2)–(6), we have the following inequalities.

$$\begin{aligned}
xy^2 &\geq y + x && \text{from Eq. (2)}\\
x^3y &\geq x^2 + y && \text{from Eq. (3)}\\
x^3y &\geq x + \max\{y, x^2\} && \text{from Eq. (4)}\\
x^4y^2 &\geq x + \max\{y, x\} + \max\{x^2y, x + y, 2x\} && \text{from Eq. (5)}\\
x^4y^2 &\geq x^2y + x + \max\{y, x^2\} && \text{from Eq. (6)}
\end{aligned}$$

Since initially the potential $p = 0$, the final time complexity is $T(k, k)$, and therefore the goal is to minimize xy. By numerical method, it can be easily verified that $x = 1.288$ and $y = 1.461$ satisfies the inequalities, i.e., $T(k, q) \leq 1.288^k \cdot 1.461^q$ for all $k, q \geq 0$. Thus, $T(k, k) \in O(1.882^k)$, and the overall time complexity is $O(1.882^k + mn)$. □

Corollary 2. *The 1-BDD problem can be solved in $O(1.882^k + mn)$ time and polynomial space, where n and m denote the numbers of vertices and edges of the input graph, respectively.*

5 Concluding Remarks

In this paper we demonstrate that using a potential function and the Measure & Conquer approach is helpful for designing parameterized algorithms. By substituting $y = x^\lambda$, we can transform Eq. (1)–(6) into one-variable recurrences and then find the same upper bound of $T(k, k)$ by the standard Measure & Conquer method. Since there are only two variables and the proof is clear, we keep the current proof in Theorem 1.

There are in fact 13 recurrences in Eq. (1)–(6). For example, there are 6 combinations which are written together in (5). However, the current upper bound is dominated by Eq. (2) and (3). It will be not surprised if the time complexity can be further improved. Particularly, by a more detailed discussion on the case that $d_W(v) \leq 3$ and $d_R(v) = 0$ for each white vertex v, one may release the constraint from Eq. (2) and then improve the time complexity. Another future work is to extend the current approach to the δ-BDD problem for $\delta > 1$.

Acknowledgments. This work was supported in part by NSC 101-2221-E-194-025-MY3 and MOST 103-2221-E-194-025-MY3 from National Science Council/Ministry of Science and Technology, Taiwan, R.O.C.

References

1. Abu-Khzam, F.N.: Kernelization algorithms for d-hitting set problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 434–445. Springer, Heidelberg (2007)
2. Betzler, N., Bredereck, R., Niedermeier, R., Uhlmann, J.: On bounded-degree vertex deletion parameterized by treewidth. Discrete Applied Mathematics **160**, 53–60 (2012)

3. Chang, M.S., Chen, L.H., Hung, L.J., Liu, Y.Z., Rossmanith, P., Sikdar, S.: An $O^*(1.4658^n)$-time exact algorithm for the maximum bounded-degree-1 set problem. In: Proceedings of the 31st Workshop on Combinatorial Mathematics and Computation Theory, pp. 9–18 (2014)
4. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: Further observations and further improvements. Journal of Algorithms **41**(2), 280–301 (2001)
5. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theoretical Computer Science **411**(40–42), 3736–3756 (2010)
6. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer-Verlag (1999)
7. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer-Verlag (2013)
8. Kardoš, F., Katrenič, J., Schiermeyer, I.: On computing the minimum 3-path vertex cover and dissociation number of graphs. Theoretical Computer Science **412**, 7009–7017 (2011)
9. Fellows, M.R., Guo, J., Moser, H., Niedermeier, R.: A generalization of nemhauser and trotters local optimization theorem. Journal of Computer and System Sciences **77**(6), 1141–1158 (2011)
10. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer (2010)
11. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56**(5), 25:1–25:32 (2009)
12. 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 2012 and FAW 2012. LNCS, vol. 7285, pp. 199–211. Springer, Heidelberg (2012)
13. Komusiewicz, C., Hffner, F., Moser, H., Niedermeier, R.: Isolation concepts for efficiently enumerating dense subgraphs. Theoretical Computer Science **410**(38–40), 3640–3654 (2009)
14. Moser, H., Niedermeier, R., Sorge, M.: Exact combinatorial algorithms and experiments for finding maximum k-plexes. Journal of Combinatorial Optimization **24**(3), 347–373 (2012)
15. Nishmura, N., Ragde, P., Thilikos, D.M.: Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. Discrete Applied Mathematics **152**, 229–245 (2005)
16. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Information Processing Letters **73**(3–4), 125–129 (2000)
17. Tu, J.: A fixed-parameter algorithm for the vertex cover P_3 problem. Information Processing Letters **115**, 96–99 (2015)
18. Tu, J., Zhou, W.: A factor 2 approximation algorithm for the vertex cover P_3 problem. Information Processing Letters **111**, 683–686 (2011)
19. Tu, J., Zhou, W.: A primal-dual approximation algorithm for the vertex cover P_3 problem. Theoretical Computer Science **412**, 7044–7048 (2011)

Random

Sampling in Space Restricted Settings

Anup Bhattacharya[1], Davis Issac[2], Ragesh Jaiswal[1](✉), and Amit Kumar[1]

[1] Department of Computer Science and Engineering, IIT Delhi, New Delhi, India
{anupb,rjaiswal,amitk}@cse.iitd.ac.in
[2] Max Planck Institute for Informatics, Saarbrücken, Germany
dissac@mpi-inf.mpg.de

Abstract. Space efficient algorithms play a central role in dealing with large amount of data. In such settings, one would like to analyse the large data using small amount of "working space". One of the key steps in many algorithms for analysing large data is to maintain a (or a small number) random sample from the data points. In this paper, we consider two space restricted settings – (i) streaming model, where data arrives over time and one can use only a small amount of storage, and (ii) query model, where we can structure the data in low space and answer sampling queries. In this paper, we prove the following results in above two settings:

– In the streaming setting, we would like to maintain a random sample from the elements seen so far. We prove that one can maintain a random sample using $O(\log n)$ random bits and $O(\log n)$ space, where n is the number of elements seen so far. We can extend this to the case when elements have weights as well.
– In the query model, there are n elements with weights $w_1, \ldots, w_n$ (which are w-bit integers) and one would like to sample a random element with probability proportional to its weight. Bringmann and Larsen (STOC 2013) showed how to sample such an element using $nw+1$ space (whereas, the information theoretic lower bound is nw). We consider the approximate sampling problem, where we are given an error parameter ε, and the sampling probability of an element can be off by an ε factor. We give matching upper and lower bounds for this problem.

1 Introduction

Space efficient algorithms are important when data is too large and cannot be stored in the working memory. Such algorithms have become important with the increasing popularity of mobile devices. These devices, in many cases have small amount of working memory. Also, there is an increasing need to process the huge amount of data being generated over the internet for purposes of data mining. In such scenarios, there is a need for analyzing the data in a *streaming* fashion.

Davis Issac—Major part of this work was done when the author was at IIT Delhi.
Ragesh Jaiswal— Ragesh Jaiswal acknowledges the support of ISF-UGC India-Israel joint research grant 2014.

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 483–494, 2015.
DOI: 10.1007/978-3-319-21398-9_38

This is popularly known as the streaming setting. Note that in all these cases, the other resources such as the running time and amount of randomness[1] are equally important because these determine the power required for processing the data. With the size of computing devices becoming smaller, power is becoming the most important resource to optimize in all such space-restricted settings.

In this work, we look at the basic problem of random sampling. The problem is very simple. Given n objects, the goal is to sample an object (or a few objects) uniformly at random. This is called uniform sampling. We can also consider non-uniform sampling where the objects come along with some weights and the goal is to sample an object with probability proportional to its weight. We discuss these sampling problems in two space-restricted settings. The first setting is the streaming setting, where the data items are available as a stream (i.e., the i^{th} data item is available at time i) and one does not a priori know the number of data items that one should expect to see. In such cases, maintaining a random sample at all times is more challenging than sampling in the classical setting where all data items are present in the memory. This is partly because we cannot store all the data items in the stream due to space-restrictions. The second space-restricted setting that we discuss is the query model. Here, we talk about non-uniform sampling with respect to a given distribution. In this model, one is allowed to pre-process the data and store a representation in small space so as to be able to quickly answer sampling queries. Next, we discuss our results in the above two settings.

1.1 Sampling in the Streaming Setting

In this setting, the data items are available as a stream. That is, the i^{th} data item can be assumed to arrive at time i. Here, we are interested in maintaining a uniformly random sample at all times. We will generalise this for non-uniform sampling. We would like the sampling algorithm to be one-pass and it should save only one data item (since each data item could be very large – files/packets) in its working memory.

The most basic method of doing this is called *reservoir sampling* and it proceeds in the following manner: Let the items be denoted by $O_1, ...$ and the storage location used to store one item in the stream be denoted by S. Store the first object O_1 in S. Subsequently, for O_i, replace the previously stored item in S with O_i with probability $\frac{1}{i}$ and continue without changing S with the remaining probability. Whenever a sample is required, output the object stored in S. Suppose n objects have been seen until the time when the sample was produced. The probability that S stores O_i is $\frac{1}{i} \cdot \frac{i}{i+1} \cdot ... \cdot \frac{n-1}{n} = \frac{1}{n}$. So, we sample with the desired uniform distribution. However, the amount of randomness required in this procedure is large. Let us try to estimate the number of random bits required in this sampling procedure. After O_i arrives, the procedure will need random bits to decide whether the item stored in S needs to be replaced with O_i. This should happen with probability exactly $\frac{1}{i}$. So, at least $\log i$ random

[1] Note that typically a pseudorandom generator is used for generating random bits.

bits will be required for this[2]. So, the number of random bits required for this procedure is at least $\sum_{i=1}^{n} \log i = \Omega(n \log n)$ in expectation[3]. This should be contrasted with the amount of random bits needed to uniformly sample in the classical setting where all the items are present in the memory. In this case, in the classical setting where all the n objects are present in the memory, we will need just $\log n$ random bits in expectation. In this work, we address the gap in the amount of randomness required in the streaming versus the classical settings. We will consider a model in which strict bounds on randomness may be defined as opposed to comparing expected amount of randomness required in classical and streaming settings. We will first formalise the problem and then show that as far as amount of randomness is concerned, there is no gap in the streaming and the classical settings.

First, we note that upper bounds on randomness cannot be defined with respect to perfectly uniform sampling. To see this, let us assume that $n > 2$ is a prime number. For the sake of contradiction, assume that a uniform sample can be generated using r random bits for some finite r. This means that there is a function $f : \{0,1\}^r \rightarrow [n]$ such that for all $i, j \in [n]$, $|\{x|f(x) = i\}| = |\{x|f(x) = j\}|$. This means that 2^r is divisible by n. This is a contradiction since n is a prime number. One natural way of formalising the question of randomness efficiency with respect to uniform sampling is to allow the sampling algorithm to return a null answer (denoted by $\perp$) with certain small probability ε. This means that the sampling algorithm is allowed not to output any member of the set $\{1, ..., n\}$ with probability at most ε. Let us call this model *uniform sampling with ε-error*. We can easily argue (see Section 2) that $\Omega(\log \frac{n}{\varepsilon})$ random bits are required for uniform sampling with ε-error. Following is a simple algorithm that does uniform sampling with ε-error using $O(\log \frac{n}{\varepsilon})$-bits of randomness: With the error parameter ε fixed, we first compute the smallest integer r such that $\lfloor 2^r/n \rfloor > 1 \quad$ and 2^r *mod* $n \leq \varepsilon \cdot 2^r$. Let $k = \lfloor 2^r/n \rfloor$. Consider a function f that maps the first k r-bit strings (ordered lexicographically) to 1, the next k r-bit strings to 2 and so on. The last (2^r *mod* n) strings are mapped to $\perp$. The sampling algorithm computes the function f on r random bits and outputs the value of the function.

Our Contributions: Can the sampling ideas of the classical setting be extended to the streaming setting? The answer is negative. The main bottleneck in the streaming setting is that the value of n is not known in advance whereas the sampling algorithm in the streaming setting must maintain a sample at all times. In this work, we give a sampling algorithm that uses $O(\log \frac{n}{\varepsilon})$-bits of randomness, uses $O(\log \frac{n}{\varepsilon})$-space, and has a running time of $O(n + \log \frac{n}{\varepsilon})$. Moreover, the running time for processing each item is a constant except for the first item which

[2] Actually, more random bits might be required since i might not be a power of 2 and hence we might need to do *rejection sampling*. We can say that $O(\log i)$ random bits are needed in expectation.

[3] We will discuss a more advanced sampling technique by Vitter that requires $\Omega((\log n)^2)$ random bits in expectation.

is $O(\log \frac{1}{\varepsilon})$. [4] We also extend these results for non-uniform sampling. Section 2 gives details of these results. It is important to point out that the lower bound on the number of random bits remains the same if the sampling algorithm is allowed to store more than one item from the stream. Our sampling algorithm matches this lower bound while storing only one item from the stream.

Related Work: The initial techniques for sampling and reservoir sampling were discussed in [Knu81, Vit84, Vit85]. Vitter's [Vit85] work was one of the early works on sampling in the streaming setting where the author was interested in sampling records that were stored in a magnetic tape by making a pass over the tape. However, the computational resource that the author was interested in optimising was the running time of the sampling algorithm and not the amount of randomness or the space. In fact, the author assumed that one can sample random numbers of arbitrary precision in the interval $[0, 1]$ in constant time. Li [Li94] gave quantitative improvement over Vitter's work, again in terms of the running time bounds. Park *et al.* [POS07] extended these ideas for sampling with replacement whereas Efraimidis and Spirakis [ES06] did the same for weighted sampling. Babcock *et al.* [BDM02] gave sampling algorithms where the sample is required to be among the most recent items seen in the stream. They maintain a random sample over a moving window of the most recent items in the stream.

Comparison with Vitter's Reservoir Sampling: Vitter's work on reservoir sampling [Vit85] is the most relevant previous work on this topic. So, it is important to compare our results with those in [Vit85]. We have already seen the most elementary reservoir sampling technique where the i^{th} item is stored with probability $1/i$. The expected number of random bits required for this is $O(\log i)$ and so the expected number of random bits required for the overall algorithm is $O(n \log n)$. Note that this basic technique accesses fresh random bits for every item of the stream. A somewhat more sophisticated technique in [Vit85] reduces the number of times random bits are accessed by the sampling algorithm. This technique works as follows: Suppose at time instance i, we have the i^{th} item stored as the sample in the storage space S. At this time, a positive integer s is chosen from a particular probability distribution $f_i : \mathbb{Z} \rightarrow [0, 1]$. This number s denotes the number of stream items that the algorithm will skip before saving the item $(i + s + 1)$. This probability distribution was defined as $f_i(s) = \frac{i}{(i+s)(i+s+1)}$. So randomness is required only for picking these "skips". It was shown in the paper that the expected number of times such skips need to be picked is $O(\log n)$. In order to sample from the distribution f_i, the paper assumes that one can uniformly sample a real number u of arbitrary precision from $[0, 1]$. One simple idea is to consider the cumulative distribution $F_i(s) = \sum_{i \leq s} f_i(s)$ and then pick the smallest value s such that $F_i(s) \geq u$.

Before further discussion regarding Vitter's work, let us draw a comparison between the models considered by our work and that in [Vit85]. First, in our

[4] The running time is in terms of the number of arithmetic operations. If we take into account the number of bit-operations, then these bounds are larger by a multiplicative factor of $O\left((\log \frac{n}{\varepsilon})^2\right)$.

model, randomness is consumed only in terms of random bits. The reservoir sampling described above uses uniform random samples of arbitrary precision from $[0, 1]$. The second difference one should note is that both basic reservoir sampling and the one described above gives guarantees in terms of expected value of the randomness used. In our model, we are interested in the worst case number of random bits used given that the sampling algorithm is allowed to make some error. So, in some sense, one may interpret our algorithm as a Monte Carlo algorithm and Vitter's reservoir sampling algorithm as a Las Vegas algorithm.

In order to compare our results more closely, we need to remove the requirement of uniform samples from $[0, 1]$ in Vitter's algorithm. So, the next question we address is whether one can sample from the distribution f_i using few random bits instead of uniform samples in $[0, 1]$. Let us try to design an algorithm that sample s from the distribution f_i such that the expected number of random bits used by the algorithm is small. Towards this, we first note that $\mathbf{Pr}[s > i] = 1 - \sum_{j \leq i} f_i(j) = 1 - \sum_{j \leq i} \left(\frac{i}{i+j} - \frac{i}{i+j+1}\right) \leq 1/2$. We now consider the problem as sampling from the set $\{0, 1, ..., i + 1\}$ as per a distribution $\mathcal{D}$, where $\forall j \leq i, \mathcal{D}(j) = f_i(j)$ and $\mathcal{D}(i + 1) = \mathbf{Pr}[s > i]$. With respect to sampling s from f_i, the $(i + 1)^{th}$ item in the above problem corresponds to the case when $s > i$ and in this case we will draw a conditional sample from $\{i + 1, i + 2, ...\}$. Note that if we can show that the expected number of random bits used in the above problem of sampling from $\{0, 1, ..., i+1\}$ is R, then the expected number of random bits required for sampling s using f_i will be $O(R)$. So, let us just focus on the sampling problem above. For this we use the technique of Bringmann and Larsen [BL13] (see section 2.1). We will construct an array A that contains numbers in $\{0, 1, ..., i + 1\}$. A contains the number j exactly $\lfloor (i + 2) \cdot \mathcal{D}(j) \rfloor + 1$ times. The sampling algorithm is as follows:

```
1. Pick a uniformly random $k \in \{1, ..., |A|\}$.
2. If ($k = 1$ or $A[k] \neq A[k - 1]$)
       with probability $(1 - frac((i + 2) \cdot \mathcal{D}(A[k])))$ go to step 1.
3. return $A[k]$.
```

Here $frac(x) = x - \lfloor x \rfloor$. Bringmann and Larsen [BL13] show that the above sampling procedure returns a sample as per distribution $\mathcal{D}$ in constant expected time. Let us estimate the randomness required by this sampling procedure. Note that $|A| \leq 2(i + 2)$ and so step 1 costs $O(\log i)$ random bits. Also, since $\mathcal{D}(j) = \frac{i}{(i+j)(i+j+1)}$, the cost for simulating step 2 is $O(\log i)$ random bits. So, the expected number of random bits required in this sampling procedure is $O(\log i)$. As per our discussion earlier, this means that the expected number of bits required to sample s from the distribution f_i is $O(\log i)$. This further means that the expected number of bits required for Vitter's reservoir sampling algorithm is $O(\log^2 n)$.

In the classical model where all the items are in memory, the expected number of random bits required to sample is $O(\log n)$. So, within the model considered by Vitter's algorithm where one is interested in the expected number of random bits, there is a gap between the bounds in the classical and the streaming settings.

An interesting question is whether this gap should exist. Recall, that in our model where we are interested in the number of random bits when the sampling algorithm is allowed to err with small probability, we show there is no such gap between the classical and streaming settings.

1.2 Succinct Sampling

The second space-restricted setting that we consider is a non-streaming setting where the set of elements are integers $\{1, ..., n\}$. The most natural model of sampling in the non-streaming setting is the *query model*. This is the model used by Bringmann and Larsen [BL13] in their work. Our work within this model may be interpreted as a natural extension of their work. The inputs are w-bit integers $x_1, ..., x_n$. The model includes a pre-processing step where appropriate data structures may be created. Queries for producing a sample as per the weighted distribution are made and should be processed quickly using the data structures created in the pre-processing step. The weighted distribution means that the query algorithm should output i with probability $\frac{x_i}{\sum_j x_j}$.

Bringmann and Larsen [BL13] observed that the classical Walker's alias method [Wal74] in the word RAM model (here unit operations may be performed on words of size w bits) has a pre-processing algorithm that runs in time $O(n)$, answers a sampling query in $O(1)$ expected time, and uses a storage of size $n(w + 2\log n + o(1))$ bits. In order to analyse the space usage, they defined a *systematic case* where the input is read-only and a *non-systematic case* where the input representation may be changed to reduce the total space. The *redundancy* of a solution is the number of bits used in addition to the information-theoretic minimum required for storing the input. Given this, the Walker's alias solution has a redundancy of $(2n\log n + o(n))$-bits. Bringmann and Larsen [BL13] improved this and gave a solution in the systematic case where the preprocessing time is $O(n)$, expected query time is $O(1)$, and the redundancy is $n + O(w)$. They also gave a solution that has 1 bit of redundancy in the non-systematic case. Furthermore, they showed optimality of their results. However, all their results are for *exact* sampling. In our work, we extend their work to *approximate* sampling in the word RAM model.

In many realistic scenarios, we might not be required to sample exactly according to the weighted distribution $x_1, ..., x_n$. One such scenario is the sampling based algorithms for k-means clustering such as the PTAS by Jaiswal *et al.* [JKS14] where the algorithms are robust against small errors in sampling probability. This indeed was the starting point of this work. It may be sufficient to sample from a distribution such that the sampling probabilities are *close* to the exact sampling probabilities defined by the weights $x_1, ..., x_n$. We will consider two models for closeness. First is the *additive model* where the i^{th} item's sampling probability may be between $\left(x_i/(\sum_j x_j) - \varepsilon\right)$ and $\left(x_i/(\sum_j x_j) + \varepsilon\right)$ for some small ε. Second is the *multiplicative model* where the i^{th} item's sampling probability may be between $(1-\varepsilon)\cdot\left(x_i/(\sum_j x_j)\right)$ and $(1+\varepsilon)\cdot\left(x_i/(\sum_j x_j)\right)$ for some small ε.

Before we state our results for approximate sampling, we should first understand the differences between exact and approximate sampling in terms of space usage. Note that the information theoretic lower bound on the amount of space required to do exact weighted sampling given n w-bit integers as input is nw. [5] However, in case of approximate sampling, the information theoretic bounds can be much lower since we can use some lossy representation of the inputs that does not affect the sampling probabilities too much but saves much space. Given this, the non-systematic case (where data is not read-only and may be re-structured) seems more relevant than the systematic case (where the inputs are read-only and have to be retained). So, in our work we discuss only the non-systematic case for approximate sampling. Note that all the algorithms that we study have optimal pre-processing time of $O(n)$ and optimal query time of $O(1)$.

Our Contributions: We show that in the multiplicative model, the lower bound on the space requirement is $\Omega(n \log w + n \log \frac{1}{\varepsilon})$. We design a sampling algorithm and show that the space usage of our algorithm matches this lower bound. In the additive model, we give similar results. However, in this case our algorithms match the lower bound only when ε is a constant independent of n. Due to space limitations, we discuss the details of these contributions in the full version of the paper. [6]

Related Work: Walker [Wal74] gave a solution for exact sampling in the classical setting. Kronmal and Peterson [KP79] improved the preprocessing time of Walker's method. Bringmann and Panagiotou [BP12] studied variants of sampling from discrete distribution problems. All the above mentioned works used Real RAM model of computation. Bringmann and Larsen [BL13] analysed Walker's alias method in Word RAM model of computation and also gave better bounds for exact sampling from discrete distribution problems. Their work is most relevant to our current work on succinct sampling and our results may be regarded as a natural extension to [BL13].

2 Sampling in the Streaming Setting

The input consists of a stream of distinct objects $O_1, O_2, ...$, where the object O_i can be thought of as arriving at time i. At any point of time, we would like to maintain a random sample from the set of objects seen so far. More formally, we would like to maintain a random variable X_t for all time t such that $\mathbf{Pr}[X_t = O_i]$ is same for all $i = 1, \ldots, t$. As mentioned in the introduction, this property cannot be achieved for all values of t. Therefore, the input also specifies a parameter ε – the algorithm is allowed to output a null object $\perp$ with probability at most ε. Therefore, we want the following property to hold for all time t: $\mathbf{Pr}[X_t = \perp] \leq \varepsilon, \mathbf{Pr}[X_t = O_1] = \mathbf{Pr}[X_t = O_2] = \cdots = \mathbf{Pr}[X_t = O_t]$. We

[5] This is not a trivial observation since $x_1, ..., x_n$ and $x_1/2, ..., x_n/2$ give the same weighted distribution. See Lemma 5.1 in [BL13].

[6] The full version of this paper may be found at the following arXiv link: http://arxiv.org/abs/1407.1689

shall call such a sequence X_t of random variables *uniform samples* (with error parameter ε, which will be implicit in the discussion).

In the setting of streaming algorithms, we would like to limit the space available to the algorithm. We allow the algorithm to store only one object at any point of time (besides some local variables) – this is motivated by the fact that each object may be quite large (objects could be large files/packets etc.), and so it may not be feasible to store too many objects in the local memory of the program.

Consider the amount of random bits needed to uniformly sample in the classical setting where all the n items are present in the memory and we need a random sample among these items. It is not difficult to show that $O\left(\log \frac{n}{\varepsilon}\right)$ bits of randomness suffice (w.r.t. uniform sampling with ε-error). In fact, it is also fairly easy to show that any algorithm (even in the non-streaming setting) needs at least these many random bits. We give details of the lower bound on number of random bits in Section 2.1. In Section 2.2, we show that we can maintain an exact sample with only $O\left(\log \frac{n}{\varepsilon}\right)$ bits of randomness (till time n). These results may be extended to the weighted case. We discuss the results for the weighted case in the full version of the paper.

2.1 Background

We consider the off-line problem of generating a uniform sample with error parameter ε from the set of objects $O_1, \ldots, O_n$. The proof of the next lemma may be found in the full version of this paper.

Lemma 1. *We can generate a uniform sample with error parameter ε from a set of n distinct objects using $O(\log \frac{n}{\varepsilon})$ random bits. Further, any algorithm for generating such a sample must use $\Omega(\log \frac{n}{\varepsilon})$ random bits.*

The above idea for upper bound does not work in the streaming setting. The main problem in the streaming setting is that the value of n is not known in advance whereas the algorithm needs to maintain a uniform sample at *all* times. One solution is reservoir sampling where fresh random bits are used after every new item arrives. However, as we have seen, this is costly in terms of the amount of randomness used. In the next section, we discuss a sampling algorithm in the streaming setting that uses $O(\log \frac{n}{\varepsilon})$ random bits till time n, and hence, matches the lower bound result mentioned above.

2.2 Uniform Samples in the Streaming Setting

Let us try to understand some of the challenges of designing sampling algorithms in the streaming setting. Recall that X_t is the random object maintained by the algorithm at time t. Since the algorithm is allowed to store only one object at any time, it does not store any other object at time t. At time $t+1$, when O_{t+1} arrives, the algorithm has only three choices for $X_{t+1} - X_t, O_{t+1}$ or $\perp$. We shall use r_t to denote the number of random bits used by our algorithm till time t.

Given a sequence x_t of r_t random bits, let $f_t(x_t)$ denote the object stored by the algorithm at time t, i.e., $X_t = f_t(x_t)$. Note that the functions f_t need to satisfy a "consistency" property: if $x \in \{0,1\}^{r_t}$ is a prefix of a string $y \in \{0,1\}^{r_{t+1}}$, then $f_{t+1}(y)$ is either $f_t(x)$ or O_{t+1} or $\perp$. This is due to the restriction of the streaming setting. Note that the only stream elements one has access to at time $t+1$ are X_t and O_{t+1}. We now describe our algorithm that we call the *doubling-chopping* algorithm.

The Algorithm. For each time t and $i \in \{1,\ldots,t\} \cup \{\perp\}$, the algorithm will maintain an ordered set $H_i^t \subseteq \{0,1\}^{r_t}$ of strings x for which $f_t(x) = O_i$ (or $\perp$). Of course, this will lead to large space complexity – we will later show that these sets can be maintained implicitly. Initially, at time 0, $H_\perp^0 = \emptyset$ and $r_0 = 0$. We first describe the doubling step in Figure 1. The goal of this step is to ensure that 2^{r_t} stays larger than $\frac{(t+1)^2}{\varepsilon}$. Whenever this does not happen, we increase the value of r_t to ensure that this is the case. The functions f_t are updated accordingly – they just look at the first r_t bits of the input.

```
Algorithm Double(t) :
 1. $r_t \leftarrow r_{t-1}$.
 2. For $i \in \{1,...,t-1\} \cup \{\perp\}$
    • Initialize $H_i^t \leftarrow H_i^{t-1}$.
 3. While $2^{r_t} < \frac{(t+1)^2}{\varepsilon}$
    (i) $r_t \leftarrow r_t + 1$.
    (ii) For $i \in \{1,...,t-1\} \cup \{\perp\}$
       • Initialize $H \leftarrow \emptyset$.
       • For each $x \in H_i^t$ in order
           append $0x$ to $H$.
       • For each $x \in H_i^t$ in order
           append $1x$ to $H$.
       • $H_i^t \leftarrow H$.
```

```
Algorithm Chop(t) :
 1. For every $i \in \{1,\ldots,t-1\}$
    • Define $T_i^t \leftarrow$ last $\left(|H_i^t| - \lfloor \frac{2^{r_t}}{t} \rfloor\right)$ strings
      in $H_i^t$.
    • Define $H_i^t \leftarrow H_i^t \setminus T_i^t$.
 2. Initialize $T \leftarrow \emptyset$.
 3. For $i = 1,\ldots,t-1$
    $T \leftarrow \mathtt{append}(T, T_i^t)$.
 4. If $|T| > \lfloor 2^{r_t}/t \rfloor$
     (a) $T_t^t \leftarrow$ last $\left(|T| - \lfloor \frac{2^{r_t}}{t} \rfloor\right)$ strings in $T$.
     (b) $H_t^t \leftarrow T \setminus T_t^t$ and $H_\perp^t \leftarrow \mathtt{append}(H_\perp^t, T_t^t)$
    Else
     (i) $T_\perp^t \leftarrow$ last $\lfloor 2^{r_t}/t \rfloor - |T|$ strings of $H_\perp^t$.
     (ii) Set $H_\perp^t \leftarrow H_\perp^t \setminus T_\perp^t$ and
       $H_t^t \leftarrow \mathtt{append}(T_\perp^t, T)$.
```

Fig. 1. The doubling and chopping steps

Note that after we call the algorithm `Double`, the new $r_t - r_{t-1}$ bits do not participate in the choice of random sample. In Step 3 of the `Double` algorithm, the set H_i^t is an ordered list – "append" adds an element to the end of the list.

The next step, which we call the chopping step, shows how to modify the function f_t so that some probability mass moves towards O_t. The algorithm is described in Figure 1. The function $\mathtt{append}(T_1, T_2)$ takes two ordered lists and outputs a new list obtained by first taking all the elements in T_1 followed by the elements in T_2 (in the same order). The algorithm maintains the sets H_i^t, where

$i \in \{1, \ldots, t\} \cup \{\perp\}$. Given these sets, the function f_t is immediate. If the string $x \in \{0,1\}^{r_t}$ lies in the set H_i^t, then $f_t(x) = O_i$.

To summarise, at time $t > 1$, we first call the function `Double`(t) and then the function `Chop`(t). The initial conditions (at time $t = 1$) are $r_1 = \lceil \log \frac{4}{\varepsilon} \rceil$, $H_1^1 = \{0,1\}^{r_1}$, and $H_\perp^1 = \emptyset$. It is also easy to check that the functions f_t satisfy the consistency criteria.

Lemma 2. *Suppose $x \in \{0,1\}^{r_{t-1}}$ and $y \in \{0,1\}^{r_t - r_{t-1}}$. Then, $f_t(yx)$ is either $f_{t-1}(x)$ or O_t or $\perp$.*

Proof. Let x and y be as above. Suppose $x \in H_i^{t-1}$ (and so, $f_{t-1}(x) = i$). After the `Double(`t`)` function call, $yx \in H_i^t$. Now consider the function `Chop(`t`)`. After Step 1, if $yx \notin H_i^t$, then it must be the case that yx gets added to the set T. Now, notice that the strings in T get added to either H_t^t or $H_\perp^t$. This proves the lemma. □

The lemma above implies that we can execute the algorithm by storing only one object at any time. Now, we show that the number of random bits used by the algorithm is small.

Lemma 3. *The number of random bits used by the algorithm till time n is $O(\log \frac{n}{\varepsilon})$.*

Proof. Till time n, the algorithm uses at most r_n bits and $2^{r_n} \leq \frac{2(n+1)^2}{\varepsilon}$. □

The correctness of the algorithm follows from the next lemma.

Lemma 4. *For all time $t > 0$, and $i \in \{1, ..., t\}$, $|H_i^t| = \lfloor \frac{2^{r_t}}{t} \rfloor$, and $|H_\perp^t| \leq \varepsilon \cdot 2^{r_t}$.*

Proof. The proof is by induction on t. The base case ($t = 1$) is true from the initial conditions $r_1 = \lceil \log \frac{4}{\varepsilon} \rceil$, $H_1^1 = \{0,1\}^{r_1}$, and $H_\perp^1 = \emptyset$. Now suppose the lemma is true for $t-1$. At time t, we first call `Double(`t`)`. For each $x \in H_i^{t-1}$, we just append all bit strings of length $r_t - r_{t-1}$ to it and this set of strings to H_i^t. Therefore, when this procedure ends, $|H_i^t| = 2^{r_t - r_{t-1}} \cdot \lfloor \frac{2^{r_{t-1}}}{t-1} \rfloor$, for $i = 1, \ldots, t-1$ (using induction hypothesis) and we have

$$|H_i^t| = 2^{r_t - r_{t-1}} \cdot \left\lfloor \frac{2^{r_{t-1}}}{t-1} \right\rfloor \geq 2^{r_t - r_{t-1}} \cdot \left(\frac{2^{r_{t-1}}}{t-1} - 1 \right) \geq \frac{2^{r_t}}{t} \quad (\text{since } 2^{r_{t-1}} \geq t^2/\varepsilon)$$

In Step 1 of the procedure `Chop(`t`)`, we ensure that $|H_i^t|$ becomes $\lfloor \frac{2^{r_t}}{t} \rfloor$ (this step can be done, because the $|H_i^t|$ was at least $\lfloor \frac{2^{r_t}}{t} \rfloor$). After this step, we do not change H_i^t for $i = 1, \ldots, t-1$, and hence, the induction hypothesis is true for these sets. It remains to check the size of H_t^t and $H_\perp^t$.

First assume that $|T| \geq \lfloor \frac{2^{r_t}}{t} \rfloor$. In this case, H_t^t gets exactly $\lfloor \frac{2^{r_t}}{t} \rfloor$ elements. Now suppose $|T| < \lfloor \frac{2^{r_t}}{t} \rfloor$. First observe that $H_\perp^t$ and T are disjoint. Since all strings not in $H_i^t, i = 1, \ldots, t-1$ belong to either $H_\perp^t$ or T, it follows that

$$|H_\perp^t| + |T| = 2^{r_t} - (t-1) \cdot \left\lfloor \frac{2^{r_t}}{t} \right\rfloor \geq \left\lfloor \frac{2^{r_t}}{t} \right\rfloor .$$

Therefore, $|H_\perp^t|$ is at least $\lfloor \frac{2^{r_t}}{t} \rfloor - |T|$, and Step 4(i) in this case can be executed. Clearly, $|H_t^t|$ becomes $\lfloor \frac{2^{r_t}}{t} \rfloor$ as well. Finally,

$$|H_\perp^t| = 2^{r_t} - t \cdot \left\lfloor \frac{2^{r_t}}{t} \right\rfloor \leq 2^{r_t} - t\left(\frac{2^{r_t}}{t} - 1\right) = t \leq \varepsilon \cdot 2^{r_t},$$

where the last inequality follows from the definition of r_t. □

Space Complexity. Note that the use of the sets H_i^t in our algorithm was just for sake of clarity. We need not maintain these sets explicitly. For the current random string x (at time t), we just need to keep track of the set H_i^t to which it belongs – call this set $L(x)$ (the *location* of x). In fact, not only we will keep track of $L(x)$, but we will also keep track of the *rank* of x in the set $L(x)$ – recall that the sets H_i^t are ordered lists, and so, the rank of an element is its position in this order. In addition, we will also keep track of $|H_i^t|$ for $i \in \{1, ..., t\} \cup \{\perp\}$. Note that this includes saving only two numbers since $|H_1^t| = ... = |H_t^t|$. The pseudocodes of our algorithms for implementation purposes are given in the full version of this paper.

Lemma 5. *For every time t, the location and the rank of the current random string x_t can be maintained using $O(\log \frac{t}{\varepsilon})$ space.*

Proof. Suppose the statement is true for $t-1$, and say, $x_{t-1} \in H_i^{t-1}$. During `Double(`t`)`, we will append a random string $y \in \{0,1\}^{r_t - r_{t-1}}$ to x_{t-1}, i.e., $x_t = yx_{t-1}$. For every string preceding x_{t-1} in H_i^{t-1}, we will add $2^{r_t - r_{t-1}}$ strings to H_i^t. Hence, one can easily determine the rank of x_t in H_i^t. Using this fact, we can check whether x_t gets transferred to T or not during Step 1 of `Chop(`t`)`. Moreover, since we know the size of the sets H_i^t (at the beginning of Step 1), we can even calculate the rank of t in T. Since we also know the size of $H_\perp^t$, we can check if x_t gets transferred to H_t^t or $H_\perp^t$ in Step 4 (and its rank in this set). The space needed by the algorithm is proportional to r_t, which is $O(\log \frac{t}{\varepsilon})$. □

Running Time. Finally, we analyse the running time of the algorithm after n time steps. The total number of iterations of `While` loop across all invocations of `Double` until time step n is at most r_n, i.e., $O(\log \frac{n}{\varepsilon})$. The time taken by `Chop(`$t$`)` is proportional to constant number of arithmetic operations – Step 1 is constant number of operations. If the string happens to be in T, its rank can be computed using constant number of operations, and similarly for Step 4. Therefore, the total running time till time n is $O(n + \log \frac{n}{\varepsilon})$. However, the running time of `Double` per unit time step can be more than a constant. It is not difficult to see that except for time step $t = 1$ (when we need to initialise $r_1 = \lceil \log \frac{4}{\varepsilon} \rceil$ etc.) and $t = 2$ (when $r_2 \leq r_1 + 3$), r_{t+1} is at most $r_t + 2$. Therefore, except for time step $t = 1$, the running time per time step is proportional to a constant number of arithmetic operations.

Note that in the above analysis, the running time is in terms of the number of arithmetic operations. However, as n grows, the number of bit operations is a more relevant measure. Since at each time step, the arithmetic operations are over numbers of size $O(\log \frac{n}{\varepsilon})$-bits, the total running time in terms of bit operations will be $O\left(\left(\log \frac{n}{\varepsilon}\right)^2 \cdot (n + \log \frac{n}{\varepsilon})\right)$ and the per item running time will be $O\left((\log (i/\varepsilon))^2\right)$ (w.r.t. the i^{th} item).

The techniques in this section generalises to weighted sampling. The analysis may be found in the full version of this paper.

Acknowledgments. RJ and AK would like to thank Karl Bringmann for discussion on Succinct Sampling. They would also like to thank the Indo-German IMPECS program for making the interaction possible.

References

[BDM02] Babcock, B., Datar, M., Motwani, R.: Sampling from a moving window over streaming data. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 633–634 (2002)

[BL13] Bringmann, K., Larsen, K.G.: Succinct sampling from discrete distributions. In: Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC 2013, pp. 775–782. ACM, New York (2013)

[BP12] Bringmann, K., Panagiotou, K.: Efficient sampling methods for discrete distributions. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 133–144. Springer, Heidelberg (2012)

[ES06] Efraimidis, P.S., Spirakis, P.G.: Weighted random sampling with a reservoir. Information Processing Letters **97**(5), 181–185 (2006)

[JKS14] Jaiswal, R., Kumar, A., Sen, S.: A simple D^2-sampling based PTAS for k-means and other clustering problems. Algorithmica **70**(1), 22–46 (2014)

[Knu81] Knuth, D.E.: The Art of Computer Programming, vol. 2. Addison-Wesley (1981)

[KP79] Kronmal, R.A., Peterson Jr, A.V.: On the alias method for generating random variables from a discrete distribution. The American Statistician **33**(4), 214–218 (1979)

[Li94] Li, K.-H.: Reservoir-sampling algorithms of time complexity $o(n(1 + \log N/n))$. ACM Trans. Math. Software **20**(4), 481–493 (1994)

[POS07] Park, B.-H., Ostrouchov, G., Samatova, N.F.: Sampling streaming data with replacement. Computational Statistics and Data Analysis **52**(2), 750–762 (2007)

[Vit84] Vitter, J.S.: Faster methods for random sampling. Comm. ACM **27**(7), 703–718 (1984)

[Vit85] Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Software **11**(1), 37–57 (1985)

[Wal74] Walker, A.J.: New fast method for generating discrete random numbers with arbitrary frequency distributions. Electronics Letters **10**(8), 127–128 (1974)

Entropy of Weight Distributions of Small-Bias Spaces and Pseudobinomiality

Louay Bazzi(✉)

ECE Department, American University of Beirut, Beirut, Lebanon
louay.bazzi@aub.edu.lb

Abstract. A classical bound in information theory asserts that small L_1-distance between probability distributions implies small difference in Shannon entropy, but the converse need not be true. We show that if a probability distribution on $\{0,1\}^n$ has small-bias, then the converse holds for its weight distribution in the proximity of the binomial distribution. Namely, we argue that if a probability distribution μ on $\{0,1\}^n$ is δ-biased, then $\|\overline{\mu} - bin_n\|_1^2 \leq (2\ln 2)(n\delta + H(bin_n) - H(\overline{\mu}))$, where $\overline{\mu}$ is the weight distribution of μ and bin_n is the binomial distribution on $\{0,\ldots,n\}$. We use this relation to study the notion of pseudobinomiality: we call a probability distributions on $\{0,1\}^n$ pseudobinomial if the weight distribution of each of its restrictions and translations is close to the binomial distribution. We show that, for spaces with small bias, the pseudobinomiality error in the L_1-sense is equivalent to that in the entropy-difference-sense. We also study the notion of average case pseudobinomiality and the resulting questions on the pseudobinomiality of sums of independent small-bias spaces.

Keywords: Pseudorandomness · Small-bias · Shannon entropy · Pseudobinomiality · Fourier analysis

1 Introduction

The ultimate goal of pseudorandomness is to construct low complexity PRGs which look random to all small circuits. Without hardness assumptions [9,17], asymptotically optimal seed lengths are not known even for simple models such as depth-2 circuits and log-space computations. Among the simplest desirable properties of a PRG are the almost k-wise independence property and the stronger small-bias property [15]. Small-bias probability distributions have various applications in pseudorandomness (e.g., [5,13,19] and the references therein). A probability distribution on $\{0,1\}^n$ has small bias if it looks like the uniform distribution to all parity functions on subsets of the n input variables. More formally, consider the characters $\{\chi_z\}_{z\in\{0,1\}^n}$ of the abelian group structure $\mathbb{Z}_2^n$ on $\{0,1\}^n$: $\chi_z(x) \overset{\text{def}}{=} (-1)^{\sum_i x_i z_i}$. A probability distribution μ on $\{0,1\}^n$ is *δ-biased* if $|E_\mu \chi_z| \leq \delta$ for each nonzero $z \in \{0,1\}^n$ [15]. Probability distributions with the δ-bias property and support size $\left(\frac{n}{\delta}\right)^{\Theta(1)}$ can be explicitly constructed from linear codes [1,15].

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 495–506, 2015.
DOI: 10.1007/978-3-319-21398-9_39

The original question which motivated the work reported in this paper is the problem of explicitly constructing small subsets $S \subset \{0,1\}^n$ such that for each nonempty subset of indices $I \subset \{1,\ldots,n\}$ and each translation vector $u \in \{0,1\}^I$, the weight distribution of the translation (over $\mathbb{F}_2$) by u of the restriction of S to $\{0,1\}^I$ looks like the binomial distribution on $\{0,\ldots,|I|\}$. We call S ϵ-pseudobinomial if the distance from the binomial distribution is at most ϵ in the L_1-sense, for each I and u.

The ϵ-pseudobinomiality property is a natural extension of the ϵ-bias property. The more general problem of constructing PRGs for combinatorial shapes was studied by Gopalan et al. [7]. Without hardness assumptions, it is an open problem to construct ϵ-pseudobinomial spaces of size polynomial in n and $\frac{1}{\epsilon}$, i.e., of seed length $O(\log \frac{n}{\epsilon})$.

Nisan generator for log-space [16] gives an n^{-c}-pseudobinomial space of seed length $O(\log^2 n)$, for each constant c. The work of Gopalan et al. [7] leads to an ϵ-pseudobinomial space of seed length $O(\log n + \log^2 \frac{1}{\epsilon})$, which matches the seed length of Nisan generator in the inverse polynomial error regime and improves on it in the subpolynomoial regime. Independently and concurrently with the present paper, De [6] constructed a PRG for combinatorial sums, which results in an ϵ-pseudobinomial space of seed length $O(\log^{3/2} \frac{n}{\epsilon})$. Independently and concurrently also, Gopalan, Kane and Meka [8] constructed an ϵ-pseudobinomial space of seed length $O(\log \frac{n}{\epsilon} \log^{O(1)} \log \frac{n}{\epsilon})$.

A related problem was studied by Lovett et al. [13] and independently by Meka and Zuckerman [14] who constructed an $O(\log n)$ seed-length PRG which fools mod-M gates, where $M = O(1)$ is a power of a prime. Another related work is by Rabani and Shpilka [18], who constructed explicit polynomial complexity ϵ-nets for threshold functions.

In the remainder of this introductory section, we summarize our results and outline the rest of the paper in Section 1.1, then we give some preliminaries in Section 1.2.

1.1 Contribution

Motivated by the above question, we present in this paper a systematic study of pseudobinomial spaces. To get stared, we ignore in Section 2 translations and restrictions, and in general we study the weight distributions $\overline{\mu}$ of probability distributions μ on $\{0,1\}^n$ compared to the binomial distribution bin_n on $\{0,\ldots,n\}$. We show that if μ has small bias, then it is enough to guarantee that the entropy $H(\overline{\mu})$ of its weight distribution is close the entropy $H(bin_n)$ of the binomial in order to conclude that $\overline{\mu}$ is close to bin_n in the L_1-sense. In particular, we show that $\|\overline{\mu} - bin_n\|_1^2 \leq (2\ln 2)(n\delta + H(bin_n) - H(\overline{\mu}))$. The key result behind this bound is a lemma which asserts the non-positivity of all the Fourier coefficients of the log-binomial function $L : \{0,1\}^n \to \mathbb{R}$ given by $L(x) = \lg bin_n(|x|)$. We also give a specific applications of the negative spectrum lemma to the weight entropy of even weight strings.

In Section 3, we study the notion of pseudobinomiality and we conclude that, for spaces with $n^{-\Theta(1)}$-small bias, the pseudobinomiality error in the L_1-sense is equivalent to that in the entropy-difference-sense, in the $n^{-\Theta(1)}$-error regime.

In Section 4, we study the minimum weight entropy $H_{min}(\mu)$ of a probability distribution μ on $\{0,1\}^n$, which we define as the minimum Shannon entropy of the weight distribution of a translation of μ. To compare $H_{min}(\mu)$ to $H(bin_n)$, we study the related notion of average weight entropy $H_{avg}(\mu)$, which we define as the average entropy of the weight distribution of a random translation of μ. Thus, $H_{min}(\mu) \le H_{avg}(\mu)$. We show that $H_{avg}(\mu) \le H(bin_n)$, where the inequality is strict unless μ is the uniform distribution. In Section 5, we study the notion of average case pseudobinomiality, and we conclude that for spaces with $n^{-\Theta(1)}$-small bias, $H_{avg}(\mu)$ is $n^{-\Theta(1)}$-close to $H(bin_n)$. In Section 6, we study local pseudobinomiality and we show that it follows from small bias.

Finally, we discuss in Section 7 resulting questions on the pseudobinomiality of sums of independent small-bias spaces. Reingold and Vadhan asked whether the sum of two independent $n^{-O(1)}$-biased spaces fools log-space [14]. Since any distribution which fools log-space must be pseudobinomial, a natural question is whether the sum of any two independent δ-biased spaces is $O((\delta n)^{\Theta(1)})$-pseudobinomial. Using the above results, we show that this question is equivalent to the weaker question of whether for all independent δ-biased random vectors $X, Y \in \{0,1\}^n$, the entropy of the weight of the sum $H(|X+Y|) \ge \min\{H(|X|), H(|Y|)\} - O((n\delta)^{\Theta(1)})$.

Due to space limitations, we sketch in this paper the key ideas behind the proofs of some of the above claims. The complete proofs are in the full version of the paper [3].

1.2 Preliminaries

We summarize in this section basic Fourier analysis and information theory preliminaries used in this paper.

Fourier Transform Preliminaries. The study of boolean functions using harmonic analysis methods dates back to the 70's (e.g., [11,12]). Identify the hypercube $\{0,1\}^n$ with the group $\mathbb{Z}_2^n$. The *characters* of the abelian group $\mathbb{Z}_2^n$ are $\{\chi_z\}_{z\in\mathbb{Z}_2^n}$, where $\chi_z : \{0,1\}^n \to \{-1,1\}$ is given by $\chi_y(x) = (-1)^{\sum_{i=1}^n x_i y_i}$. Consider the vector space $\mathcal{L}(\{0,1\}^n)$ of complex[1] valued functions on $\{0,1\}^n$ endowed with the inner product $\langle , \rangle$ associated with the uniform distribution U_n on $\{0,1\}^n$: $\langle f, g\rangle = E_{U_n} f\overline{g} = 2^{-n}\sum_x f(x)\overline{g}(x)$, where $^-$ is the complex conjugation operator. The characters $\{\chi_z\}_z$ form an orthonormal basis of $\mathcal{L}(\{0,1\}^n)$, i.e., for each $z, z' \in \{0,1\}^n$, $\langle \chi_z, \chi_{z'}\rangle = 0$ if $z \ne z'$ and $\langle \chi_z, \chi_z\rangle = 1$.

If $f \in \mathcal{L}(\{0,1\}^n)$, its Fourier transform $\widehat{f} \in \mathcal{L}(\{0,1\}^n)$ is given by the coefficients of the unique expansion of f in terms of $\{\chi_z\}_z$:

$$f(x) = \textstyle\sum_z \widehat{f}(z)\chi_z(x) \text{ and } \widehat{f}(z) = \langle f, \chi_z\rangle = E_{U_n} f\chi_z.$$

[1] Except for Lemma 4, all the objects in this paper are over the reals.

We have $\widehat{\widehat{f}} = 2^n f$ and $\langle f, g\rangle = 2^n \langle \widehat{f}, \widehat{g}\rangle = \sum_z \widehat{f}(z)\widehat{g}(z)$. A special case of the latter identity is *Parseval's equality*: $E_{U_n}|f|^2 = \sum_z |\widehat{f}(z)|^2 = \|\widehat{f}\|_2^2$.

Weight Distributions. Throughout the paper, $n \geq 1$ is an integer. If $x \in \{0,1\}^n$, the *weight* of x, which we denote by $|x|$, is the number of nonzero coordinates of x. If μ is a probability distributions on $\{0,1\}^n$, we denote the *weight distribution* of μ by $\overline{\mu}$. That is, $\overline{\mu}$ is the probability distribution on $[0:n] \overset{\text{def}}{=} \{0,1,\ldots,n\}$ given by $\overline{\mu}(w) = \mu(x \in \{0,1\}^n : |x| = w)$. The *uniform distribution* on $\{0,1\}^n$ is denoted by U_n and the *binomial distribution* on $[0:n]$ is denoted by bin_n. Thus, $bin_n(w) = \frac{\binom{n}{w}}{2^n}$ for all $w \in [0:n]$, and $bin_n = \overline{U_n}$.

Information Theory Preliminaries. Let $\mathcal{X}$ be a finite set and γ_1 and γ_2 be two probability distributions on $\mathcal{X}$. In our setup, we are interested in $\mathcal{X} = [0:n]$. The similarity between γ_1 and γ_2 is captured by various measures, some of which are the L_1 distance, the relative entropy and the much weaker notions of distance in entropy. The L_1 distance is also called *total variation* since

$$\|\gamma_1 - \gamma_2\|_1 \overset{\text{def}}{=} \textstyle\sum_w |\gamma_1(w) - \gamma_2(w)| = 2\max_{A\subset\mathcal{X}} |\gamma_1(A) - \gamma_2(A)|. \tag{1}$$

The *relative entropy* of γ_1 with respect to γ_2 is given by

$$D(\gamma_1\|\gamma_2) \overset{\text{def}}{=} \sum_w \gamma_1(w) \lg \frac{\gamma_1(w)}{\gamma_2(w)},$$

where $\lg = \log_2$ is the base-2 logarithm. The relative entropy is not symmetric but it satisfies the nonnegativity property $D(\gamma_1\|\gamma_2) \geq 0$ with equality iff $\gamma_1 = \gamma_2$. If γ is a probability distributions on $\mathcal{X}$, the *Shannon entropy* of γ is defined as $H(\gamma) \overset{\text{def}}{=} -\sum_w \gamma(w) \lg \gamma(w)$.

We have the following classical relations between L_1 distance, relative entropy and distance in entropy.

Lemma 1 ([4]). *If γ_1 and γ_2 are two probability distributions on a finite set $\mathcal{X}$, then*

(a) **(Pinsker's bound)** $\|\gamma_1 - \gamma_2\|_1^2 \leq (2\ln 2)D(\gamma_1\|\gamma_2)$
(b) **(Entropy-difference bound)** *If $\epsilon = \|\gamma_1 - \gamma_2\|_1 \leq \frac{1}{2}$, then $|H(\gamma_1) - H(\gamma_2)| \leq \epsilon \lg \frac{|\mathcal{X}|}{\epsilon}$. Note that for $\mathcal{X} = [0:n]$ and $\epsilon = n^{-c}$, where $c > 0$ is constant, we have $\epsilon \lg \frac{|\mathcal{X}|}{\epsilon} = O(n^{-c}\log n)$.*

See [4] for a general information theory reference. The entropy is a function of the probability distribution, but in some cases it is convenient to argue on the random variables. If A is a random variable taking values in a finite set, its entropy $H(A) \overset{\text{def}}{=} H(\mu_A)$, where μ_A is the probability distribution of A. If B is another random variable taking values in a finite set, for each value of b of B, $H(A|B=b) \overset{\text{def}}{=} H(\mu_{A|B=b})$, where $\mu_{A|B=b}$ is the probability distribution of A

given $B = b$. The *conditional entropy* of A given B is $H(A|B) \overset{\text{def}}{=} E_{b \sim \mu_B} H(A|B = b)$. The *mutual information* is $I(A; B) \overset{\text{def}}{=} H(A) - H(A|B)$. The *joint entropy* is $H(A, B) \overset{\text{def}}{=} H(\mu_{A,B})$, where $\mu_{A,B}$ is the joint probability distributions of A and B. The basic properties of mutual information are: $I(A; B) = I(B; A) = H(A, B) - H(A) - H(B) \geq 0$.

2 Entropy of Weight Distributions and Small-Bias

We are interested in probability distributions on $[0 : n]$ which are weight distributions $\overline{\mu}$ of probability distributions μ on $\{0, 1\}^n$. We would like to study the conditions under which $\overline{\mu}$ is close to the binomial distribution bin_n. The entropy-difference bound (Lemma 1.b) asserts that small L_1-distance between probability distributions implies small difference in entropy, but the converse need not be true. We show in this section that if μ has small-bias, then the converse holds for its weight distribution $\overline{\mu}$ in the proximity of the binomial distribution (Theorem 1). The key result behind this bound is the negative spectrum lemma (Lemma 3). We conclude this section with a specific applications of the negative spectrum lemma: we show that the entropy of the weight of even weight strings is strictly larger than that of odd weight strings if n is even.

The maximum entropy of a probability distribution on $[0 : n]$ is $\lg(n + 1)$ and it is achieved by the uniform distribution $U_{[0:n]}$ on $[0 : n]$. The entropy $H(bin_n)$ of the binomial distribution is is approximately half the maximum entropy:

Lemma 2 (Entropy of the binomial [10]). $H(bin_n) = \frac{1}{2} \lg \frac{\pi e n}{2} + O(\frac{1}{n})$. *Thus,* $H(bin_n) = \left(\frac{1}{2} + \Theta\left(\frac{1}{\lg n}\right)\right) H(U_{[0:n]})$.

It is not hard to see that there are probability distributions μ on $\{0, 1\}^n$ such that $|H(bin_n) - H(\overline{\mu})|$ is as small as zero but $\|bin_n - \overline{\mu}\|_1 = \Theta(1)$. We show that if a probability distribution μ on $\{0, 1\}^n$ has small bias, then it is enough to guarantee that $H(bin_n) - H(\overline{\mu})$ is small to conclude that $\|\overline{\mu} - bin_n\|_1$ is small.

Theorem 1 (Entropy-difference converse bound). *Let* μ *be a probability distributions on* $\{0, 1\}^n$. *If* μ *is* δ*-biased, then*

$$D(\overline{\mu}||bin_n) \leq n\delta + H(bin_n) - H(\overline{\mu}).$$

Hence, $\|\overline{\mu} - bin_n\|_1^2 \leq (2 \ln 2)(n\delta + H(bin_n) - H(\overline{\mu}))$ *by Pinsker's bound (Lemma 1.a).*

Accordingly, we obtain from the nonnegativity of relative entropy the following corollary.

Corollary 1 (Maximum entropy of the weight distribtions of small-bias spaces). *Let* μ *be a probability distribution on* $\{0, 1\}^n$. *If* μ *is* δ*-biased, then* $H(\overline{\mu}) \leq H(bin_n) + n\delta$.

That is, unlike arbitrary probability distribution on $\{0,1\}^n$, $H(\overline{\mu})$ cannot be significantly higher than the entropy of the binomial if it has small bias. For an arbitrary μ, the entropy of its weight distribution can be as large as $H(U_{[0:n]}) = \lg(n+1) \approx 2H(bin_n)$ (e.g., let μ be uniformly supported by a subset $S \subset \{0,1\}^n$ such that for each $w \in [0:n]$, S contains exactly one string of weight w).

The key behind the entropy-difference converse bound is the following lemma.

Lemma 3 (Negative spectrum lemma). *Let $L : \{0,1\}^n \to \mathbb{R}$ be log-binomial function given by: $L(x) = \lg bin_n(|x|)$. Then the Fourier transform $\widehat{L}$ of L is non-positive: $\widehat{L}(z) \leq 0$ for each $z \in \{0,1\}^n$. Moreover, $\hat{L}(z) = 0$ if $|z|$ odd, and $\hat{L}(z) < 0$ if $z \neq 0$ and $|z|$ even.*

The origin of the log-binomial function in our context is the equation

$$D(\overline{\mu}||bin_n) = E_{U_n}L - E_{\mu}L + H(bin_n) - H(\overline{\mu}),$$

which holds for any probability distribution μ on $\{0,1\}^n$ and follows from the definitions of D, H, and L. The negative spectrum lemma implies that $\|\widehat{L}\|_1$ is small, namely $\|\widehat{L}\|_1 = -\sum_z \widehat{L}(z) = -L(0) = -\lg bin_n(0) = n$, which implies the entropy-difference converse bound since $|E_{U_n}L - E_{\mu}L| \leq \|\widehat{L}\|_1 \delta$.

The proof of the negative spectrum lemma is analytical in nature. It boils down to showing that if $m \geq 2$ is even and $a, b \geq 0$ are integers, then

$$\beta_m(a,b) \overset{\text{def}}{=} \sum_{w=0}^{m} (-1)^w \binom{m}{w} \lg \binom{m+a+b}{w+a} < 0.$$

At a high level, we do this by first showing that $\lim_{c\to\infty} \beta_m(c,c) = 0$ using de Moivre-Laplace normal approximation of the binomial. Then we argue that β_m is strictly increasing in the sense that $\beta_m(a,b) < \beta_m(a+a', b+b')$ for each $a, b, a', b' \geq 0$ such that not both a' and b' are zero. We derive the latter from the inequality $\sum_w (-1)^w \binom{m}{w} \lg(w+i) < 0$, for all $i \geq 0$ and even $m \geq 2$, which we establish by examining the Taylor series of the logarithm and the moment generating function $g_s : \{0,1\}^m \to \mathbb{R}$ given by $g_s(x) = (-1)^{|x|} e^{s(|x|-m/2)}$.

Specific Application of the Negative Spectrum Lemma: It follows from the negative spectrum lemma that the entropy of the weight distribution of even weight strings is strictly larger than that of odd weight strings if n is even.

Corollary 2 (Weight entropy of the even weight strings). *Let $E_n \in \{0,1\}^n$ be a uniformly random vector of even weight and $O_n \in \{0,1\}^n$ a uniformly random vector of odd weight. Then $H(|E_n|) = H(|O_n|)$ if n is odd, and $H(|E_n|) > H(|O_n|)$ if n is even*

3 Pseudobinomiality

We study in this section the notion of pseudobinomiality in the L_1 and entropy senses. We conclude from the entropy-difference bound and the entropy-difference converse bound that for spaces with $n^{-\Theta(1)}$-small bias, the two notions are equivalent in the $n^{-\Theta(1)}$-error regime (Corollary 3).

First we need some notations on translations and restrictions. If μ is a probability distribution on $\{0,1\}^n$ and $u \in \{0,1\}^n$ is a translation vector, define the *translation* $\sigma_u \mu$ of μ by u to be the probability distribution on $\{0,1\}^n$ given by $(\sigma_u\mu)(x) = \mu(x+u)$. If $I \subset [n]$ is nonempty, the *restriction* μ^I of μ on I is the probability distribution on $\{0,1\}^I$ given by $\mu^I(y) = \mu(x : x_I = y)$. We also use the previously defined notations on $\{0,1\}^n$ for probability distributions on $\{0,1\}^I$. For instance, $bin_{|I|}$ is the binomial distribution on $[0 : |I|]$, and if $u \in \{0,1\}^I$, then $\overline{\sigma_u\mu^I}$ is the weight distribution of the translation $\sigma_u\mu^I$ of the restriction μ^I of μ.

Definition 1 (Pseudobinomiality in the L_1-sense). *A probability distribution μ on $\{0,1\}^n$ is called* ϵ-pseudobinomial in the L_1-sense *if the weight distribution of each translation of a restriction of μ is ϵ-close to the binomial distribution in the L_1-sense. That is, μ is ϵ-pseudobinomial in the L_1-sense if for each nonempty set of indices $I \subset [n]$ and each translation vector $u \in \{0,1\}^I$, we have $\|\overline{\sigma_u\mu^I} - bin_{|I|}\|_1 \leq \epsilon$.*

It follows from the total variation equation (1) that μ is ϵ-pseudobinomial in L_1-sense iff $2|\overline{\sigma_u\mu^I}(A) - bin_{|I|}(A)| \leq \epsilon$, for all nonempty $I \subset [n]$, all $A \subset I$ and $u \in \{0,1\}^I$. The ϵ-bias property corresponds to the special case when A is the subset of even numbers in $[0 : |I|]$. Accordingly, ϵ-pseudobinomiality in the L_1-sense implies ϵ-bias. It is a natural extension of the ϵ-bias property, which is also invariant under translations and preserved by restrictions.

Similarly, we can define pseudobinomiality in the L_∞-sense and the L_2-sense. They are both equivalent to pseudobinomiality in the L_1 sense in the $n^{-\Theta(1)}$-error regime [2] . In more classical pseudorandomness terms, ϵ-pseudobinomiality in the L_∞-sense is equivalent to ϵ-fooling[3] all functions $f_{a,b} : \{0,1\}^n \to \{0,1\}$ given by $f_{a,b}(x) = 1$ iff $\sum_i a_i x_i = b$, where $a \in \{0,\pm 1\}^n$ and $b \in \mathbb{Z}$. Thus, constructing ϵ-pseudobinomial spaces is a special case of the problem of of constructing PRGs for combinatorial shapes considered by Gopalan et al. [7].

A related work is that of Lovett et al. [13] and Meka and Zuckerman [14] who gave an $O(\log n)$ seed-length PRG which fools mod-M gates, where $M = O(1)$ is a power of a prime. The framework of Mod-M gates allows the coefficients $\{a_i\}_i$ to take arbitrary values mod M, but the restriction $M = O(1)$ makes pseudobinomiality more difficult to achieve. Another related work is by Rabani and

[2] With the L_p pseudobinomiality error of μ defined as $\epsilon_p(\mu) \overset{\text{def}}{=} \min_{I,u} \|\overline{\sigma_u\mu^I} - bin_{|I|}\|_p$, we have $\epsilon_\infty(\mu) \leq \epsilon_2(\mu) \leq \epsilon_1(\mu) \leq \sqrt{(n+1)}\epsilon_2(\mu) \leq (n+1)\epsilon_\infty(\mu)$.

[3] A probability distribution μ on $\{0,1\}^n$ *ϵ-fools* a function $f : \{0,1\}^n \to \{0,1\}$ if $|E_\mu f - E_{U_n} f| \leq \epsilon$.

Shpilka [18], who constructed explicit polynomial complexity ϵ-nets for threshold functions. A threshold function $t_{a,b} : \{0,1\}^n \to \{0,1\}$ given by $t_{a,b}(x) = 1$ iff $\sum_i a_i x_i \leq b$, where $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The result of [18] is an explicit constriction of a subset $S \subset \{0,1\}^n$ of size polynomial in n and $\frac{1}{\epsilon}$ such that for all $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$, if $E_{U_n} t_{a,b} > \epsilon$, then $E_S t_{a,b} \neq 0$. In the context of threshold functions, $n^{-\Theta(1)}$-pseudobinomiality is equivalent to $n^{-\Theta(1)}-$fooling binary threshold functions $t_{a,b}$ corresponding to the case when $a \in \{0, \pm 1\}^n$ and $b \in \mathbb{Z}$.

We define below pseudobinomiality in the entropy-sense.

Definition 2 (Minium weight entropy). *If μ is a probability distribution on $\{0,1\}^n$, define the* min-weight entropy *of μ: $H_{min}(\mu) = \min_{u \in \{0,1\}^n} H(\overline{\sigma_u \mu})$, i.e., $H_{min}(\mu)$ is the minimum Shannon entropy of the weight distribution of a translation of μ.*

Definition 3 (Pseudobinomiality in the entropy-sense). *A probability distribution μ on $\{0,1\}^n$ is called* ϵ-pseudobinomial in the entropy-sense *if for each nonempty index subset $I \subset [n]$, we have $H_{min}(\mu^I) \geq H(bin_{|I|}) - \epsilon$.*

It follows from the entropy-difference bound (Lemma 1.b) and the entropy-difference converse bound (Theorem 1) that for spaces with $n^{-\Theta(1)}$-small bias, pseudobinomiality in the L_1-sense is equivalent to pseudobinomiality in the entropy-sense in the $n^{-\Theta(1)}$-error regime.

Corollary 3 (Pseudobinomiality: L_1 and entropy equivalence). *let μ be a δ-biased probability distribution μ on $\{0,1\}^n$ and $\epsilon > 0$. Then:*

a) If $\epsilon \leq 1/2$ and μ is ϵ-pseudobinomial in the L_1-sense, then it $\epsilon \lg \frac{n+1}{\epsilon}$-pseudobinomial in the entropy-sense.

b) If μ is ϵ-pseudobinomial in entropy-sense then it is $\sqrt{(2\ln 2)(n\delta + \epsilon)}$-pseudo binomial in the L_1-sense.

4 Min-Weight Entropy, Average-Weight Entropy, and the Binomial Entropy

We elaborate in this section on the notion of min-weight entropy compared to the binomial entropy, and we study the related notion of average-weight entropy. There are distributions μ on $\{0,1\}^n$ such that the weight distribution $\overline{\mu}$ of μ is the uniform distribution on $[0:n]$, and hence $H(\overline{\mu}) = \log(n+1) \approx 2H(bin_n)$. We note that each probability μ on $\{0,1\}^n$ has a translation whose weight distribution has entropy at most $H(bin_n)$, and hence $H_{min}(\mu) \leq H(bin_n)$. We argue also that the inequality is strict unless μ is the uniform distribution U_n on $\{0,1\}^n$. To do so, we study the notion of average-weight entropy which we obtain by replacing the min with an average and we can interpret in terms of conditional entropy. We compare $H(bin_n)$ to the average-weight entropy, which is an upper bound on the min-weight entropy.

Definition 4 (Average-weight entropy). *If μ be a probability distribution on $\{0,1\}^n$, define the* average-weight entropy $H_{avg}(\mu)$ *of μ to be the average Shannon entropy of the weight distribution of a random translation of μ, i.e., $H_{avg}(\mu) \overset{\text{def}}{=} E_{u \sim U_n} H(\overline{\sigma_u \mu}) = H(|X+U| \mid U)$, where $X \sim \mu$ and $U \sim U_n$ are independent.*

By definition, $H_{min}(\mu) \leq H_{avg}(\mu)$. On the other hand, $H(bin_n) - H_{avg}(\mu)$ is the mutual information between the weight $|X+U|$ of $X+U$ and U:

$$H(bin_n) - H_{avg}(\mu) = H(|X+U|) - H(|X+U| \mid U) = I(|X+U|;U) \geq 0.$$

We argue that the inequality is strict unless μ is the uniform distribution.

Theorem 2 (Min-weight entropy, avg-weight-entropy, and the binomial entropy). *Let μ be a probability distribution on $\{0,1\}^n$. Then $H_{min}(\mu) \leq H_{avg}(\mu) \leq H(bin_n)$, where the inequality $H_{avg}(\mu) \leq H(bin_n)$ is strict unless μ is the uniform distribution U_n on $\{0,1\}^n$. That is, U_n is the unique maximum min-weight entropy distribution and the unique maximum average-weight entropy distribution.*

5 Average Case Pseudobinomiality

We study in this section average-case pseudobinomiality. We show that for spaces with small bias, the weight distribution of a random translation of the space is close the binomial distribution in both the L_1 and entropy senses. To do so, we start with the L_2 distance.

Theorem 3 (Average case pseudobinomiality). *Let μ be a δ-biased probability distribution on $\{0,1\}^n$, then*

a) $E_{u \sim U_n} \|\overline{\sigma_u \mu} - bin_n\|_2^2 \leq \delta^2$
b) $E_{u \sim U_n} \|\overline{\sigma_u \mu} - bin_n\|_1 \leq \delta \sqrt{n+1}$
c) $I(|X+U|;U) = H(bin_n) - H_{avg}(\mu) = O(n\delta + \sqrt{n}\delta \lg \frac{1}{\delta})$*, where $X \sim \mu$ and $U \sim U_n$ are independent, and I is the mutual information function.*

Thus, if the bias is small, almost all translation of μ have weight distributions close to the binomial distribution. In this sense, small-bias implies average case pseudobinomiality.

The key behind the average L_2-bound (a) is the following lemma which is inspired by an argument in the paper of Viola [19] (the argument used to establish Lemma 3 in [19]).

Lemma 4 (Variance bound). *If $f : \{0,1\}^n \to \mathbb{C}$ and μ is a δ-biased probability distribution on $\{0,1\}^n$, then $E_{u \sim U_n} |E_{\sigma_u \mu} f - E_{U_n} f|^2 \leq \delta^2 (E_{U_n}|f|^2 - |E_{U_n} f|^2)$.*

The proof of the variance bound is based on Parseval's equality. We derive (a) by applying the variance bound to $f = I_w$ for all $w \in [0:n]$, where $I_w : \{0,1\}^n \to \{0,1\}$ is the weight indicator function given by $I_w(x) = 1$ iff $|x| = w$. The average L_1-bound (b) follows from (a) via Jensen's inequality. The average-weight entropy bound (c) follows from (b), Jensen's inequality and the negative spectrum lemma.

6 Local Pseudobinomiality

It is not hard to show that small-bias does not imply pseudobinomiality in the L_1-sense or the entropy-sense even if the bias is exponentially small, but it is enough to guarantee local pseudobinomiality on small subsets of indices.

Lemma 5. *There exists a $2^{-\Omega(n)}$-biased probability distribution μ on $\{0,1\}^n$ such that $\|\overline{\mu} - bin_n\|_1 = \Theta(1)$ and $H(bin_n) - H(\overline{\mu}) = \Omega(1)$.*

Lemma 6 (Local pseudobinomiality). *Let $k \geq 1$ and μ be a δ-biased probability distribution μ on $\{0,1\}^n$. Then for each nonempty $I \subset [n]$ of size $|I| \leq k$, and each $u \in \{0,1\}^I$, we have $\|\overline{\sigma_u \mu^I} - bin_{|I|}\|_1 \leq 2\delta 2^{k/2}$.*

7 Sum of Spaces Conjectures

The work of Viola [19] suggests exploring the derandomization capabilities of small-bias spaces. Reingold and Vadhan asked whether the sum of two independent $n^{-O(1)}$-biased spaces fools log-space [14]. Since any distribution which fools log-space must be pseudobinomial, a natural question is whether the sum of two independent δ-biased spaces is $O((\delta n)^{\Theta(1)})$-pseudobinomial.

If $X, Y \in \{0,1\}^n$ are independent random vectors distributed according to μ_X and μ_Y, consider the sum $X + Y$ over $\mathbb{F}_2$. The probability distribution of $X + Y$ is the convolution $\mu_X * \mu_Y$: $(\mu_X * \mu_Y)(z) = \sum_x \mu_X(x)\mu_Y(x+z)$.

Conjecture 1 (**Pseudobinomiality of sum).** For all independent δ-biased random vectors $X, Y \in \{0,1\}^n$, the sum $X + Y$ is $O((n\delta)^{\Theta(1)})$-pseudobinomial in the L_1-sense.

That is, there exist constants $a, b, c, n_0 > 0$, such that for each $\delta > 0$, for each integer $n > n_0$, and all independent δ-biased random vectors $X, Y \in \{0,1\}^n$, the sum $X + Y$ is $a\delta^b n^c$-pseudobinomial in the L_1-sense.

Using the above results, we show that Conjecture 1 is equivalent to each of the following three conjectures.

Conjecture 2 (**Entropy of sum).** For all independent δ-biased random vectors $X, Y \in \{0,1\}^n$, the Shannon entropy of the weight of $X + Y$ satisfies $H(|X + Y|) \geq H(bin_n) - O((n\delta)^{\Theta(1)})$.

That is, there exist constants $a, b, c, n_0 > 0$, such that for each $\delta > 0$, for each integer $n > n_0$, and all independent δ-biased random vectors $X, Y \in \{0,1\}^n$, $H(|X + Y|) \geq H(bin_n) - a\delta^b n^c$.

Conjecture 3 (**Entropy of sum without the binomial: max version).** For all independent δ-biased random vectors $X, Y \in \{0,1\}^n$,

$$H(|X + Y|) \geq \max\{H(|X|), H(|Y|)\} - O((n\delta)^{\Theta(1)}).$$

Conjecture 4 (**Entropy of sum without the binomial: min version).** For all independent δ-biased random vectors $X, Y \in \{0,1\}^n$,

$$H(|X+Y|) \geq \min\{H(|X|), H(|Y|)\} - O((n\delta)^{\Theta(1)}).$$

Lemma 7. *Conjectures 1,2,3 and 4 are equivalent.*

If A and B are real valued random variables (taking values in a finite set for instance), a simple conditioning argument show that

$$H(A+B) \geq \max\{H(A), H(B)\}.$$

Unfortunately, the picture is more complex in the context of weight distributions and mod-2 sums; we show that for highly biased X or Y, the entropy $H(|X+Y|)$ can be smaller than $H(|X|)$ and $H(|Y|)$.

Lemma 8. *For infinitely many values of n, there exists a coset $S \subset \mathbb{F}_2^n$ of an $\mathbb{F}_2$-linear code such that $H(|X|) = H(|Y|) = \Theta(\log n)$ but $H(|X+Y|) = \Theta(1)$, where X and Y are random vectors chosen independently and uniformly from S.*

Lemma 9. . *There exist a $2^{-\Omega(n)}$-biased random vector $X \in \{0,1\}^n$ and a deterministic vector $Y \in \{0,1\}^n$ such that $H(|X+Y|) = H(|X|) - \Omega(1)$.*

The equivalence between Conjectures 1 and 2 follows from the equivalence between pseudobinomiality in the L_1-sense and the entropy-sense, the fact that small-bias is preserved by restrictions and is invariant under translations, and the local pseudobinomiality of small-bias spaces. In Conjectures 3 and 4, we get rid of the binomial distribution using Theorem 3.c, which says that the average-weight entropy of a small-bias space is a good approximation of the entropy of the binomial distribution. Namely, to show that Conjecture 2 follows from Conjecture 4, we argue using Theorem 3.c that there is $u \in \{0,1\}^n$ such that both $H(|X+u|)$ and $H(|Y+u|)$ are close to $H(bin_n)$, and hence by Conjecture 4 applied to $X+u$ and $Y+u$, $H(|X+Y|) = H(|(X+u)+(Y+u)|)$ is close to $H(bin_n)$. The fact that Conjecture 3 follows from Conjecture 2 is based on Corollary 1, which asserts that the entropy of the weight distribution of a probability distribution on $\{0,1\}^n$ with small bias cannot significantly exceed $H(bin_n)$.

Finally, we note that, by conditioning on Y, the above conjectures follow from the following (possibly stronger) conjecture.

Conjecture 5 (**Lower sandwiching the weight-entropy function).** For any δ-biased random vector $X \in \{0,1\}^n$, the weight-entropy function $h : \{0,1\}^n \to \mathbb{R}$ given by $h(u) \overset{\text{def}}{=} H(|X+u|)$ has a lower sandwiching function $g \leq h$ such that $E_{U_n}(h-g) = O((n\delta)^{\Theta(1)})$ and $\delta\|\widehat{g}\|_1 = O((n\delta)^{\Theta(1)})$.

Lemma 10. *Conjecture 5 implies Conjecture 2.*

Acknowledgments. The author would like to thank Ibrahim Abou-Faycal for helpful discussions.

References

1. Alon, N., Goldreich, O., Hastad, J., Peralta, R.: Simple Constructions of Almost k-wise Independent Random Variables. Random Structures and Algorithms **3**(3), 289–304 (1992)
2. Azar, Y., Motwani, R., Naor, J.: Approximating probability distributions using small sample spaces. Combinatorica **18**(2), 151–171 (1998)
3. Bazzi, L.: Entropy of weight distributions of small-bias spaces and pseudobinomiality. Submitted. Available at Electronic Colloquium on Computational Complexity, Report TR14-112 (2014)
4. Cover, T., Thomas, J.: Elements of Information Theory, 2nd edn. Wiley (2006)
5. De, A., Etesami, O., Trevisan, L., Tulsiani, M.: Improved pseudorandom generators. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX 2010. LNCS, vol. 6302, pp. 504–517. Springer, Heidelberg (2010)
6. De, A.: Beyond the central limit theorem: asymptotic expansions and pseudorandomness for combinatorial sums. Available at Electronic Colloquium on Computational Complexity, Report TR14-125 (2014)
7. Gopalan, P., Meka, R., Reingold, O., Zuckerman, D.: Pseudorandom generators for combinatorial shapes. SIAM Journal of Computing **42**, 1051–1076 (2013)
8. Gopalan, P., Kane, D., Meka, R.: Pseudorandomness for concentration bounds and signed majorities. arXiv:1411.4584 [cs.CC] (2014)
9. Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: derandomizing the XOR lemma. In: Proc. 29th Annual ACM Symposium on the Theory of Computing, pp. 220–229 (1997)
10. Jacquet, P., Szpankowski, W.: Entropy computations via analytic depoissonization. IEEE Transactions on Information Theory. **45**(4), 1072–1081 (1999)
11. Kahn, J., Kalai, G., Linial, N.: The influence of variables on Boolean functions. In: Proc. of the 29th Annual Symposium on Foundations of Computer Science, pp. 68–80 (1988)
12. Lechner, R.J.: Harmonic analysis of switching functions. In: Recent Development in Switching Theory, pp. 122–229. Academic Press (1971)
13. Lovett, S., Reingold, O., Trevisan, L., Vadhan, S.: Pseudorandom bit generators that fool modular sums. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) Approximation, Randomization, and Combinatorial Optimization. LNCS, vol. 5687, pp. 615–630. Springer, Heidelberg (2009)
14. Meka, R., Zuckerman, D.: Small-bias spaces for group products. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) Approximation, Randomization, and Combinatorial Optimization. LNCS, vol. 5687, pp. 658–672. Springer, Heidelberg (2009)
15. Naor, J., Naor, M.: Small bias probability spaces: efficient constructions and applications. SIAM J. on Computing **22**(4), 838–856 (1993)
16. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica **12**(4), 449–461 (1992)
17. Nisan, N., Wigderson, A.: Hardness vs. randomness. In: Proc. 29th IEEE Symposium on Foundations of Computer Science, pp. 2–11 (1988)
18. Rabani, Y., Shpilka, A.: Explicit construction of a small epsilon-net for linear threshold functions. In: Proc. 40th Annual ACM Symposium on Theory of Computing, pp. 649–658 (2009)
19. Viola, E.: The sum of d small-bias generators fools polynomials of degree d. In: IEEE Conference on Computational Complexity, pp. 124–127 (2008)

Optimal Algorithms for Running Max and Min Filters on Random Inputs

Hongyu Liang[1], Shengxin Liu[2](✉), and Hao Yuan[3]

[1] Facebook, Inc., Menlo Park, USA
hongyuliang86@gmail.com

[2] Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong (HK)
shengxliu2-c@my.cityu.edu.hk

[3] Bopu Technologies, Shenzhen, China
hao@bopufund.com

Abstract. Given a d-dimensional array and an integer p, the max (or min) filter is the set of maximum (or minimum) elements within a d-dimensional sliding window of edge length p inside the array. The current best algorithm for computing the 1D max (or min) filter, due to Yuan and Atallah [14], uses $1 + o(1)$ comparisons per sample in the worst case. As a direct consequence, the d-dimensional max (or min) filter can be computed in $(1 + o(1))d$ comparisons per sample, and the d-dimensional max and min filters can be computed simultaneously using $(2 + o(1))d$ comparisons per sample. Both bounds are the best known results for the corresponding problems, on both worst-case inputs and independently and identically distributed (i.i.d.) inputs.

In this paper, we present an algorithm for computing d-dimensional max and min filters simultaneously on i.i.d. inputs that uses $1.5 + o(1)$ expected comparisons per sample. This is the first algorithm for d-dimensional max and min filters (on i.i.d. inputs) that gets rid of the dependence on d in (the dominating term of) the number of comparisons needed. It is also asymptotically optimal. In particular, for the 1D case, our algorithm improves the previous best upper bound of $2 + o(1)$ to $1.5 + o(1)$. As a by-product of our algorithm, we can also compute the d-dimensional max (or min) filter on i.i.d. inputs using $1 + o(1)$ expected comparisons per sample, which matches the bound for the 1D case.

1 Introduction

Given an array $X = (X[1], X[2], \cdots, X[n])$ and an integer p, the *1D running max filter* of X is the collection of maximum elements within each window of length p in X, i.e., $\max_{j \in [i, i+p-1]} X[j]$ for all $1 \le i \le n - p + 1$. By changing max to min we get the *1D running min filter* of X. The *d-dimensional max and min filters* can be defined similarly, where the input is a d-dimensional array and

This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 124411].

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 507–520, 2015.
DOI: 10.1007/978-3-319-21398-9_40

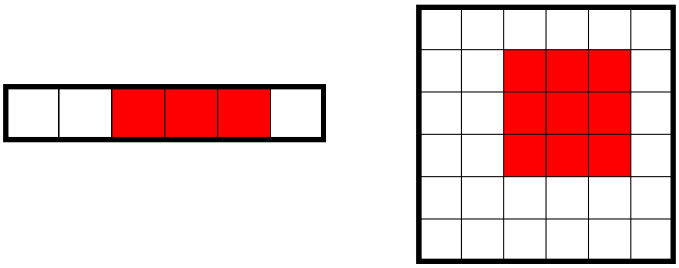


Fig. 1. Examples of running max/min filters with $n = 6$ and $p = 3$. The picture on the left represents a 1D instance, and that on the right is a 2D example. The maximum over each red area is an item of the corresponding max filter.

each window is a d-dimensional hypercube of edge length p. (See Figure 1 for examples of $d = 1$ and $d = 2$.)

The running max and min filters correspond to the dilation and erosion operators over gray-scale images using flat structuring elements, which are fundamental operators in the area of morphological image analysis [10]. Thus, the problem of computing max and min filters has attracted much attention these years. Following previous work (e.g. [1,5,11]), the efficiency of algorithms for computing the filters is measured by the number of *comparisons per sample*, i.e., the total number of comparisons between input elements made by the algorithm over the number of sample (or output) points. The other computational costs (e.g., comparisons between indices used in an iteration) are not taken into account.

It is obvious that the 1D max (or min) filter can be computed using $p - 1$ comparisons per sample in the naïve way, but this is far from optimal. In fact, there is an algorithm due to Yuan and Atallah [14] that computes the 1D max (or min) filter using only $1 + o(1)$ comparisons per sample, which is optimal up to the $o(1)$ term. (Here the $o(1)$ term is with respect to p, i.e., it tends to 0 when p tends to infinity.) A direct consequence is that the d-dimensional max (or min) filter can be computed in $(1 + o(1))d$ comparisons per sample by applying the algorithm for the 1D case d times, using a structuring element decomposition approach [7]. Also, the d-dimensional max and min filters can be computed simultaneously in $(2 + o(1))d$ comparisons per sample. These are the currently best known upper bounds for d-dimensional max and min filters, on both worst-case inputs and independently and identically distributed (i.i.d.) inputs. There are two natural and interesting questions proposed in the literature:

– (Proposed in [14]) Can we simultaneously compute max and min filters faster than simply applying the algorithm for max (or min) filter twice?
– (Proposed in [5]) Can we do better in the case of i.i.d. inputs?

1.1 Our Contributions

In this work, we study d-dimensional max and min filters on i.i.d. inputs with fixed dimension $d \geq 1$, and answer both questions above in the affirmative.

Table 1. The number of comparisons per sample used in various algorithms for d-dimensional max and/or min filters on i.i.d. inputs. An entry marked with '$\star$' means that the same bound also applies to worst-case inputs.

	max (or min) filter	max and min filters together
Pitas [9]	$O(d \log p)$ *	$O(d \log p)$ *
van Herk [11]; Gil and Werman [6]	$(3 - o(1))d$ *	$(6 - o(1))d$ *
Gevorkian et al. [4]	$(2.5 - o(1))d$	$(5 - o(1))d$
Gil and Kimmel [5]	$(1.25 + o(1))d$	$(2 + o(1))d$
Yuan and Atallah [14]	$(1 + o(1))d$ *	$(2 + o(1))d$ *
This work	$1 + o(1)$	$1.5 + o(1)$

More specifically, we present an algorithm for computing d-dimensional max and min filters simultaneously on i.i.d. inputs that uses $1.5 + o(1)$ expected comparisons per sample, for every fixed $d \geq 1$. To the best of our knowledge, ours is the first algorithm for d-dimensional max and min filters on i.i.d. inputs that gets rid of the dependence on d in (the dominating term of) the complexity. In particular, for the special 1D case, our algorithm uses $1.5 + o(1)$ comparisons per sample for computing 1D max and min filters simultaneously, improving upon the previous best upper bound of $2 + o(1)$ (which can be obtained by applying the algorithm of Yuan and Atallah [14] twice). We also note that our algorithm is optimal up to the $o(1)$ term, due to the fact that at least $1.5n$ comparisons are required for computing both the maximum and minimum elements of a given sequence of length n with i.i.d. input elements (see [2]).

As a by-product of our result, we can also compute the d-dimensional max (or min) filter on i.i.d. inputs using $1 + o(1)$ expected comparisons per sample, which matches the bound for the 1D case due to Yuan and Atallah [14] and improves the previous best result of $(1 + o(1))d$ for every fixed $d \geq 2$. Table 1 compares our results with some previous work on d-dimensional max and/or min filters with i.i.d. inputs.

We remark that our contributions are mainly theoretical, i.e., we give an *asymptotically* optimal algorithm for computing d-dimensional max and min filters. Due to the heavy uses of iterations and partitions over the input array, our algorithm may not perform better than the existing algorithms on instances with low dimensions or small window sizes. More details of our algorithm will be shown later. Moreover, we omit most technical proofs due to space limitations.

1.2 Related Work

Pitas [9] showed how to compute the 1D max filter using $O(\log p)$ comparisons per sample, which is the first work that improves upon the trivial bound of $p-1$. The first algorithm for 1D max filter that achieves a constant upper bound was proposed independently by van Herk [11] and Gil and Werman [6], which requires $3 - o(1)$ comparisons per sample. Their algorithm (HGW for short) fits implicitly into a two-stage framework, which consists of a preprocessing stage and a merging stage. This framework was followed by almost all the later works in this line, which achieve their improvements by reducing the complexity of the preprocessing stage and/or the merging stage.

Gil and Kimmel [5] presented an algorithm (which we will call GK for short) for 1D max filter that uses $1.5 + o(1)$ comparisons per sample, which is obtained by improving both the preprocessing and merging stages of HGW. Recently, Yuan and Atallah [14] further reduced the complexity to $1 + o(1)$ comparisons per sample, which is the best possible up to the lower order term. Their algorithm (YA for short) also indicates that the d-dimensional max filter can be computed by $(1 + o(1))d$ comparisons per sample. All the aforementioned algorithms can be trivially adapted for the min filter as well.

Previous works are also concerned with the case where the input array is composed of independently and identically distributed (i.i.d.) elements. Gevorkian, Astola, and Atourian [4] gave an algorithm that uses $2.5 - o(1)$ expected comparisons per sample to compute the 1D max filter on i.i.d. inputs. The GK algorithm [5] can compute the 1D max filter in $1.25 + o(1)$ expected comparisons per sample on i.i.d. inputs. They also showed that the 1D max and min filters can be computed simultaneously using $2 + o(1)$ comparisons per sample, which is better than running their algorithm twice. Their result for 1D max filter was superseded by the YA algorithm [14], which only makes $1 + o(1)$ comparisons per sample even on worst-case inputs (and thus also on i.i.d. inputs).

The problem of computing min (or max) filter can also be reduced to the Range Minimum Query (RMQ) problem. In the RMQ problem, we are given an input array and are required to preprocess the array, so that each query asking for the minimum element within some window can be answered efficiently. When the input to RMQ is a 1D array, there is an algorithm with linear preprocessing time and space that can answer each query in constant time [3]. The idea is to reduce the problem to the Nearest Common Ancestor problem [8] on the Cartesian tree [12]. Using this approach, the 1D min (or max) filter can be computed using at most 2 comparisons per sample. The RMQ problem on high-dimension input arrays was studied by Yuan and Atallah [13]. For each fixed $d \geq 2$, their algorithm preprocesses the d-dimensional input array in linear time, and answers each query using $2^d - 1$ comparisons.

2 Preliminaries

In this section, we introduce some notation used in the paper. Given two positive integers p and d, let $[p] := \{1, 2, \ldots, p\}$ and $[p]^d := \{(i_1, i_2, \cdots, i_d) \mid 1 \leq i_1, \cdots, i_d \leq p\}$. For an array A of length n, we write $A[i]$ to denote its i-th element, and use $A[i \sim j]$ to denote the subarray of A that starts from $A[i]$ and ends with $A[j]$. The array A can also be written as $A[1 \sim n]$. The maximum and minimum over elements in A are denoted by $\max A$ and $\min A$, respectively. For example, $\max A[3 \sim 5] = \max\{A[3], A[4], A[5]\}$. These notations can be naturally generalized to the high-dimension case. For example, given a 2D array

$B = B[1 \sim 2, 1 \sim 3]$, we have $\min B[1 \sim 1, 2 \sim 3] = \min\{B[1,2], B[1,3]\}$. We use log to denote the logarithm with base 2.

Windows and Max/Min Filters

We next formally introduce the d-dimensional max/min filter problems studied in this paper. We start with the special case $d = 1$, which has been extensively studied before.

Let $X = X[1 \sim n]$ be an input array (or sequence) of length n, and $p \geq 2$ be an integer. We assume that the elements of the array are drawn from a totally ordered set $\mathscr{S}$ (for example, the set of integers). The *1D max filter* of X is an array $Y = Y[1 \sim n - p + 1]$ where $Y[i] = \max X[i \sim i + p - 1]$ for all $1 \leq i \leq n - p + 1$. Similarly, the 1D min filter of X is an array $Z = Z[1 \sim n - p + 1]$ with $Z[i] = \min X[i \sim i + p - 1]$ for $1 \leq i \leq n - p + 1$. Equivalently, the max (or min) filter can be regarded as the set of maximum (or minimum) elements over a sliding window of length p in X.

We now generalize the concepts to higher dimensions. Suppose $X = X[1 \sim n, \cdots, 1 \sim n]$ is a d-dimensional array where each dimension has length n, and $p \geq 2$ is an integer. For each $(i_1, \cdots, i_d) \in [n - p + 1]^d$, we define

$$\mathcal{W}_{i_1,\cdots,i_d} = X[i_1 \sim i_1 + p - 1, \cdots, i_d \sim i_d + p - 1],$$

which is called a *window with edge length* p in X. We also call $(i_1, \cdots, i_d)$ the *starting index* of $\mathcal{W}_{i_1,\cdots,i_d}$. (For example, if $d = 1, n = 4$ and $p = 3$, then there are two windows of edge length p, namely $\mathcal{W}_1 = X[1 \sim 3]$ and $\mathcal{W}_2 = X[2 \sim 4]$.)

Let $\mathscr{W}(X)$ denote the set of all windows with edge length p in X, i.e., $\mathscr{W}(X) = \{\mathcal{W}_{i_1,\cdots,i_d} \mid (i_1, \cdots, i_d) \in [n - p + 1]^d\}$. The *max filter* of X (with edge length p) is $\{\max \mathcal{W} \mid \mathcal{W} \in \mathscr{W}(X)\}$, and the *min filter* of X (with edge length p) is $\{\min \mathcal{W} \mid \mathcal{W} \in \mathscr{W}(X)\}$. (Here we regard them as multisets, i.e., two elements in the set with the same value are still considered as distinct elements.)

In the *d-dimensional max filter problem*, we are given a d-dimensional array and an integer p, and the goal is to compute the max filter with edge length p. The *d-dimensional min filter problem* can be defined analogously. The *d-dimensional max and min filters problem* is to compute the max and min filters simultaneously.

Complexity Measure

We are interested in designing algorithms for computing max and/or min filters with low complexity in the *comparison model*, i.e., only the number of *comparisons between elements in* $\mathscr{S}$ is counted. The other computational costs, like comparisons between indices used in an iteration, are not taken into consideration. Following convention in the literature, the complexity is measured by the number of *comparisons per sample*, i.e., the ratio of the total number of comparisons over the number of output terms.

We focus on the asymptotic complexity; that is, we assume $n \to \infty$ and $p \to \infty$. In accordance with usual situations and previous studies, we also assume

n is very large compared to p, i.e., $n/p \to \infty$ as $n \to \infty$. However, in this paper we will only consider the case where d is a fixed integer, i.e., it stays fixed as n and p tend to infinity.

Notice that the d-dimensional max (or min) filter problem requires $(n-p+1)^d$ outputs. By our assumption on the parameters, we have

$$n^d \geq (n-p+1)^d = n^d(1-(p-1)/n)^d \geq n^d(1-d(p-1)/n) = (1-o(1))n^d.$$

Therefore, when calculating the number of comparisons per sample, if we replace the denominator $(n-p+1)^d$ with n^d, the result will increase by at most a $1+o(1)$ factor, which does not affect the asymptotic form. Thus, for simplicity of expressions, we may assume that we have n^d output terms.

3 Our Results

Our main result in this paper is as follows.

Theorem 1. *For every fixed integer $d \geq 1$, there is an algorithm for computing the d-dimensional max and min filters simultaneously that makes $1.5+o(1)$ expected comparisons per sample on i.i.d. inputs.*

As pointed out before, the previously best known algorithm for the d-dimensional max and min filters requires $(2+o(1))d$ comparisons per sample, on both worst-case and i.i.d. inputs. Thus, our algorithm is the first one that gets rid of the dependence on d in (the dominating term of) the complexity. In particular, for the 1D case, our algorithm improves the previous best upper bound of $2+o(1)$ to $1.5+o(1)$.

We also note that our bound is optimal up to the $o(1)$ term, which can be easily proved using the following fact (see [2]): given an input sequence of length n, computing both the maximum and the minimum of the whole input cannot be done in less than $1.5n$ comparisons, even for i.i.d. inputs.

As a by-product of our algorithm, we also improve the complexity for computing d-dimensional max (or min) filter on i.i.d. inputs, removing for the first time the dependence on d in (the dominating term of) the complexity.

Theorem 2. *For every fixed integer $d \geq 1$, there is an algorithm for computing the d-dimensional max (or min) filter that uses $1+o(1)$ expected comparisons per sample on i.i.d. inputs.*

To solve the problem, we follow a two-stage framework similar to that mentioned before. Note that all previous approaches under this framework only work for the 1D case. We develop novel preprocessing and merging stages that work for any constant dimension, whose efficiency is characterized by the following two theorems.

Theorem 3. *For any fixed $d \geq 1$, our preprocessing stage for d-dimensional max and min filters can be completed using $1.5+o(1)$ expected comparisons per sample on i.i.d. inputs. If only max (or min) filter is required, it can be finished using $1+o(1)$ expected comparisons per sample.*

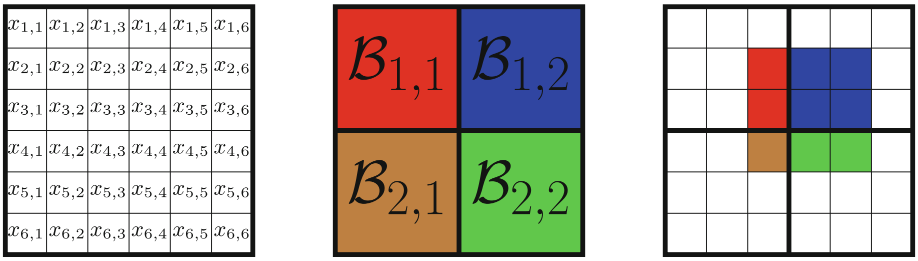

$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$\mathcal{B}_{1,1}$
$\mathcal{B}_{1,2}$
$\mathcal{B}_{2,1}$
$\mathcal{B}_{2,2}$

Fig. 2. An instance with $d = 2$, $n = 6$, and $p = 3$. (For space reasons we use $x_{i,j}$ to denote $X[i, j]$.) The middle picture represents the division of the input array into $(n/p)^d = 4$ blocks in the preprocessing stage. The right picture reflects how the maximum of a $p \times p$ window is computed in the merging stage; each such window can be decomposed into (at most) $2^d = 4$ basic hyperrectangles.

Theorem 4. *For any fixed $d \geq 1$, our merging stage for d-dimensional max and min filters can be completed using $o(1)$ comparisons per sample.*

Theorem 3, combined with the merging stage of the GK algorithm [5], directly yields an algorithm for 1D max and min filters using $1.5 + o(1)$ comparisons per sample. However, their merging stage does not directly apply to the case $d \geq 2$. To deal with this issue, we also design a new merging stage that works for all constant dimensions using only $o(1)$ comparisons per sample.

4 Two-Stage Framework for *d*-dimensional Max and Min Filters

In this section, we present the two-stage framework for computing the d-dimensional max and min filters on i.i.d. inputs.

Assume the input d-dimensional array is $X[1 \sim n, \cdots, 1 \sim n]$, and the edge-length of a window is p. The elements of X are independently and identically distributed. Recall that the set of windows with edge length p is $\mathscr{W}(X) = \{\mathcal{W}_{i_1,\cdots,i_d} \mid 1 \leq i_1, \cdots, i_d \leq n - p + 1\}$, where $\mathcal{W}_{i_1,\cdots,i_d} = X[i_1 \sim i_1 + p - 1, \cdots, i_d \sim i_d + p - 1]$. The goal is to compute the max filter $\{\max \mathcal{W} \mid \mathcal{W} \in \mathscr{W}(X)\}$ and min filter $\{\min \mathcal{W} \mid \mathcal{W} \in \mathscr{W}(X)\}$. We write $\mathscr{W} = \mathscr{W}(X)$ for notational simplicity.

Intuitively, we can regard (indices of) the input array as a d-dimensional hypercube of edge length n, with the indices considered as the coordinates. (Note that a 1D hypercube is just a line segment, and a 2D hypercube is a square. See Figure 2 for examples of $d = 2$.)

4.1 The Preprocessing Stage

The structure of our preprocessing stage is summarized as Algorithm 1. The details of the algorithm are explained below.

Algorithm 1. The preprocessing stage

```
1 Divide the entire array into a set of $(n/p)^d$ disjoint blocks
  $\mathscr{B} = \{\mathcal{B}_{i_1,\cdots,i_d} \mid 1 \le i_1, \cdots, i_d \le n/p\}$, where each block is a smaller
  $d$-dimensional array with length $p$ in each dimension.
2 for each block $\mathcal{H} \in \mathscr{B}$ do
3     Compute $\max \mathcal{R}$ and $\min \mathcal{R}$ for every basic hyperrectangle $\mathcal{R} \in \mathscr{R}(\mathcal{H})$ using
      Algorithm 2.
4 end
```

Line 1 of Algorithm 1

At the start of Algorithm 1, we partition the entire array into $(n/p)^d$ smaller d-dimensional arrays each of which is called a *block*. More formally, the set of blocks is

$$\mathscr{B} = \{\mathcal{B}_{i_1,\cdots,i_d} \mid 1 \le i_1, \cdots, i_d \le n/p\},$$

where, for all $1 \le i_1, \cdots, i_d \le n/p$,

$$\mathcal{B}_{i_1,\cdots,i_d} = X[(i_1-1)p+1 \sim i_1 p, (i_2-1)p+1 \sim i_2 p, \cdots, (i_d-1)p+1 \sim i_d p].$$

(We assume w.l.o.g. that n is a multiple of p; otherwise some blocks near the boundary may have edge length smaller than p, but the asymptotic form of our results will not be affected.) Note that each block is also a window in $\mathscr{W}$.

See Figure 2 for an example of this hypercube-division idea in 2D.

Corners and Basic Hyperrectangles

We next explain what a basic hyperrectangle means (which is referred to in line 3 of Algorithm 1). Consider a particular block $\mathcal{H} = \mathcal{B}_{i_1,\cdots,i_d} \in \mathscr{B}$. Clearly, when regarded as a hypercube, $\mathcal{H}$ contains 2^d "corners." (For instance, in the 1D case, the two endpoints of a line segment are its corners.) We are interested in those (integer-coordinated) hyperrectangles lying inside $\mathcal{H}$ that contain at least one such corner, which are called *basic hyperrectangles*. To be rigorous, we give formal definitions of the corners and basic hyperrectangles associated with a block.

Definition 1 (Corners). *Let $\mathcal{H} = \mathcal{H}[1 \sim q, \cdots, 1 \sim q]$ be a d-dimensional array of edge length q. Define the* corner set *of $\mathcal{H}$ as*

$$\mathcal{C}(\mathcal{H}) := \{(j_1, j_2, \cdots, j_d) \mid j_1, j_2, \cdots, j_d \in \{1, q\}\}.$$

(Note that it is a set of indices.) Each element of $\mathcal{C}(\mathcal{H})$ is called a corner *of $\mathcal{H}$. Clearly $|\mathcal{C}(\mathcal{H})| = 2^d$.*

Definition 2 (Basic Hyperrectangles). *Let $\mathcal{H} = \mathcal{H}[1 \sim q, \cdots, 1 \sim q]$ be a d-dimensional array of edge length q. A* basic hyperrectangle *inside $\mathcal{H}$ is a sub-array of $\mathcal{H}$ of the form*

$$\mathcal{H}[\mathsf{lb}_1 \sim \mathsf{ub}_1, \mathsf{lb}_2 \sim \mathsf{ub}_2, \cdots, \mathsf{lb}_d \sim \mathsf{ub}_d],$$

where for any $1 \leq t \leq d$, *we require* $1 \leq \mathsf{lb}_t \leq \mathsf{ub}_t \leq q$, *and* $\mathsf{lb}_t = 1$ or $\mathsf{ub}_t = q$. *(Intuitively, at least one of* lb_t *and* ub_t *"hits the boundary.")*

Clearly each basic hyperrectangle inside $\mathcal{H}$ *contains at least one corner of* $\mathcal{H}$. *For each corner* $c \in \mathcal{C}(\mathcal{H})$, *let* $\mathscr{R}(\mathcal{H}, c)$ *denote the set of basic hyperrectangles containing* c. *Let* $\mathscr{R} = \bigcup_{c \in \mathcal{C}(\mathcal{H})} \mathscr{R}(\mathcal{H}, c)$ *be the set of all basic hyperrectangles inside* $\mathcal{H}$.

Let us see an example. For the 2D case, see the instance in Figure 2 which shows an instance of the 1D case, the corner set of the block $\mathcal{B}_1$ is $\mathcal{C}(\mathcal{B}_1) = \{1, 3\}$. For each corner of a block, there are 9 basic hyperrectangles containing the corner. For example, the basic hyperrectangles containing the top-right corner of $\mathcal{B}_{1,2}$ are $X[1 \sim i, j \sim 6]$ for all $1 \leq i \leq 3$ and $4 \leq j \leq 6$.

The following propositions are immediate from the definitions.

Proposition 1. *For each block* $\mathcal{H} \in \mathscr{B}$ *and each corner* $c \in \mathcal{C}(\mathcal{H})$, *there are exactly* p^d *basic hyperrectangles containing* c, *i.e.,* $|\mathscr{R}(\mathcal{H}, c)| = p^d$.

Proposition 2. *For each block* $\mathcal{H} \in \mathscr{B}$, $|\mathscr{R}(\mathcal{H})| = |\bigcup_{c \in \mathcal{C}(\mathcal{H})} \mathscr{R}(\mathcal{H}, c)| \leq (2p)^d$.

Lines 2–4 of Algorithm 1

We now turn back to Algorithm 1. In lines 2–4, Algorithm 1 computes $\max \mathcal{R}$ and $\min \mathcal{R}$ for every basic hyperrectangle $\mathcal{R}$. The task is decomposed into subproblems, i.e., for each block $\mathcal{H}$ (the **for** loop in line 2), it computes $\max \mathcal{R}$ and $\min \mathcal{R}$ for all basic hyperrectangles in $\mathscr{R}(\mathcal{H})$ (line 3) using Algorithm 2. This is the most substantial part of this stage, for which we design an algorithm that uses only $1.5 + o(1)$ comparisons per sample on i.i.d. inputs (or $1 + o(1)$ comparisons per sample if only the max (or min) filter is needed). The details of this part will be given later in this section.

After finishing the computations for all the $(n/p)^d$ blocks, our preprocessing stage is complete.

Remark. We note that, for the special case $d = 1$, our preprocessing stage actually has the same goal with that of GK [5], but we use a different approach. We explain below why their tasks coincide. In the GK framework, the input sequence is first divided into n/p subsequences each of size p, which are exactly the blocks in our definition. Then, they compute the maximums (and minimums) over all prefixes and suffixes of each block; these prefixes and suffixes are precisely the basic hyperrectangles in our notation. Thus, when restricted on 1D inputs, the function of our preprocessing stage coincides with that of the existing framework. However, our algorithm for computing the prefix/suffix maximums and minimums is different from the previous ones (GK [5] and YA [14]), and is more efficient on i.i.d. inputs.

Algorithm 2. Algorithm for computing $\{\max\mathcal{R}, \min\mathcal{R} \mid \mathcal{R} \in \mathscr{R}(\mathcal{H})\}$ for fixed $\mathcal{H} \in \mathscr{B}$

```
1 Divide $\mathcal{H}$ into a collection of $p^{d/2}$ sub-blocks
  $\mathsf{sub}(\mathcal{H}) = \{\mathcal{H}_{k_1,\cdots,k_d} \mid 1 \le k_1, \cdots, k_d \le \sqrt{p}\}$, each of which is a $d$-dimensional
  hypercube with edge length $\sqrt{p}$.
2 for each sub-block $\mathcal{H}_{k_1,\cdots,k_d} \in \mathsf{sub}(\mathcal{H})$ do
3     Compute $z^{\max}_{k_1,\cdots,k_d} = \max\mathcal{H}_{k_1,\cdots,k_d}$ and $z^{\min}_{k_1,\cdots,k_d} = \min\mathcal{H}_{k_1,\cdots,k_d}$.
4 end
5 for each corner $c \in \mathcal{C}(\mathcal{H})$ do
6     Compute $\max R$ and $\min R$ for all $\mathcal{R} \in \mathscr{R}(\mathcal{H}, c)$.
7 end
```

Dealing with Basic Hyperrectangles

In this part, we show how to compute the maximums and minimums over all basic hyperrectangles inside a block. Fix a block $\mathcal{H} = \mathcal{B}_{i_1,\cdots,i_d}$. Recall the notation that $\mathcal{C}(\mathcal{H})$ is the corner set of $\mathcal{H}$, $\mathscr{R}(\mathcal{H})$ is the set of basic hyperrectangles inside $\mathcal{H}$, and $\mathscr{R}(\mathcal{H}, c)$ is the set of basic hyperrectangles containing the corner c. Our main result in this section is as follows.

Theorem 5. *For each block $\mathcal{H}$, we can compute $\{\max\mathcal{R} \mid \mathcal{R} \in \mathscr{R}(\mathcal{H})\}$ and $\{\min\mathcal{R} \mid \mathcal{R} \in \mathscr{R}(\mathcal{H})\}$ using $(1.5 + o(1))p^d$ expected comparisons. If only $\{\max\mathcal{R} \mid \mathcal{R} \in \mathscr{R}(\mathcal{H})\}$ or $\{\min\mathcal{R} \mid \mathcal{R} \in \mathscr{R}(\mathcal{H})\}$ are needed, the computation can be done in $(1 + o(1))p^d$ expected comparisons.*

Plugging this result into Algorithm 1, we can perform the preprocessing stage using $1.5 + o(1)$ expected comparisons per sample (or $1 + o(1)$ expected comparisons per sample if only the max filter is required), and thus Theorem 3 is proved. We present our algorithm that achieves the bound in Theorem 5 as Algorithm 2.

4.2 The Merging Stage

The goal of this stage is to obtain the max and min filters using the information gathered in the preprocessing stage, i.e., the maximums and minimums of all the basic hyperrectangles. Clearly, each window $\mathcal{W} \in \mathscr{W}$ is the (disjoint) union of at most 2^d basic hyperrectangles from distinct blocks (see Figure 2). This observation directly yields a (trivial) algorithm for the merging stage that uses $2(2^d-1)$ comparisons per sample: to compute $\max\mathcal{W}$, just use 2^d-1 comparisons to find out the maximum among the 2^d maximums of basic hyperrectangles, and another $2^d - 1$ comparisons for the minimums.

To reduce the merging complexity to $o(1)$, our idea is to use a binary search procedure similar to the merging stage of GK [5] which works only for the 1D case. The basic idea is as follows. As noted before, each window $\mathcal{W} \in \mathscr{W}$ is the disjoint union of (at most) 2^d basic hyperrectangles. On each of the d dimensions, it "crosses" (at most) two "large" blocks, each of which consists of 2^{d-1} normal blocks. (See Figure 2 for an example. Consider the vertical dimension. Then the two large blocks are $\mathcal{B}_{1,1} \cup \mathcal{B}_{1,2}$ and $\mathcal{B}_{2,1} \cup \mathcal{B}_{2,2}$. The window shown in the right picture crosses the boundary between these two large blocks.)

We want to determine which of the two large blocks contains the maximum element of $\mathcal{W}$. We show that it can be done using a binary search procedure in $o(1)$ comparisons per sample on every dimension. Then, we know exactly which basic hyperrectangle contains the maximum element of $\mathcal{W}$. (For example, in Figure 2, if we know that the maximum of the window is in both the top part and the left side, then it is in the top-left red basic hyperrectangle.) Since the maximum over this basic hyperrectangle has been obtained in the preprocessing stage, we can determine $\max \mathcal{W}$ without further comparisons. In consequence, our merging stage can be finished in $o(1)$ comparisons per sample. More details are exhibited below. (Note that Theorem 4 does not require the input elements to be i.i.d., and thus it may be useful in possible future algorithms dealing with worst-case inputs.)

Generalized Binary Search

This part is devoted to proving Theorem 4. We will give our merging stage for d-dimensional max and min filters that only uses $o(1)$ comparisons per sample. Our algorithm is a binary search procedure generalizing that of GK [5]. The idea is to find out, over each of the d dimensions, on which side the maximum of a window lies. After gathering these information, we can find a basic hyperrectangle that contains a maximum element of the window, and thus can determine the maximum of the window using the information obtained in the preprocessing stage.

We give an outline of our merging stage for the max filter in Algorithm 3, which can be trivially adapted for the min filter by changing the objective from max to min. To make our statement clearer, we need to introduce some more notations. Consider a window $\mathcal{W} = \mathcal{W}_{i_1,\cdots,i_d} \in \mathscr{W}$. For each dimension $t \in [d]$, let $l_t = \lceil i_t/p \rceil$ and $r_t = l_t + 1 = \lceil i_t/p \rceil + 1$. We define

$$\mathcal{B}(\mathcal{W}) = \{\mathcal{B}_{b_1,\cdots,b_d} \mid (\forall t \in [d])\ b_t \in \{l_t, r_t\}\}.$$

Clearly $|\mathcal{B}(\mathcal{W})| = 2^d$. By the definition of blocks, we know that $\mathcal{W}$ is totally contained in the union of the 2^d blocks in $\mathcal{B}(\mathcal{W})$.

Next, for every dimension $t \in [d]$, we define

$$\mathcal{B}_t^{\mathsf{low}}(\mathcal{W}) = \{\mathcal{B}_{b_1,\cdots,b_d} \in \mathcal{B}(\mathcal{W}) \mid b_t = l_t\} \tag{1}$$

and

$$\mathcal{B}_t^{\mathsf{high}}(\mathcal{W}) = \{\mathcal{B}_{b_1,\cdots,b_d} \in \mathcal{B}(\mathcal{W}) \mid b_t = r_t\}. \tag{2}$$

Intuitively, $\mathcal{B}_t^{\mathsf{low}}(\mathcal{W})$ and $\mathcal{B}_t^{\mathsf{high}}(\mathcal{W})$ partition the blocks into two sides, a "low" side and a "high" side, according to their indices on the t-th dimension. We take

Figure 2 as an example. Consider the window $\mathcal{W}_{2,3}$ shown in the right picture. In this case we have $l_1 = 1, r_1 = 2, l_2 = 1, r_2 = 2$. Then,

$$\mathcal{B}(\mathcal{W}) = \{\mathcal{B}_{1,1}, \mathcal{B}_{1,2}, \mathcal{B}_{2,1}, \mathcal{B}_{2,2}\}.$$

On the first dimension (or, the vertical dimension), we have $\mathcal{B}_1^{\mathsf{low}}(\mathcal{W}) = \{\mathcal{B}_{1,1}, \mathcal{B}_{1,2}\}$ and $\mathcal{B}_1^{\mathsf{high}}(\mathcal{W}) = \{\mathcal{B}_{2,1}, \mathcal{B}_{2,2}\}$. On the second dimension (or, the horizontal dimension), we have $\mathcal{B}_2^{\mathsf{low}}(\mathcal{W}) = \{\mathcal{B}_{1,1}, \mathcal{B}_{2,1}\}$ and $\mathcal{B}_2^{\mathsf{high}}(\mathcal{W}) = \{\mathcal{B}_{1,2}, \mathcal{B}_{2,2}\}$.

We say window $\mathcal{W}$ is *low on dimension t* if (at least one of) its maximum element(s) belongs to some block in $\mathcal{B}_t^{\mathsf{low}}(\mathcal{W})$, and say $\mathcal{W}$ is *high on dimension t* otherwise.

Analysis of Algorithm 3

We now analyze Algorithm 3. In lines 1–3, it aims to decide whether $\mathcal{W}$ is low or high on each dimension for all $\mathcal{W} \in \mathscr{W}$. The idea is as follows: For any window $\mathcal{W} \in \mathscr{W}$, if we know whether $\mathcal{W}$ is low or high on every dimension, we can find out a block, say $\mathcal{H}$, which contains a maximum element of $\mathcal{W}$. Note that $\mathcal{W} \cap \mathcal{H}$ is a basic hyperrectangle inside $\mathcal{H}$, whose maximum has been obtained in the preprocessing stage. Thus, we can determine $\max \mathcal{W} = \max(\mathcal{W} \cap \mathcal{H})$ without further comparisons. This is what the algorithm does in lines 4–7. (For example, in Figure 2, if we know the maximum element of the window is low in the vertical dimension and high in the horizontal dimension, then it must belong to the top-right blue basic hyperrectangle inside $\mathcal{B}_{1,2}$.)

Algorithm 3. Merging stage for d-dimensional max filters

```
  Output: Compute $\max \mathcal{W}$ for all $\mathcal{W} \in \mathscr{W}$.
1 for each dimension $r \in [d]$ do
2     Determine whether the windows in $\mathscr{W}$ are low or high on dimension $r$.
3 end
4 for each window $\mathcal{W} \in \mathscr{W}$ do
5     Find the block $\mathcal{H}$ that contains a maximum element of $\mathcal{W}$. Then $\mathcal{W} \cap \mathcal{H}$ is a
      basic hyperrectangle inside $\mathcal{H}$.
6     Set $\max \mathcal{W} \leftarrow \max(\mathcal{W} \cap \mathcal{H})$.
7 end
8 return $\{\max \mathcal{W} \mid \mathcal{W} \in \mathscr{W}\}$
```

By the above analysis, the comparisons between elements that happened in Algorithm 3 are only due to lines 1–3. The next theorem shows that we do not need many comparisons for these steps.

Theorem 6. *For any dimension $t \in [d]$, we can decide using $O(n^p \log(p)/p)$ comparisons whether $\mathcal{W}$ is low or high on dimension t for all $\mathcal{W} \in \mathscr{W}$ simultaneously.*

Since there are only $d = O(1)$ dimensions, by Theorem 6 we can implement lines 1–3 of Algorithm 3 using $O(n^p \log(p)/p)$ comparisons, or $O(\log(p)/p) = o(1)$ comparisons per sample. Hence, Theorem 4 is proved.

5 Conclusions

In this work, we showed how to compute the max and min filters simultaneously on constant-dimension arrays with i.i.d. elements using $1.5 + o(1)$ comparisons per sample, which is optimal up to the lower order term. Also, the d-dimensional max filter itself can be computed using $1 + o(1)$ comparisons per sample on i.i.d. inputs for fixed d. However, on worst-case inputs, the best known algorithm for d-dimensional max (or min) filter still requires $(1 + o(1))d$ comparisons per sample. An interesting open question is whether this dependence of d in the complexity is intrinsic. Also, for the special 1D case, can we improve the worst-case comparison bound for simultaneous max and min filters from $2 + o(1)$ to less than 2, or even $1.5 + o(1)$?

References

1. Coltuc, D.: Mathematical complexity of running filters on semi-groups and related problems. IEEE Transactions on Signal Processing **56**(7), 3191–3197 (2008)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, 3rd edn. MIT press (2009)
3. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC), pp. 135–143 (1984)
4. Gevorkian, D.Z., Astola, J.T., Atourian, S.M.: Improving Gil-Werman algorithm for running min and max filters. IEEE Transactions on Pattern Analysis and Machine Intelligence **19**(5), 526–529 (1997)
5. Gil, J., Kimmel, R.: Efficient dilation, erosion, opening, and closing algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(12), 1606–1617 (2002)
6. Gil, J., Werman, M.: Computing 2-D min, median, and max filters. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(5), 504–507 (1993)
7. Haralick, R.M., Sternberg, S.R., Zhuang, X.: Image analysis using mathematical morphology. IEEE Transactions on Pattern Analysis and Machine Intelligence **9**(4), 532–550 (1987)
8. Harel, D., Tagjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM Journal on Computing **13**(2), 338–355 (1984)
9. Pitas, I.: Fast algorithms for running ordering and max/min calculation. IEEE Transactions on Circuits and Systems **36**(6), 795–804 (1989)
10. Soille, P.: Morphological Image Analysis: Principles and Applications. Springer-Verlag New York, Inc. (2003)
11. van Herk, M.: A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. Pattern Recognition Letters **13**(7), 517–521 (1992)

12. Vuillemin, J.: A unifying look at data structures. Communications of the ACM **23**(4), 229–239 (1980)
13. Yuan, H., Atallah, M.J.: Data structures for range minimum queries in multidimensional arrays. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 150–160 (2010)
14. Yuan, H., Atallah, M.J.: Running max/min filters using 1+o(1) comparisons per sample. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(12), 2544–2548 (2011)

Model Checking MSVL Programs Based on Dynamic Symbolic Execution

Zhenhua Duan, Kangkang Bu, Cong Tian(✉), and Nan Zhang

Institute of Computing Theory and Technology, and ISN Laboratory, Xidian University, Xi'an 710071, China
ctian@mail.xidian.edu.cn

Abstract. In this paper, we propose a DSE based model checking approach (DSE-MC) for verifying programs written in Modelling, Simulation and Verification Language (MSVL) [1,3]. For doing so, we adopt a DSE method to execute an MSVL program to generate a symbolic execution tree (SEtree) which is used as the abstract model of the program. Further, a property to be verified is specified by a Propositional Projection Temporal Logic (PPTL) formula [8,13]. To check whether or not the program satisfies the property, first the SEtree and the negation of the property are both described in Labelled Normal Form Graphs (LNFGs) [21], then the product of two LNFGs is produced. As a result, a counter example is encountered if the product is not empty. Otherwise, we cannot determine if the program satisfies the property. In this case, the verification process could be restarted with new inputs. In this way, a software system written in C can also be verified since the C program can be transformed to an MSVL program automatically by using toolkit MSV [19] developed by us.

Keywords: DSE · Constraints · Temporal logic · MSVL · Model checking

1 Introduction

Symbolic execution (SE) is first proposed for program testing [2] that uses symbolic values as inputs instead of real data. It represents the values of program variables as symbolic expressions which are functions of the input symbolic values. Subsequently, SE has been used for bugs finding [4] and verification condition generation [5,6], among others. Dynamic symbolic execution (DSE) [7,9] enhances traditional symbolic execution [2] by combing concrete execution and symbolic execution together. Essentially, DSE repeatedly runs a program both concretely and symbolically. After each run, all the branches off the execution path are collected, and then one of them is selected to generate new inputs for the next run to explore a new path. A symbolic execution tree (SEtree) depicts

The research is supported by the NSFC under Grant No. 61133001, 61322202, 61420106004 and 91418201.

all executed paths during the symbolic execution. A path condition is maintained for each path and it is a formula over the symbolic inputs built by accumulating constraints satisfied by those inputs in order for execution to follow that path. A path is infeasible if its path condition is unsatisfiable. Otherwise, the path is feasible.

Recently, symbolic execution has been used for software verification [10–12] as an alternative to the existing model checking techniques based on Counter Example Guided Abstraction Refinement (CEGAR) [14,15] method. Essentially, the general technique followed by symbolic execution-like tools starts with the concrete model of the program and then, the model is checked for the desired property via symbolic execution by proving that all paths to certain error nodes are infeasible (i.e., error nodes are unreachable). For instance, when DSE is applied to checking a program against a regular property specified by an automaton [16] , an execution path satisfies the property if the sequence of the events in the path is accepted by the automaton, and this path is an accepted path. DSE improves the coverage through symbolic execution, and avoids false alarms by actually running the program. More importantly, DSE can use the information of the concrete execution to simplify complex symbolic reasonings and handle the environment problem. However, the disadvantages of DSE based verification are obvious. The first challenge is the exponential number of symbolic paths. The approaches of [10–12] tackle successfully this fundamental problem by eliminating from the concrete model those facts which are irrelevant or too-specific for proving the unreachability of the error nodes. However, the problem is still remained. The second problem is that different notations are used to specify programs, symbolic execution trees and properties of programs to be verified. For instance, usually, a program is written in C or Java language, an SEtree is described by a graph and a property of the program is specified by an automaton [16] or temporal logic formula [17]. This makes the verification much complicated. The third problem is that some probes need to be inserted in the context of the program, and they are normally done manually.

In this paper, we are motivated to propose a DSE based model checking approach (DSE-MC) for verifying programs written in Modelling, Simulation and Verification Language (MSVL) [1,3]. For doing so, we adopt a DSE method to execute an MSVL program to generate an SEtree which is later used as the abstract model of the program. Further, a property to be verified is specified by a Propositional Projection Temporal Logic (PPTL) formula [8,13]. To check whether or not the program satisfies the property, first the SEtree and the negation of the property are both described in Label Normal Form Graphs (LNFGs) [21], then the product of two LNFGs is produced. As a result, a counter example is encountered if the product is not empty. Otherwise, we do not know if the program satisfies the property. In this case, the verification process could be restarted with new inputs. This means that the proposed model checking approach is incomplete. In this aspect, it is some what similar to Bounded Model Checking (BMC) [18]. In this way, a software system written in C can also be verified since the C program can be transformed to an MSVL program automatically by toolkit MSV [19] developed by us. However, DSE-MC is an abstract model checking and is suitable for verifying software systems. Comparing with CEGAR approach, the advantages of DSE-MC lie in four aspects: (1) the abstract model (i.e., symbolic execution tree) is automatically generated by means of dynamic symbolic execution of MSVL programs and SMT solver (i.e., Z_3); (2)we do not need to insert probes into the context of an MSVL program as other DSE methods do; (3) a program and a property of the program are both in the same logic system (PTL) [8]; (4) the speed of DSE-MC is relatively quicker than CEGAR method since an SEtree is smaller than an abstract model generated using CEGAR.

The paper is organized as follows: the next section describes the Dynamic Symbolic Execution of MSVL programs including DSE algorithm, extended MSVL interpreter, search strategy and constraint solving, and generating symbolic execution tree. The DSE based model checking approach is formalized in detail in section 3. In section 4, a case study is given to show how DSE based model checking works. Finally, conclusion is drawn in section 5.

2 Dynamic Symbolic Execution of MSVL Programs

Dynamic symbolic execution of MSVL programs is an improvement of symbolic execution where symbolic execution is performed simultaneously with concrete execution. DSE includes three main phases. The first phase is to execute programs symbolically and concretely to record the satisfied path constraints. The second phase is to select a constraint to negate from the path constraints according to the search strategy. The third phase is to solve the modified path constraints by using SMT solver Z_3 to generate new inputs and to execute the program with the new inputs. By repeating this process, almost all feasible execution paths are swept through.

2.1 Dynamic Symbolic Execution Algorithm

Essentially, to implement the dynamic symbolic execution of a program, the most frequently used method is the instrumentation-based approach. This approach instruments a program by inserting probes which perform symbolic execution to extract symbolic path constraints, then executes the instrumented code on a standard compiler or an interpreter which is determined by the type of the programming language. With this approach, the instrumentation code performs symbolic execution, while the original code performs concrete execution, then, as a result, the whole program is executed both symbolically and concretely at the same time.

Unlike the instrumentation-based approach, the dynamic symbolic execution of MSVL programs does not require such instrumentation, instead, implements a non-standard interpreter of MSVL programs which extends the original MSVL interpreter[3]. We extends the standard, concrete execution semantics of MSVL programs with symbolic execution semantics. The symbolic information is stored in *symbolic memory map* $\boldsymbol{S}$ and *symbolic path constraints* $\boldsymbol{PCs}$, and is propagated as needed, during the symbolic execution. The *symbolic memory map* $\boldsymbol{S}$ is defined as a mapping from program variables to symbolic expressions which record the symbolic values of all variables that are handled symbolically. The *symbolic path constraints* $\boldsymbol{PCs = < pc_1, pc_2, \cdots, pc_n >}$ are defined as a list of formulas over symbolic input values which record the symbolic path constraints the current execution satisfies. Further, we need to define *Input map* $\boldsymbol{I}$, which maps input variables to its initial values before the execution of the program. One advantage of our approach is that we can exploit all execution information at runtime, since the MSVL Interpreter possesses all necessary information.

We now give the algorithm DSE-MC for dynamic symbolic executing MSVL programs. The basic idea of this algorithm is similar to the traditional one. Given an MSVL program P as input, we first initialize I, S and PCs to be $\emptyset$. Next, P is executed with input I by the extended MSVL Interpreter both symbolically and concretely. During the execution, the symbolic values of program variables and the path constraints along the path of the execution are collected and stored in S and PCs respectively. Once an MSVL program is executed, a symbolic path is generated and added to the SEtree of P subsequently. The symbolic path represents the models of P generated by the inputs which satisfy the path constraints of the current execution. Then, we select one constraint from PCs and negate the selected constraint, and solve the modified path constraints, to generate further inputs which possibly direct the program along alternative paths. The selection of the constraints to be negated depends on a search strategy. Finally, we execute P with the new inputs repeatedly until the termination conditions are met. Actually, if all the feasible paths are executed or the pre-defined bound is reached, the termination conditions are met. The pseudo-code for DSE-MC is shown in Algorithm 1.

Algorithm 1. Dynamic Symbolic Execution of MSVL Programs

```
Function DSE-MC($P$)
/* Input: $P$ is an MSVL program*/
/* Output: The Symbolic Execution Tree of $P$ */
begin function
  1:    $I = \emptyset$; $S = \emptyset$; $PCs = \emptyset$;
  2:    while termination conditions are not met
  3:        $SymbolicPath$=ExtMSVLInterpreter ($P$, $I$, $S$, $PCs$);
  4:        AddToSET ($SymbolicPath$);
  5:        $i$ = pick a constraint from $PCs$ ;
  6:        $I$ = Solve ($pc_1$,$pc_2$,$\cdots$, $\neg pc_i$ );
  7:    end while
end function
```

Extended MSVL Interpreter: The MSVL programs are executed only concretely by the original MSVL Interpreter. To execute MSVL programs both concretely and symbolically, we extend the MSVL Interpreter to implement the symbolic execution semantics. There are three main extensions of the MSVL Interpreter.

Algorithm 2. Dynamic Symbolic Execution of MSVL Programs using BDFS

```
Function DSE-MC-BDFS($P$, $max\_depth$)
/* Input: $P$ is an MSVL program, $max\_depth$ is the max execution times of $P$ */
/* Output: The Symbolic Execution Tree of $P$ */
begin function
  1:    $I = \emptyset$; $S = \emptyset$;$PCs = \emptyset$;
  2:    $SymbolicPath$=ExtMSVLInterpreter ($P$, $I$, $S$, $PCs$);
  3:    AddToSET ($SymbolicPath$);
  4:    BDFS (0, $max\_depth$, $PCs$, $P$, $I$, $S$);
end function
function BDFS($pos$, $depth$, $PCs$, $P$, $I$, $S$)
/* Input:$pos$ is the index of the path constraint picked to negate, $depth$ is the max execution times of $P$
        $PCs$ is the path constraints collected in previous execution
        $P$ is a MSVL program, $I$ is the input of $P$, $S$ is the symbolic memory map */
begin function
  1:    if $depth \geq 0$ do
  2:        for $i = pos$ to $PCs.size()$ do
  3:            $I$ = Solve($pc_1$,$pc_2$,$\cdots$, $\neg pc_i$);
  4:            $SymbolicPath$ =ExtMSVLInterpreter ($P$, $I$, $S$, $PCs$);
  5:            AddToSET ($SymbolicPath$);
  6:            $depth - -$;
  7:            BDFS ($i + 1$, $depth$, $PCs$, $P$, $I$, $S$);
  8:        end for
  9:    end if
end function
```

An MSVL program can be written as a logically equivalent conjunction of two programs *Present* and *Remains* (i.e., normal form) by the reduction process. *Present* part consists of immediate assignments to program variables, output of program variables, true, false or empty, and is executed at the current state. *Remains* part is what is executed in the subsequent state. In other words, any MSVL statements such as *Sequential statements*, *Projection statements*, *Parallel statements* and *while statements* can be reduced to their normal forms (NFs). In particular, the *while statements* can be transformed to *if statement* according to its definition $while\ b\ do\ p \overset{\text{def}}{=} (p \wedge q)^* \wedge \Box(empty \rightarrow \neg b)$. Therefore, to execute an MSVL program, we first transform it to its NF, then to interpret the *present* part, and further to execute its next part (*Remains*) recursively. In the extended interpreter, we focus on the dynamic symbolic execution as follows.

(1) For each input statement such as *input() and empty*, the symbolic memory map $\boldsymbol{S}$ introduces a mapping $x \longmapsto s_x$ from variable x to a fresh symbolic constant s_x which is regarded as the symbolic value of variable x. When the MSVL program is first executed, the input map $\boldsymbol{I}$ is $\emptyset$. Thus, we need to generate the inputs randomly. When the MSVL program is executed again, the inputs are generated by SMT solver Z_3.

(2) For each assignment statement such as $x <== e$ where e is an arithmetic expression that contains symbolic variables, the symbolic memory map $\boldsymbol{S}$ updates the mapping of x to $\boldsymbol{S}(e)$ which is obtained by evaluating e in the current symbolic memory map. For example, suppose that the symbolic memory map is $S = [x \longmapsto s_x, y \longmapsto s_y, z \longmapsto s_z]$, after the symbolic execution of $x <== 2*y+z$, the symbolic memory map becomes $S = [x \longmapsto 2*s_y+s_z, y \longmapsto$

$s_y, z \longmapsto s_z]$. For each assignment statement such as $x <== e$ where e is an arithmetic expression that contains no symbolic variables, the symbolic memory map is not updated, the statements are just executed concretely.

(3) For each conditional statement such as *if b then p else q*, if the concrete execution takes the *then branch*, the symbolic constraint $\boldsymbol{S}(b)$ is appended to $\boldsymbol{PCs}$. Otherwise, the symbolic constraint $\boldsymbol{S}(\neg b)$ is added.

Formally, we can construct a function called $ExtMSVLInterpreter$ (P, I, S, PCs) to fulfil the above functions. In fact, this function is the program that implements the new interpreter. It is omitted here.

Search Strategy and Constraint Solving: To execute all the paths in MSVL programs, we implement a commonly used bounded-depth-first(BDFS) strategy. In the BDFS, each run is executed based on path constraints collected in the previous run. The pseudo-code implementing BDFS strategy is shown in Algorithm 2.

The procedure Solve in Algorithm 2 transforms the new path constraints into the format that SMT solver Z_3 can deal with and then calls Z_3 to produce a solution that satisfies the new path constraints as the new inputs. If the new path constraints are unsatisfiable, another constraint is negated until satisfiable path constraints are found. We choose Z_3 as our constraint solver, because it provides more extensive supports to solve linear arithmetic and non-linear arithmetic constraints.

Symbolic Execution Tree: The SEtree characterizes the execution paths of a program and is generated at the end of DSE. In general, in an SEtree, a node represents the statements executed, labeled with the statement number, and a directed arc represents a transfer of control between statement executions, labeled with the changes of the *symbolic memory map* $\boldsymbol{S}$ and *symbolic path constraints* $\boldsymbol{PCs}$, if any, caused by the execution of the preceding statement. The traditional SEtree is modified to be the system model of MSVL programs. On one hand, each node is changed to specify a program in MSVL. On the other hand, each directed arc is changed to specify a state formula. The modified SEtree can be regarded as the LNFG of the MSVL program. For example, the corresponding traditional and modified SEtrees of the MSVL program in Fig.1(a) are shown in Fig.1(b) and Fig.2(d) respectively.

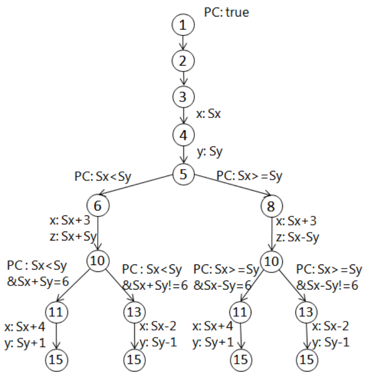
2.2 An Example

We use a simple example to illustrate how an MSVL program performs DSE to generate its SEtree. Consider the MSVL program P shown in Fig. 1(a). In P, there are three variables that can be handled symbolically and four paths introduced by the two *if statements* in lines 5-9 and lines 10-14. According to the values of the three variables and the relationships among them, P runs following different paths. To construct the SEtree of P, we need to execute it at least four times to explore all the existing four paths in P.

There are two points we should pay attention to when executing P symbolically. The first is that input variables x and y are given symbolic values s_x and

```
1.  frame(x,y,z) and (
2.      int x, y, z and skip;
3.      input(x) and empty;
4.      input(y) and empty;
5.      if(x<y) then {
6.          x:= x+3 and z := x+y
7.      }else{
8.          x:= x+3 and z := x-y
9.      };
10.     if(z=6) then {
11.         x:= x+1 and y := y+1
12.     }else{
13.         x:= x-1 and y := y-1
14.     }
15. )
```

(a) An MSVL Program P

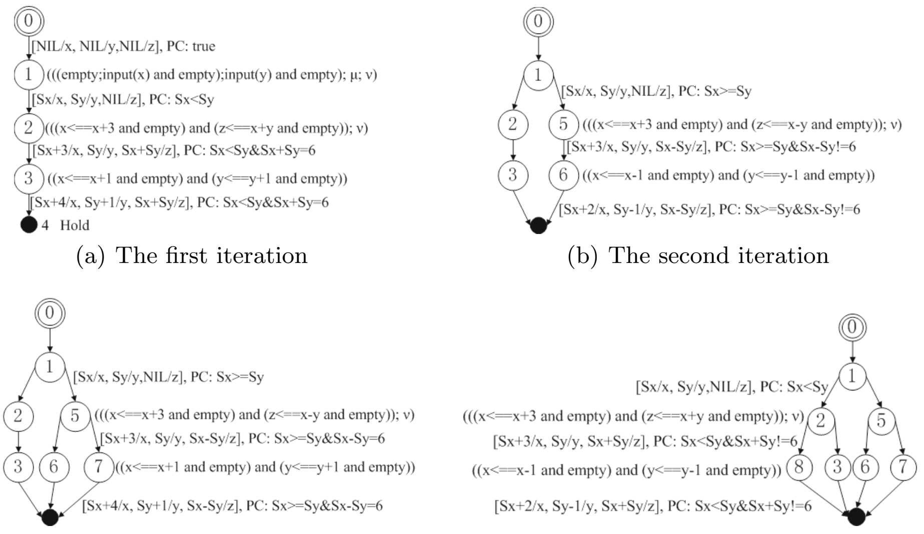


(b) Traditional SEtree of P

Fig. 1. An MSVL program and its traditional SEtree

s_y respectively when the input statements in line 3 and line 4 are executed. The second is that the ***PCs*** are initialized to ø before the execution, and updated after the execution of *if statements* in lines 5-9 and lines 10-14.

Now we describe the dynamic symbolic execution of P. In the first iteration of the DSE, it is assumed that the initial input is $\boldsymbol{I_1 = (x = 1, y = 5)}$ which is generated randomly. Executing P both concretely and symbolically by the extended MSVL Interpreter, the path this execution followed is recorded in *symbolic path constraints* ***PCs***, the details of the execution such as the changes of symbolic memory map ***S*** and ***PCs*** are shown in Fig. 2(a), where μ and ν in the figure represent statements *if(x < y) then* {*x:=x+3 and z:=x+y*}*else*{*x:=x+3 and z:=x-y*} and *if(z=6) then* {*x:= x+1 and y := y+1*}*else*{*x:= x-1 and y:=y-1*} respectively. $[s_x/x, s_y/y, NIL/z]$ in Fig. 2 means the values of variables x, y and z are s_x, s_y and NIL respectively. The symbolic path constraints collected in the first iteration are $\boldsymbol{PCs\_1 =< s_x < s_y, s_x + s_y = 6 >}$, which mean P takes the *then branch* of the first and the second *if statement*. With BDFS search strategy, we negate a constraint in the current execution and remove the subsequent constraints to obtain a modified symbolic path constraints $PCs\_1' =< s_x \geq s_y >$. Next, we solve PCs_1' to obtain input I_2 that drives the next execution along an alternative path. Suppose that the new input generated by Z_3 is $\boldsymbol{I_2 = (x = 3, y = 2)}$. Then, in the second iteration, P takes the *else branch* of the first and the second *if statement*. The path $< 0, 1, 5, 6, 4 >$ in Fig. 2(b) is the path this execution followed. For this execution, the path constraints $\boldsymbol{PCs\_2 =< s_x \geq s_y, s_x - s_y \neq 6 >}$ are gathered. We next solve the $PCs\_2' =< s_x \geq s_y, s_x - s_y = 6 >$, obtained by negating the last constraint and generate I_3 for the third execution. The third and fourth iterations that run with inputs $\boldsymbol{I_3 = (x = 8, y = 2)}$ and $\boldsymbol{I_4 = (x = 1, y = 2)}$ are similar to the second iteration. The paths $< 0, 1, 5, 7, 4 >$ in Fig. 2(c) and $< 0, 1, 2, 8, 4 >$ in Fig. 2(d) are the paths the third and fourth execution take respectively. After the


[NIL/x, NIL/y,NIL/z], PC: true
(((empty;input(x) and empty);input(y) and empty); μ; ν)
[Sx/x, Sy/y,NIL/z], PC: Sx<Sy
(((x<==x+3 and empty) and (z<==x+y and empty)); ν)
[Sx+3/x, Sy/y, Sx+Sy/z], PC: Sx<Sy&Sx+Sy=6
((x<==x+1 and empty) and (y<==y+1 and empty))
[Sx+4/x, Sy+1/y, Sx+Sy/z], PC: Sx<Sy&Sx+Sy=6
4 Hold
[Sx/x, Sy/y,NIL/z], PC: Sx>=Sy
(((x<==x+3 and empty) and (z<==x-y and empty)); ν)
[Sx+3/x, Sy/y, Sx-Sy/z], PC: Sx>=Sy&Sx-Sy!=6
((x<==x-1 and empty) and (y<==y-1 and empty))
[Sx+2/x, Sy-1/y, Sx-Sy/z], PC: Sx>=Sy&Sx-Sy!=6
[Sx+3/x, Sy/y, Sx-Sy/z], PC: Sx>=Sy&Sx-Sy=6
((x<==x+1 and empty) and (y<==y+1 and empty))
[Sx+4/x, Sy+1/y, Sx-Sy/z], PC: Sx>=Sy&Sx-Sy=6
[Sx/x, Sy/y,NIL/z], PC: Sx<Sy
(((x<==x+3 and empty) and (z<==x+y and empty)); ν)
[Sx+3/x, Sy/y, Sx+Sy/z], PC: Sx<Sy&Sx+Sy!=6
((x<==x-1 and empty) and (y<==y-1 and empty))
[Sx+2/x, Sy-1/y, Sx+Sy/z], PC: Sx<Sy&Sx+Sy!=6

(a) The first iteration (b) The second iteration

(c) The third iteration (d) The fourth iteration

Fig. 2. Dynamic Symbolic Execution of MSVL Program P

fourth execution of P, the dynamic symbolic execution algorithm is terminated because all the four paths have been explored. At last, the whole SEtree of P is constructed as shown in Fig. 2(d).

3 Model Checking MSVL Programs

The DSE based model checking approach is formalized in detail in this section. This approach can be divided into three major parts:the system model, the property and the model checking algorithm.

3.1 System Model

A path of a program execution is a model of the program, the set of all the paths of program executions is called all the models of the program, or *program models* for short. A path in an SEtree is an abstract path with constraints, which is the representative of all the paths satisfying the constraints. In other words, each path in the SEtree represents an equivalence class of paths (or inputs). Thus, the SEtree can be regarded as an abstract program model of an MSVL program.

With the method of program verification based on model checking, a system model means all the models of the program to be verified. Thus, the correctness of program verification can be guaranteed only if the SEtree completely covers the program models. However, we may not obtain complete abstract models in some cases. To combat the problem, we should increase the coverage degree of the program models as much as possible. Here we give a formal definition of the coverage degree, if $A \subsetneqq B \subsetneqq C$ where A, B and C are sets, we say, C includes B and A, B includes A. On the contrary, A and B cover part of C, and further B has a higher coverage degree over C than A.

3.2 Property

Propositional projection temporal logic (PPTL) [8,13] with the full regular expressive power is employed as the property specification language in our offline model checking approach. Thus, any MSVL programs with regular properties can be automatically verified with our offline model checking approach.

Let *Prop* be a countable set of atomic propositions and $B = \{true, false\}$ the boolean domain. Usually, we use small letters, possibly with subscripts, like *p,q,r* to denote atomic propositions and capital letters, possibly with subscripts, like *P, Q, R* to represent general PPTL formulas. Then the formula P of PPTL is defined by the following grammar:

$$P ::= p \mid \neg P \mid P_1 \wedge P_2 \mid \bigcirc P \mid (P_1, \ldots, P_m)\,\mathsf{prj}\,P \mid P^+$$

where $p \in Prop$, $\bigcirc$ (next), $+$ (chop-plus) and prj (projection) are temporal operators, and $\neg, \wedge$ are similar as that in the classical propositional logic. We define a *state* s over *Prop* to be a mapping from *Prop* to$B$, $s : Prop \rightarrow B$. We use $s[p]$ to denote the valuation of p at state s. Intervals and interpretations can be defined in the same way as in the first order case PTL [8]. The definitions for NFs and NFGs as well as LNFGs of PPTL formulas are the same as that presented in [21]. If the property to be verified is specified by a PPTL formula P, we can construct the LNFG of $\neg P$ for the subsequent verification according to its normal form [21].

3.3 Model Checking Algorithm

With our offline model checking algorithm for an MSVL program, the MSVL program M is modeled as its SEtree A_m, and property to be verified is specified by a PPTL formula P. To check if the MSVL program satisfying P is valid, we first obtain SEtree A_m by the dynamic symbolic execution of M. Then, we transform $\neg P$ to an LNFG A_φ, the transformation algorithm is given in [21]. Finally, we calculate the product of A_m and A_φ. If the product LNFG is not empty then a counter example is found, otherwise we cannot determine if the MSVL program satisfying the property is valid. In this case, the verification process could be restarted with new inputs. The algorithm is shown in Fig. 3.

Since model A_m and property P of a system are both described in the same logic framework PTL, it enables us to improve the efficiency of verification because of the avoidance of some transformations. However, there exists a problem that we should pay attention to, that is, it cannot be guaranteed that the SEtree we obtain covers all the models of an MSVL program. If we find a counter example during verification, the MSVL program satisfying the property is not valid. However, if there exist no counter examples, whether or not the

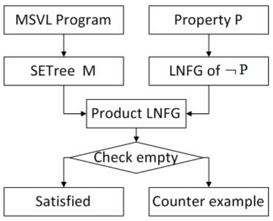


Fig. 3. Offline Model Checking MSVL

MSVL program satisfies the property cannot be determined. Thus, we should improve the coverage degree of the program models in the corresponding SEtree to improve the completeness of DSE-MC approach.

4 A Case Study

In this section, a typical example is presented to illustrate how the offline model checking approach can be utilized in the verification of real-world programs.

4.1 AC-controller Example

We use the AC-controller example presented in [7], which is used to compare the efficiency of traditional testing approach with the testing approach based on dynamic symbolic execution. Fig. 4(a) shows the AC-controller example simulating a controller for an air-conditioning system which is implemented by C. Initially, the room is hot and the door is closed, so the ac_controller is on. We can send a message to control the system. Different messages have different meanings and operations. From the code, we can also find when *is_room_hot&&is_door_closed&&!ac* is true, the system will be collapsed.

```
#include<stdio.h>

int is_room_hot=1,is_door_closed=1,ac=1;

void main(){
      int message;
      scanf("%d", &message);
      if (message == 1){
           is_room_hot = 0;
      }
      if (message == 2) {
          is_door_closed=0;
      }
      if (message == 3) {
          ac=0;
      }
      if (is_room_hot && is_door_closed && !ac)
           abort(); /* check correctness */
}
```

(a) A C program

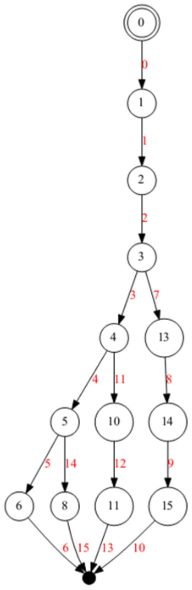
```
frame(is_room_hot,is_door_closed,ac) and (
    int is_room_hot<==1,is_door_closed<==1,ac<==1 and skip;
    function main ( ){
         frame(main_message) and (
             int main_message and skip;
             input(main_message) and skip;
             if(main_message=1)
                 then {is_room_hot:=0} else {skip };
             if(main_message=2)
                 then {is_door_closed:=0}else {skip };
             if(main_message=3)
                  then {ac:=0}else {skip }
         )
    };
    main()
)
```

(b) A MSVL program

Fig. 4. AC- controller example

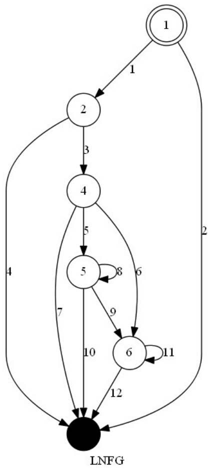
4.2 Verification

To verify AC-controller example, we first transform the original C program (remove the statement $if(is\_room\_hot \&\& is\_door\_closed \&\& !ac)abort();$) to an equivalent MSVL program as shown in Fig. 4(b) using toolkit MSV developed



Edge0:$[1/h, 1/d, 1/a, S_m/m]$, true
Edge1:$[1/h, 1/d, 1/a, S_m/m]$, true
Edge2:$[1/h, 1/d, 1/a, S_m/m]$, true
Edge3:$[1/h, 1/d, 1/a, S_m/m]$, $S_m \neq 1$
Edge4:$[1/h, 1/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m \neq 2$
Edge5:$[1/h, 1/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m \neq 2 \& S_m \neq 3$
Edge6:$[1/h, 1/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m \neq 2 \& S_m \neq 3$
Edge7:$[1/h, 1/d, 1/a, S_m/m]$, $S_m = 1$
Edge8:$[0/h, 1/d, 1/a, S_m/m]$, $S_m = 1 \& S_m \neq 2$
Edge9:$[0/h, 1/d, 1/a, S_m/m]$, $S_m = 1 \& S_m \neq 2 \& S_m \neq 3$
Edge10:$[0/h, 1/d, 1/a, S_m/m]$, $S_m = 1 \& S_m \neq 2 \& S_m \neq 3$
Edge11:$[1/h, 1/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m = 2$
Edge12:$[1/h, 0/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m = 2 \& S_m \neq 3$
Edge13:$[1/h, 0/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m = 2 \& S_m \neq 3$
Edge14:$[1/h, 1/d, 1/a, S_m/m]$, $S_m \neq 1 \& S_m \neq 2 \& S_m = 3$
Edge15:$[1/h, 1/d, 0/a, S_m/m]$, $S_m \neq 1 \& S_m \neq 2 \& S_m = 3$

Fig. 5. The symbolic execution tree of AC- controller example



Node ID	Corresponding PPTL formula	Label
Node1	!((len(2)) ; [](!(p) || !(q) || (r)))	
Node2	!(()EMPTY ; [](!(p) || !(q) || (r)))	
Node3	EMPTY	
Node4	<>((p) && (q) && !(r))	
Node5	(fin(l1)) ; (p) && (q) && !(r)	l1
Node6	TRUE	

Nodes of LNFG

Edge ID	state formula	->
Edge1	TRUE	Node1->Node2
Edge2	TRUE	Node1->Node3
Edge3	TRUE	Node2->Node4
Edge4	TRUE	Node2->Node3
Edge5	TRUE	Node4->Node5
Edge6	(q) && (p) && !(r)	Node4->Node6
Edge7	(q) && (p) && !(r)	Node4->Node3
Edge8	TRUE	Node5->Node5
Edge9	(q) && (p) && (l1) && !(r)	Node5->Node6
Edge10	(q) && (p) && (l1) && !(r)	Node5->Node3
Edge11	TRUE	Node6->Node6
Edge12	TRUE	Node6->Node3

Edges of LNFG

Fig. 6. The LNFG of the negation of Property

by us [19]. Then, we construct the SEtree of the MSVL program using Algorithm 2. The SEtree is shown in Fig. 5. In Fig. 5, letters h, d, a and m represent the variables is_room_hot, is_door_closed, ac and $message$ respectively, and s_m represents the symbolic value of $message$. Next, we specify the property to be verified by a PPTL formula. It is easy to find out that the property needs to be verified is "After a message is sent, AC-Controller should never be off when the room is hot and the door is closed". By employing propositions p, q and r to denote $is\_room\_hot = 1$, $is\_door\_closed = 1$ and $ac = 1$ respectively, this property can be specified by $P = len(2); \Box((p \wedge q) \rightarrow r)$ in PPTL. The LNFG of $\neg P$ is shown in Fig. 6. Finally, the product of the SEtree and the LNFG of $\neg P$ is calculated. We can find a counter example that violates P when the input value is 3. The path$< 0, 1, 2, 3, 4, 14, 15 >$ in Fig. 5 is the counter example.

5 Conclusion

We proposed a DSE based model checking approach for verifying MSVL programs. This approach can be used to verify software systems since C/Verilog programs can be transformed to MSVL programs by using toolkit MSV developed by us. Basically, the approach is based on abstract models because an SEtree can be viewed as an abstract model. The advantage is that the proposed method is a unified, automatical and quick approach. However, this method is an incomplete model checking approach like BMC in some sense because a counter example is the real one for the original program if the product of the SEtree and the LNFG of the negation of property is not empty. However, if there are no counter examples to be encountered we cannot determine if the original program satisfies the property. Therefore, in the future, we intend to further investigate how to improve the coverage degree of symbolic execution trees to the models of a program.

References

1. Ma, Q., Duan, Z., Zhang, N., Wang, X.: Verification of distributed systems with the axiomatic system of MSVL. Formal Aspects of Computing **27**(1), 103–131 (2015)
2. King, J.C.: Symbolic Execution and Program Testing. Journal of ACM, 385–394 (1976)
3. Ma, Y., Duan, Z., Wang, X., Yang, X.: An Interpreter for framed tempura and its application. In: Proceedings TASE 2007, pp. 251–260 (2007)
4. Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: Exe: automatically generating inputs of death. In: Proceedings of CCS 2006, pp. 322–335 (2006)
5. Beckert, B., Hahnle, R., Schmitt, P.H.: Verification of Object-Oriented Software. LNCS (LNAI), vol. 4334. Springer, Heidelberg (2007)
6. Jacobs, B., Piessens, F.: The Verifast Program Verifier (2008)
7. Godefroid, P., Klarlund, N., Sen, K.: DART: directed automated random testing. In: Proceedings of PLDI 2005, pp. 213–223 (2010)
8. Duan, Z.: Temporal Logic and Temporal Logic Programming. Science Press, Beijing (2006)

9. Sen, K., Marinov, D., Agha, G.: CUTE: a concolic unit testing engine for C. In: Proceedings of ESEC FSE 2005, pp. 263C–272 (2005)
10. Jaffar, J., Santosa, A.E., Voicu, R.: An interpolation method for CLP traversal. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 454–469. Springer, Heidelberg (2009)
11. McMillan, K.L.: Lazy annotation for program testing and verification. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 104–118. Springer, Heidelberg (2010)
12. Harris, W.R., Sankaranarayanan, S., Ivančić, F., Gupta, A.: Program analysis via satisfiability modulo path programs. In: Proceedings of POPL 2010, pp. 71–82 (2010)
13. Duan, Z.: An extended interval temporal logic and a framing technique for temporal logic programming, Ph.D. thesis, University of Newcastle upon Tyne (1996)
14. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: CounterrExample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
15. Ball, T., Majumdar, R., Millstein, T., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proceedings of PLDI 2001, pp. 203–213 (2001)
16. Zhang, Y., Chen, Z., Wang, J., Dongy, W., Liu, Z.: Regular property guided dynamic symbolic execution. In: Proceedings of ICSE 2015 (2015)
17. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems-specification. Springer (1992). ISBN 978-3-540-97664-6
18. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Bounded model checking. Advances in computers **58**, 117–148 (2003)
19. Yang, K., Duan, Z., Tian, C.: Modeling and Verification of REC Handover Protocol. Electronic Notes Theoretical Computer Science **309**, 51–62 (2014)
20. Duan, Z., Tian, C.: A unified model checking approach with projection temporal logic. In: Proceedings of ICFEM 2008, pp. 167–186 (2008)
21. Duan, Z., Tian, C.: A practical decision procedure for Propositional Projection Temporal Logic with infinite models. Theoretical Computer Science **554**, 169–190 (2014)

Geometric Cover

On the Complete Width and Edge Clique Cover Problems

Van Bang Le[1] and Sheng-Lung Peng[2](✉)

[1] Institut Für Informatik, Universität Rostock, Rostock, Germany
van-bang.le@uni-rostock.de

[2] Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien 974, Taiwan
slpeng@mail.ndhu.edu.tw

Abstract. A complete graph is the graph in which every two vertices are adjacent. For a graph $G = (V, E)$, the complete width of G is the minimum k such that there exist k independent sets $\mathtt{N}_i \subseteq V$, $1 \leq i \leq k$, such that the graph G' obtained from G by adding some new edges between certain vertices inside the sets $\mathtt{N}_i$, $1 \leq i \leq k$, is a complete graph. The complete width problem is to compute the complete width of a given graph. In this paper we study the complete width problem. We show that the complete width problem is NP-hard on $3K_2$-free bipartite graphs and polynomially solvable on $2K_2$-free bipartite graphs and on $(2K_2, C_4)$-free graphs. As a by-product, we obtain the following new results: the edge clique cover problem is NP-complete on $K_{2,2,2}$-free co-bipartite graphs and polynomially solvable on $K_{2,2}$-free co-bipartite graphs and on $(2K_2, C_4)$-free graphs. We also determine all graphs of small complete width $k \leq 3$.

Keywords: Probe graphs · Split graphs · Bipartite graphs · Complete width · Edge clique cover

1 Introduction

Let $G = (V, E)$ be a simple and undirected graph. A subset $U \subseteq V$ is an *independent set*, respectively, a *clique* if no two, respectively, every two vertices of U are adjacent. The complete graph with n vertices is denoted by K_n. The path and cycle with n vertices of length $n-1$, respectively, of length n, is denoted by P_n, respectively, C_n. For a vertex $v \in V$ we write $N(v)$ for the set of its neighbors in G. A *universal* vertex v is one such that $N(v) \cup \{v\} = V$. For a subset $U \subseteq V$ we write $G[U]$ for the subgraph of G induced by U and $G - U$ for the graph $G[V - U]$; for a vertex v we write $G - v$ rather than $G[V \setminus \{v\}]$.

Given a graph class $\mathcal{C}$, a graph $G = (V, E)$ is called a *probe $\mathcal{C}$ graph* if there exists an independent set $\mathtt{N} \subseteq V$ (of *nonprobes*) and a set of new edges $E' \subseteq \binom{\mathtt{N}}{2}$ between certain nonprobe vertices such that the graph $G' = (V, E \cup E')$ is in the class $\mathcal{C}$, where $\binom{\mathtt{N}}{2}$ stands for the set of all 2-element subsets of $\mathtt{N}$. A graph $G = (V, E)$ with a *given* independent set $\mathtt{N} \subseteq V$ is said to be a *partitioned probe*

$\mathcal{C}$ *graph* if there exists a set $E' \subseteq \binom{\mathtt{N}}{2}$ such that the graph $G' = (V, E \cup E')$ is in the class $\mathcal{C}$. In both cases, G' is called a $\mathcal{C}$ *embedding* of G. Thus, a graph is a (partitioned) probe $\mathcal{C}$ graph if and only if it admits a $\mathcal{C}$ embedding.

Recently, the concept of probe graphs has been generalized as a width parameter of graph class in [4]. Let $\mathcal{C}$ be a class of graphs. The *$\mathcal{C}$-width* of a graph G is the minimum number k of independent sets $\mathtt{N}_1, \ldots, \mathtt{N}_k$ in G such that there exists an embedding $G' \in \mathcal{C}$ of G such that for every edge xy in G' which is not an edge of G there exists an i with $x, y \in \mathtt{N}_i$. A collection of such k independent sets $\mathtt{N}_i, i = 1, \ldots, k$ is called a $\mathcal{C}$ *witness* (of G'). In the case $k = 1$, G is a *probe $\mathcal{C}$-graph.* The *$\mathcal{C}$-width* problem asks for a given graph G and an integer k if the $\mathcal{C}$-width of G is at most k. Graphs of $\mathcal{C}$-width k are also called k-probe $\mathcal{C}$-graph. Note that graphs in $\mathcal{C}$ are, by convenience, 1-probe $\mathcal{C}$-graphs.

In [4], the complete width and block-graph width have been investigated. The authors proved that, for fixed k, graphs of complete width k can be characterized by finitely many forbidden induced graphs, their proof is however not constructive. They also showed, implicitly, that complete width k graphs and block-graph width k graphs can be recognized in cubic time. The case $k = 1$, *e.g.*, probe complete graphs and probe block graphs, has been discussed in depth in [17]. The case $k = 2$ is discussed in [18].

Graphs do not contain an induced subgraph isomorphic to a graph H are called *H-free*. More generally, a graph is *$(H_1, \ldots, H_t)$-free* if it does not contain an induced subgraph isomorphic to one of the graphs $H_1, \ldots, H_t$. For two graphs G and H, we write $G + H$ for the disjoint union of G and H, and for an integer $t \geq 2$, tG stands for the disjoint union of t copies of G. The complete k-partite with n_i vertices in color class i is denoted by $K_{n_1,\ldots,n_k}$. For graph classes not defined here see, for example, [2,3,10].

In this paper we study the COMPLETE WIDTH problem (given G and k, is the complete width of G at most k?). We show that

– COMPLETE WIDTH is NP-complete, even on $3K_2$-free bipartite graphs, and
– computing the complete width of a $2K_2$-free bipartite graph (chain graph), and more generally, of a $(2K_2, K_3)$-free graph can be done in polynomial time,
– computing the complete width of a $2K_2$-free chordal graph (split graph), and more generally, of a $(2K_2, C_4)$-free graph can be done in polynomial time.

Moreover, we give structural characterizations for graphs of complete width at most 3.

In the next section we point out a relation between complete width and the most popular notion of edge clique cover of graphs. Then we prove our results in the last three sections. As we will see, it follows from our results on complete width that edge clique cover is NP-complete on $K_{2,2,2}$-free co-bipartite graphs and is polynomially solvable on $K_{2,2}$-free co-bipartite graphs.

2 Complete Width and Edge Clique Cover

An *edge clique cover* of a graph G is a family of cliques (complete subgraphs) such that each edge of G is in at least one member of the family. The minimal cardinality of an edge clique cover is the *edge clique cover number*, denoted by $\theta_e(G)$.

The EDGE CLIQUE COVER problem, the problem of deciding if $\theta_e(G) \leq k$, for a given graph G and an integer k, is NP-complete [13,16,22], even when restricted to graphs with maximum degree at most six [14], or planar graphs [6]. EDGE CLIQUE COVER is polynomially solvable for graphs with maximum degree at most five [14], for line graphs [22,23], for chordal graphs [7,24], and for circular-arc graphs [15].

In [16] it is shown that approximating the clique covering number within a constant factor smaller than two is NP-hard. In [11], it is shown that EDGE CLIQUE COVER is fixed-parameter tractable with respect to parameter k; see also [8] for more recent discussions on the parameterized complexity aspects.

We write $cow(G)$ to denote the complete width of the graph G. As usual, $\overline{G}$ denotes the complement of G. In [4], the authors showed that COMPLETE WIDTH is NP-complete on general graphs, by observing that

Proposition 1 ([4]). *For any graph G, $cow(G) = \theta_e(\overline{G})$*

Proposition 1 and the known results about EDGE CLIQUE COVER imply:

Theorem 1. (1) *Computing the complete width is NP-hard, and remains NP-hard when restricted to graphs of minimum degree at least $n-7$, and to co-planar graphs.*

(2) *Computing the complete width of graphs of minimum degree at least $n-6$ and of co-chordal graphs can be done in polynomial time.*

In [5], it is conjectured that EDGE CLIQUE COVER, and thus COMPLETE WIDTH, is NP-complete for P_4-free graphs (also called cographs).

3 Computing Complete Width is Hard for $3K_2$-free Bipartite Graphs

A bipartite graph $G = (V, E)$ is a graph whose vertex set V can be partitioned into two sets X and Y such that for any edge $xy \in E$, $x \in X$ and $y \in Y$. Bipartite graphs without induced cycles of length at least six are called *chordal bipartite*. A *biclique cover* of a graph G is a family of complete bipartite subgraphs of G whose edges cover the edges of G. The *biclique cover number*, also called the *bipartite dimension*, of G is the minimum number of bicliques needed to cover all edges of G.

Given a graph G and a positive integer k, the BICLIQUE COVER problem of G asks whether the edges of G can be biclique covered by at most k bicliques. The following theorem is well known.

Theorem 2 ([21,22]). BICLIQUE COVER *is NP-complete on bipartite graphs, and remains NP-complete on chordal bipartite graphs.*

For convenience, a bipartite graph $G = (V, E)$ with a bipartition $V = X \cup Y$ into independent sets X and Y is denoted as $G = (X + Y, E)$. Let $BC(G) = (X + Y, F)$, where $F = \{xy \mid x \in X, y \in Y, \text{ and } xy \notin E\}$. We call $BC(G)$ the *bipartite complement* of $G = (X + Y, E)$. Note that $BC(C_6) = 3K_2$ and $BC(C_8) = C_8$. Hence if G is chordal bipartite, then $BC(G)$ is $(3K_2, C_8)$-free bipartite.

In [4], the authors showed that the complete width problem is NP-complete on general graphs. We now establish our main theorem for sharping that result of [4].

Theorem 3. COMPLETE WIDTH *is NP-complete on bipartite graphs, and remains NP-complete on $(3K_2, C_8)$-free bipartite graphs.*

Proof. We prove this theorem by reducing BICLIQUE COVER to COMPLETE WIDTH.

Let (G, k) be an input instance of the biclique cover problem, where $G = (X + Y, E)$ is a bipartite graph. We construct an input instance (G', k') of the complete width problem as follows.

- G' is the bipartite graph obtained from the bipartite complement $BC(G) = (X + Y, F)$ of G by adding two new vertices x and y and adding all edges between x and vertices in $Y \cup \{y\}$ and between y and vertices in $X \cup \{x\}$. More formally, $G' = (X' + Y', F')$ with $X' = X \cup \{x\}, Y' = Y \cup \{y\}$, and $F' = F \cup \{xu \mid u \in Y \cup \{y\}\} \cup \{yv \mid v \in X \cup \{x\}\}$.
- Set $k' := k + 2$.

We claim that the biclique cover number of G is at most k if and only if the complete width of G' is at most $k' = k + 2$.

First, let $\{B_i \mid 1 \leq i \leq k\}$ be a biclique cover of G, where $B_i = (X_i + Y_i, E_i)$ with $X_i \subseteq X, Y_i \subseteq Y$. Then, as each B_i is a biclique in G, each $N_i = X_i \cup Y_i$ is an independent set in G'. Set $N_{k+1} := X'$ and $N_{k+2} := Y'$. Then it is easy to check that the $k' = k + 2$ independent sets N_i, $1 \leq i \leq k + 2$, form a complete witness of G'. That is, $cow(G') \leq k'$.

Conversely, let $\{N_i \mid 1 \leq i \leq k + 2\}$ be a complete witness of G'. Then we may assume that

$$x, y \notin N_i, 1 \leq i \leq k.$$

(To see this, consider a vertex $u \in X$. As $\{N_i \mid 1 \leq i \leq k + 2\}$ is a complete witness of G', u and x must belong to N_t for some $t \in \{1, \ldots, k+2\}$. Therefore, $N_t \subseteq X \cup \{x\} = X'$ because x is adjacent to all vertices in Y'. Clearly, we can replace N_t by X' and, if $x \in N_i$ for some $i \neq t$, replace N_i by $N_i - \{x\}$ to obtain a new witness such that $N_t = X'$ and x is contained only in N_t. Similarly, there is some s such that $N_s = Y'$ and y is contained only in N_s. By re-numbering if necessary, we may assume that $t = k + 1$ and $s = k + 2$.)

Thus, by construction of G', $N_1, \dots, N_k$ are independent sets in $BC(G)$ and form a complete witness of $BC(G)$. Therefore, $B_i = G[N_i]$, $1 \le i \le k$, are bicliques in G forming a biclique cover of G. That is, $\theta_{\mathrm{e}}(G) \le k$.

Note that if G is chordal bipartite, then $BC(G)$, hence the bipartite graph G' cannot contain $3K_2$ and C_8 as induced subgraphs. □

Theorem 3 and Proposition 1 imply the following corollary.

Corollary 1. EDGE CLIQUE COVER *is NP-complete on $K_{2,2,2}$-free co-bipartite graphs.*

4 Polynomially Solvable Cases

In this section we establish some cases in which COMPLETE WIDTH can be solved in polynomial time. Actually, in each of these cases we will show that the complete width of the graphs under consideration can be computed in polynomial time.

4.1 $2K_2$-free Bipartite Graphs

Bipartite graphs without induced $2K_2$ are known in literature under the name *chain graphs* ([25]) or *difference graphs* ([12]). They can be characterized as follows.

Proposition 2 (see [20]). *A bipartite graph $G = (X + Y, E)$ is a chain graph if and only if for all vertices $u, v \in X$, $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$.*

Theorem 4. *The complete width of chain graphs can be computed in polynomial time.*

Proof. (Sketch) Observe first that if u, v are two vertices in a graph G such that $N(u) = N(v)$ (in particular, u and v are non-adjacent), then $cow(G) = cow(G-u)$ (if v is not universal in $G-u$) or $cow(G) = cow(G-u)+1$ (otherwise).

Let $G = (X+Y, E)$ be a $2K_2$-free bipartite graph with at least three vertices. As observed above, we may assume that for any pair of vertices u, v of G, $N(u) \neq N(v)$. Thus, $|X| \geq 2, |Y| \geq 2$, and G has at most one non-trivial connected component and at most one trivial component which is then the unique isolated vertex of G. Let us also assume that the isolated vertex (if any) of G belongs to X. By Proposition 2, the vertices of X can be numbered $v_1, v_2, \dots, v_{|X|}$ such that $N(v_1) \subset N(v_2) \subset \dots \subset N(v_{|X|}) = Y$. Thus, G is disconnected if and only if v_1 is the isolated vertex of G if and only if $N(v_1) = \emptyset$. Clearly, such a numbering can be computed in polynomial time.

Write $\mathtt{N}_i = \{v_1, \dots, v_i\} \cup (Y \setminus N(v_i))$, $1 \le i \le |X|$. Since $N(v_j) \subset N(v_i)$ for $j < i$, $\mathtt{N}_i$ is an independent set, and since $N(v_{|X|}) = Y$, $\mathtt{N}_{|X|} = X$. In case $N(v_1) \neq \emptyset$, let $\mathtt{N}_{|X|+1} = Y$. (Note that in case $N(v_1) = \emptyset$, *i.e.*, v_1 is the isolated vertex of G, $\mathtt{N}_1 = Y \cup \{v_1\}$.)

We claim that

$$cow(G) = \begin{cases} |X|, & \text{if } N(v_1) = \emptyset \\ |X| + 1, & \text{otherwise} \end{cases}$$

Moreover, $\mathtt{N}_1, \ldots, \mathtt{N}_{|X|}$ and $\mathtt{N}_{|X|+1}$ (if $N(v_1) \neq \emptyset$) together form a complete witness of G.

Proof of the Claim: First, to see that the collection of the independent sets $\mathtt{N}_1$, $\ldots$, $\mathtt{N}_{|X|}$ and $\mathtt{N}_{|X|+1}$ (if $N(v_1) \neq \emptyset$) is a complete witness of G, let u, v be two non-adjacent vertices of G. If u and v in X, say $u = v_i$ or $v = v_j$ for some $1 \leq i < j \leq |X|$, then $u, v \in \mathtt{N}_j$. If $u \in X$ and $v \in Y$, say $u = v_i$ for some $1 \leq i \leq |X|$, then $u, v \in \mathtt{N}_i$. So let $u, v \in Y$. In this case, let $i \leq j$ be the smallest integers such that $u \in N(v_i), v \in N(v_j)$. If $i > 1$ then $u, v \notin N(v_1)$, hence $u, v \in \mathtt{N}_1$. Thus, let $u \in N(v_1)$. Then, in particular $N(v_1) \neq \emptyset$ and hence $u, v \in \mathtt{N}_{|X|+1} = Y$.

In particular, $cow(G)$ is at most the right hand side stated in the claim.

Next, observe that the claim is clearly true in case $|X| = 2$. So, let $|X| \geq 3$. Note that in $G - v_1$, $N(v_2)$ is not empty, hence by induction, $cow(G - v_1) = |X \setminus \{v_1\}| + 1 = |X|$ and $\mathtt{N}'_1 = \mathtt{N}_2 \setminus \{v_1\}, \ldots, \mathtt{N}'_{|X|-1} = \mathtt{N}_{|X|} \setminus \{v_1\}$ and $\mathtt{N}'_{|X|} = \mathtt{N}_{|X|+1} = Y$ form a complete witness of $G - v_1$. Now, if $N(v_1) = \emptyset$ then $cow(G) \geq cow(G - v_1) = |X|$, hence $cow(G) = |X|$. So, let $N(v_1) \neq \emptyset$. In this case, for any $u \in N(v_2) \setminus N(v_1)$ and any maximal independent set I of G containing v_1 and u, $\mathtt{N}'_i \not\subseteq I$. Thus, $cow(G) \geq cow(G - v_1) + 1 = |X| + 1$, hence $cow(G) = |X| + 1$. □

Theorem 4 and Proposition 1 imply the following corollary.

Corollary 2. *The edge clique cover number of C_4-free co-bipartite graphs can be computed in polynomial time.*

4.2 $(2K_2, K_3)$-free Graphs

We extend Theorem 4 on K_2-free bipartite graphs by showing that COMPLETE WIDTH is polynomially solvable for large class of $2K_2$-free triangle-free graphs.

Theorem 5. *The complete width of $(2K_2, K_3)$-free graphs can be computed in polynomial time.*

Proof. (Sketch) Let G be a $(2K_2, K_3)$-free graph. If G has no induced C_5, then G is $2K_2$-free bipartite, hence we are done by Theorem 4.

So let G contain an induced C_5, say $C = v_1v_2v_3v_4v_5v_1$. As in the proof of Theorem 4, we may assume that $N(u) \neq N(v)$ for any non-adjacent vertices u and v of G. Then it can be shown that C is the non-trivial connected component of G. Thus, $cow(G) = 5$. □

4.3 Split Graphs

A *split graph* is one whose vertex set can be partitioned into a clique Q and an independent set S. For convenience, a split graph is denoted as $G = (Q + S, E)$. It is well known that split graphs can be characterized as follows.

Proposition 3 ([9]). *The following statements are equivalent for any graph G.*

(i) *G a split graph;*
(ii) *G is a $(2K_2, C_4, C_5)$-free graph;*
(iii) *G is a $2K_2$-free chordal graph;*
(iv) *G and $\overline{G}$ are chordal.*

In particular, split graphs are complements of chordal graphs. Hence, by Theorem 1 (2), computing the complete width of split graphs can be done in polynomial time. Below, however, we give here a simple and direct way for doing this. Moreover, our solution will be useful for computing the complete width of pseudo split graphs. The class of pseudo split graphs are not necessarily co-chordal and properly contains all split graphs.

It is not hard to see that a universal vertex is impossible to be a non-probe vertex. Thus in the following, we consider the split graphs $G = (Q + S, E)$ with no universal vertex.

Theorem 6. *For a split graph $G = (Q + S, E)$ with no universal vertex, the complete width of G is either $|Q|$ or $|Q| + 1$.*

Proof. Assume that the complete width of G is k. That is, there is an embedding G' of G such that for every edge xy in G' but not in G there are independent sets $\mathtt{N}_1, \ldots, \mathtt{N}_k$ in G such that $\{x, y\} \subseteq \mathtt{N}_i$ for some i. By the definition, $G[Q]$ is a clique. Thus it is impossible that there are two vertices of Q in the same $\mathtt{N}_i$ for $1 \leq i \leq k$. That is, each $\mathtt{N}_i$ contains at most one vertex in Q. Therefore, the complete width of G is at least $|Q|$.

On the other hand, for each vertex $v \in Q$, let $\mathtt{N}_v = V \setminus N(v)$. Then, each $\mathtt{N}_v$, $v \in Q$, is an independent set. Further, for each $\mathtt{N}_v$, we can fill edges vu, $u \in \mathtt{N}_v - v$. Finally, for the final set S, we make $G[S]$ a clique by filling edge xy for any two vertices $x, y \in S$. The resulting graph is a complete graph. That is, the complete width of G is at most $|Q| + 1$. This completes the proof. □

By Theorem 6, only two cases for determining the complete width of a split graph. For the split graph $G = (Q + S, E)$, let $\mathtt{N}_v = V \setminus N(v)$ for $v \in Q$. We have the following lemma.

Lemma 1. *For a split graph $G = (Q + S, E)$ with no universal vertex, if for any two vertices $x, y \in S$, there is an $\mathtt{N}_v$, $v \in Q$, such that $x, y \in \mathtt{N}_v$, then the complete width of G is $|Q|$; otherwise it is $|Q| + 1$.*

Proof. Assume that for any two vertices $x, y \in S$, there is an $\mathtt{N}_v$, $v \in Q$ such that $x, y \in \mathtt{N}_v$. We show that the complete width of G is $|Q|$. Without loss of generality, we assume all the $\mathtt{N}_v$'s are ordered as the sequence of $\mathtt{N}_1, \mathtt{N}_2, \ldots, \mathtt{N}_{|Q|}$. For completing G into K_n, for each $\mathtt{N}_v$, we fill the edges vu, $u \in (\mathtt{N}_v \cap S)$. Furthermore, assume that $\mathtt{N}_i$ is the last set that contains x and y for any two vertices $x, y \in S$. That is, $\{x, y\} \subseteq \mathtt{N}_i$ but $\{x, y\} \not\subseteq \mathtt{N}_j$ for each $j > i$. Then the edge xy is filled in $\mathtt{N}_i$. By assumption, every edge in $\overline{G}[S]$ can be filled in some $\mathtt{N}_i$. Thus the complete width of G is $|Q|$.

On the other hand, if there is no $\mathtt{N}_i$ contains x and y for some $x, y \in S$, then there is no way to fill x, y in $\mathtt{N}_1, \mathtt{N}_2, \ldots, \mathtt{N}_{|Q|}$. Therefore the complete width of G is $|Q| + 1$. □

By Lemma 1, for any two vertices $x, y \in S$, we can check whether there is a vertex $v \in Q$ such that both xv and yv are not in E. By using adjacency matrix of G, all the work can be done in $O(n^3)$ time. Thus, we have the following theorem.

Theorem 7. *The complete width of split graphs can be computed in polynomial time.*

4.4 Pseudo-split Graphs

Graphs without induced $2K_2$ and no induced C_4 are called *pseudo-split graphs*. Thus, by Proposition 3, the class of pseudo-split graphs properly contains the class of split graphs. Note that a pseudo-split graph may contain an induced C_5, hence it need not be co-chordal. Pseudo-split graphs can be characterized as follows.

Theorem 8 ([1,19]). *A graph is pseudo-split if and only if its vertex set can be partitioned into three sets Q, S, C such that Q is a clique, S is an independent set, C induces a C_5 or is empty, there are all possible edges between Q and C, and there are no edges between S and C.*

Note that it can be recognized in linear time if a graph is a pseudo split graph, and if so, a partition stated in Theorem 8 can be found in linear time [19].

Theorem 9. *The complete width of pseudo-split graphs can be computed in polynomial time.*

Proof. Let $G = (V, E)$ be a pseudo-split graph without universal vertices. Let $V = Q + S + C$ be a partition as in Theorem 8. We may assume that $C \neq \emptyset$ otherwise we are done by Theorem 7.

So let C be the induced $C_5 = v_1v_2v_3v_4v_5v_1$. Then, clearly, the $|Q| + 5$ independent sets $V - N(v)$, $v \in Q$, and $S \cup \{v_i, v_{i+2}\}$ (indices are taken modulo 5), $1 \leq i \leq 5$, can be used for completing G. Thus, by Theorem 6, and by noting that $cow(C_5) = 5$, we have $cow(G) = |Q| + 5$. □

Theorem 4 and Proposition 1 imply the following corollary (note that the complement of a pseudo-split is also a pseudo-split graph).

Corollary 3. *The edge clique cover number of pseudo-split graphs can be computed in polynomial time.*

5 Graphs of Small Complete Width

We describe in this section graphs of small complete width $k \leq 3$. These are particularly $2K_2$-free and our descriptions are good in the sense that they imply polynomial time recognition for these graph classes.

Complete Width-1 and Complete Width-2 Graphs

A *complete split graph* is a split graph $G = (Q + S, E)$ such that every vertex in the clique Q is adjacent to every vertex in the independent set S. Such a partition is also called a *complete split partition* of a split graph. Note that if the complete split graph $G = (Q + S, E)$ is not a clique, then G has exactly one complete split partition $V = Q \cup S$. Furthermore, each vertex in Q is a universal vertex.

Graphs of complete width one can be characterized as follows.

Theorem 10 ([17]). *The following statements are equivalent.*

(i) *G is a probe complete graph;*
(ii) *G is a $\{K_2 + K_1, C_4\}$-free graph;*
(iii) *G is a complete split graph.*

The *join* $G \star H$ is obtained from $G + H$ by adding all possible edges xy between any vertex x in G and any vertex y in H. Graphs of complete width at most two can be characterized as follows.

Theorem 11 ([18]). *A graph G is a 2-probe complete graph if and only if G is $\{2K_2, P_4, K_3 + K_1, (K_2 + K_1) \star 2K_1, C_4 \star 2K_1\}$-free.*

Complete Width-3 Graphs

Substituting a vertex v in a graph G by a graph H results in the graph obtained from $(G - v) \cup H$ by adding all edges between vertices in $N_G(v)$ and vertices in H. The *Net* consists of six vertices a, b, c, a', b' and c' and six edges $aa', bb', cc', a'b', b'c'$ and $a'c'$.

Graphs of complete width at most 3 can be characterized as follows.

Theorem 12. *Let Q be the set (possibly empty) of universal vertices of the graph G. G is a 3-probe complete graph if and only if $G - Q$ has at most one non-trivial connected component which is obtained from the Net by substituting the vertices by (possibly empty) independent sets.*

Proof. (Sketch) First, assume that G is a 3-probe complete graph, and let $\mathtt{N}_1, \mathtt{N}_2, \mathtt{N}_3$ be a complete witness of G. Then G is $2K_2$-free and $Q = V(G) \setminus \mathtt{N}_1 \cap \mathtt{N}_2 \cap \mathtt{N}_3$ is the set of all universal vertices of G. Moreover, as G is $2K_2$-free, $G - Q$ has at most one non-trivial connected component and $I = \mathtt{N}_1 \cap N_2 \cap \mathtt{N}_3$ is the set of all isolated vertices of $G - Q$.

The non-trivial connected component N of $G - Q$ is partitioned into the following six independent sets. $I_{12} = (\mathtt{N}_1 \cap \mathtt{N}_2) \setminus \mathtt{N}_3$, $I_{13} = (\mathtt{N}_1 \cap \mathtt{N}_3) \setminus \mathtt{N}_2$, $I_{23} = (\mathtt{N}_2 \cap \mathtt{N}_3) \setminus \mathtt{N}_1$, $I_1 = \mathtt{N}_1 \setminus (\mathtt{N}_2 \cup \mathtt{N}_3)$, $I_2 = \mathtt{N}_2 \setminus (\mathtt{N}_1 \cup \mathtt{N}_3)$, and $I_3 = \mathtt{N}_3 \setminus (\mathtt{N}_1 \cup \mathtt{N}_2)$. Then there are all possible edges between I_1 and I_2, I_3, I_{23}, between I_2 and I_1, I_3, I_{13}, between I_3 and I_1, I_2, I_{12}, and there are no other edges between these independent sets. Thus, N is obtained from the Net by substituting the vertices by these six independent sets.

Next, assume that $G - Q$ has at most one non-trivial connected component which is obtained from the Net by substituting the vertices by (possibly empty) independent sets. Let I be the set of trivial connected components and let N be the non-trivial connected component of $G - Q$. Let N be obtained from the Net by substituting its vertices a, b, c, a', b', c' by independent set $I_a, I_b, I_c, I_{a'}, I_{b'}, I_{c'}$, respectively. Then $\mathtt{N}_1 = I \cup I_a \cup I_{b'} \cup I_{c'}$, $\mathtt{N}_2 = I \cup I_b \cup I_{a'} \cup I_{c'}$, and $\mathtt{N}_3 = I \cup I_c \cup I_{a'} \cup I_{b'}$ from a complete witness of G. □

We note that, using modular decomposition, one can recognize graphs obtained from the Net by substituting vertices by independent sets in linear time. Hence Theorem 12 gives a linear time recognition for 3-probe complete graphs. We also remark that there is a characterization for 3-probe complete graphs by 14 forbidden induced subgraphs.

6 Conclusion

In this paper we have shown that COMPLETE WIDTH is NP-complete on $3K_2$-free bipartite graphs (equivalently, EDGE CLIQUE COVER is NP-complete on $K_{2,2,2}$-free co-bipartite graphs). So, an obvious open question is: What is the computational complexity of COMPLETE WIDTH on $2K_2$-free graphs? Equivalently, what is the computational complexity of EDGE CLIQUE COVER on C_4-free graphs? We have given partial results in this direction by showing that COMPLETE WIDTH is polynomially solvable on $(2K_2, K_3)$-free graphs and on $(2K_2, C_4)$-free graphs. (Equivalently, EDGE CLIQUE COVER is polynomially solvable on $(C_4, 3K_1)$-free graphs and on $(C_4, 2K_2)$-free graphs.)

Another interesting question is the following. The time complexities of many problems coincide on split graphs and bipartite graphs, *e.g.*, the dominating set problem. However, for the complete width problem, they are different, one is in P and the other is in NP-complete. Trees are a special class of bipartite graphs. Many problems become easy on trees. However, we do not know the hardness of the complete width problem on trees.

References

1. Blázsik, Z., Hujter, M., Pluhár, A., Tuza, Z.: Graphs with no induced C_4 and $2K_2$. Discrete Mathematics **115**, 51–55 (1993)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia (1999)

3. Chandler, D.B., Chang, M.-S., Kloks, T., Peng, S.-L.: Probe Graphs, (2009). http://www.cs.ccu.edu.tw/~hunglc/ProbeGraphs.pdf
4. Chang, M.-S., Hung, L.-J., Kloks, T., Peng, S.-L.: Block-graph width. Theoretical Computer Science **412**, 2496–2502 (2011)
5. Chang, M.-S., Kloks, T., Liu, C.-H.: Edge-clique graphs of cocktail parties have unbounded rankwidth, (2012). arXiv:1205.2483 [cs.DM]
6. Chang, M.-S., Müller, H.: On the tree-degree of graphs. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 44–54. Springer, Heidelberg (2001)
7. Ma, S., Wallis, W.D., Wu, J.: Clique covering of chordal graphs. Utilitas Mathematica **36**, 151–152 (1989)
8. Cygan, M., Pilipczuky, M., Pilipczuk, M.: Known algorithms for EDGE CLIQUE COVER are probably optimal. In: Proc. SODA, 1044–1053 (2013)
9. Foldes, S., Hammer, P.L.: Split graphs. Congressus Numerantium, No. XIX, 311–315 (1977)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Math., vol. 57, 2nd edn. Elsevier, Amsterdam (2004)
11. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Data reduction and exact algorithms for clique cover. ACM Journal of Experimental Algorithmics **13** (2008). Article 2.2
12. Hammer, P.L., Peled, U.N., Sun, X.: Difference graphs. Discrete Applied Mathematics **28**, 35–44 (1990)
13. Holyer, I.: The NP-completeness of some edge-partition problems. SIAM Journal on Computing **4**, 713–717 (1981)
14. Hoover, D.N.: Complexity of graph covering problems for graphs of low degree. Journal of Combinatorial Mathematics and Combinatorial Computing **11**, 187–208 (1992)
15. Hsu, W.-L., Tsai, K.-H.: Linear time algorithms on circular-arc graphs. Inf. Process. Lett. **40**, 123–129 (1991)
16. Kou, L.T., Stockmeyer, L.J., Wong, C.K.: Covering edges by cliques with regard to keyword conflicts and intersection graphs. Comm. ACM **21**, 135–139 (1978)
17. Le, V.B., Peng, S.-L.: Characterizing and recognizing probe block graphs. Theoretical Computer Science **568**, 97–102 (2015)
18. Le, V.B., Peng, S.-L.: Good characterizations and linear time recognition for 2-probe block graphs. In: Proceedings of the International Computer Symposium, Taichung, Taiwan, December 12–14, 2014, pp. 22–31. IOS Press (2015). doi:10.3233/978-1-61499-484-8-22
19. Maffray, F., Preissmann, M.: Linear recognition of pseudo-split graphs. Discrete Applied Mathematics **52**, 307–312 (1994)
20. Mahadev, N.V.R., Peled, U.N.: Threshold Graphs and Related Topics. Annals of discrete mathematics, vol. 56. Elsevier, Amsterdam (1995)
21. Müller, H.: On edge perfectness and classes of bipartite graphs. Discrete Math. **149**, 159–187 (1996)
22. Orlin, J.: Contentment in graph theory: covering graphs with cliques. Indagationes Mathematicae **80**, 406–424 (1977)
23. Pullman, N.J.: Clique covering of graphs IV. Algorithms. SIAM Journal on Computing **13**, 57–75 (1984)
24. Raychaudhuri, A.: Intersection number and edge clique graphs of chordal and strongly chordal graphs. Congressus Numer. **67**, 197–204 (1988)
25. Yannakakis, M.: Node-delection problems on bipartite graphs. SIAM Journal on Computing **10**, 310–327 (1981)

Unique Covering Problems with Geometric Sets

Pradeesha Ashok[1], Sudeshna Kolay[1], Neeldhara Misra[2](✉), and Saket Saurabh[1]

[1] Institute of Mathematical Sciences, Chennai, India
{pradeesha,skolay,saket}@imsc.res.in
[2] Indian Institute of Science, Bangalore, India
mail@neeldhara.com

Abstract. The Exact Cover problem takes a universe U of n elements, a family $\mathcal{F}$ of m subsets of U and a positive integer k, and decides whether there exists a subfamily(set cover) $\mathcal{F}'$ of size at most k such that each element is covered by exactly one set. The Unique Cover problem also takes the same input and decides whether there is a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that at least k of the elements $\mathcal{F}'$ covers are covered uniquely(by exactly one set). Both these problems are known to be NP-complete. In the parameterized setting, when parameterized by k, Exact Cover is W[1]-hard. While Unique Cover is FPT under the same parameter, it is known to not admit a polynomial kernel under standard complexity-theoretic assumptions.

In this paper, we investigate these two problems under the assumption that every set satisfies a given geometric property Π. Specifically, we consider the universe to be a set of n points in a real space $\mathbb{R}^d$, d being a positive integer. When $d = 2$ we consider the problem when Π requires all sets to be unit squares or lines. When $d > 2$, we consider the problem where Π requires all sets to be hyperplanes in $\mathbb{R}^d$. These special versions of the problems are also known to be NP-complete. When parameterizing by k, the Unique Cover problem has a polynomial size kernel for all the above geometric versions. The Exact Cover problem turns out to be W[1]-hard for squares, but FPT for lines and hyperplanes. Further, we also consider the Unique Set Cover problem, which takes the same input and decides whether there is a set cover which covers at least k elements uniquely. To the best of our knowledge, this is a new problem, and we show that it is NP-complete (even for the case of lines). In fact, the problem turns out to be W[1]-hard in the abstract setting, when parameterized by k. However, when we restrict ourselves to the lines and hyperplanes versions, we obtain FPT algorithms.

1 Introduction

The classic Set Cover problem is the following: For a set system $(U, \mathcal{F})$ where U is a finite universe of n elements and $\mathcal{F}$ is a family of subsets of U, is there a

N. Misra—Supported by the INSPIRE Faculty Scheme, DST India (project DSTO-1209).

DOI: 10.1007/978-3-319-21398-9_43

sub family of at most k sets in $\mathcal{F}$ whose union is U. We say that an element x in U is *covered* by a set S from $\mathcal{F}$ if the set S contains the element x.

For several applications, it turns out that we would like to not only cover elements of U using sets in $\mathcal{F}$, but also cover them uniquely. A common motivation involves problems where covering elements by more than one set leads to noise (for example, wireless networks), so we would like to ensure that an element is covered, but by only one of the sets. This desired refinement manifests itself in the following three natural variations of the Set Cover problem.

– All elements must be covered uniquely by at most k sets. (Exact Cover)
– All elements must be covered, *and* at least k elements must be covered uniquely. (Unique Set Cover)
– At least k elements are covered uniquely. (Unique Cover)

In the first two variants, we are looking for a set cover with additional properties. Note that in the last setting, a valid solution may not be a set cover.

The Exact Cover problem was one of the twenty-one problems shown to be NP-complete by Karp [6]. The Unique Cover problem was introduced by Demaine et al in [1], and it may be considered a natural "maximization" variant of Set Cover, and also a generalization of the Max Cut problem. The Unique Set Cover problem combines elements of both these variants, and is NP-complete as well.

The Geometric Setting. Geometric settings are among the most promising contexts for developing improved algorithms when faced with hardness in a general setting. The geometric nature of the problem opens up several algorithmic possibilities, and this is amply evidenced in the context of approximation algorithms. Many geometric problems are known to admit good approximation algorithms, even PTASes. In particular, the classical set cover and hitting set problems have been very well-explored in the context of geometric objects [5,12]. In this situation, the universe is a point set in d-dimensional Euclidean space, and the sets are defined by intersection of geometric objects with the point set. An object covers a point if it contains it. We study the unique coverage variants for several geometric objects, including lines, hyperplanes, squares, and rectangles.

Our Approach. In this work, we focus on the parameterized complexity of these problems, both in the general abstract setting and carefully considered special cases. In parameterized complexity each problem instance comes with a parameter k and a central notion in parameterized complexity is *fixed parameter tractability (FPT).* This means, for a given instance (x, k), solvability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. The parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm (the degree of polynomial is independent of k), called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by a polynomial $p(k)$ in k, while preserving the answer.

Table 1. A summary of our results

	Exact Cover		Unique Cover	Unique Set Cover
Parameter	Size of solution		Number of elements uniquely covered	
Abstract Sets	W[1]-hard		FPT	W[1]-hard
			Quadratic Element Kernel	
Lines	FPT	VC-Dimension	FPT	FPT
	Quadratic Kernel		Quadratic Kernel	Poly Kernel
Hyperplanes $\mathbb{R}^d$	$k^{O(d^2)}$ kernel		$k^{O(d)}$ Instance Kernel (Quadratic Element Kernel)	$k^{O(d^2)}$ kernel
Unit Squares	W[1]-hard		Poly Kernel	Open

Studying the parameterized complexity of geometric problems has interesting implications. On the one hand, a tractability result demonstrates the utility of the geometric structure in contrast with the abstract setting. On the other, a hardness result often has consequences for hardness of approximation; usually it establishes evidence for the non-existence of the EPTAS. This has motivated several studies of geometric problems from a parameterized perspective [8].

Our Results. In this work, we establish the following results, summarized also in Table 1.

Exact Cover. We show that EXACT COVER is W[1]-hard even in the restricted setting where all the objects are unit squares (Lemma 1). On the positive side, we show that EXACT COVER is FPT for lines (Lemma 2). Further, if the objects are hyperplanes in a d-dimensional Euclidean space, the EXACT COVER continues to be FPT parameterized by k and d (Lemma 3).

Unique Cover. For UNIQUE COVER, a simple argument shows that the number of elements in the universe can be bounded by $O(k^2)$ (Lemma 4). This shows that the problem is FPT. It turns out that this also implies a polynomial kernel for various geometric objects (Corollary 2), using the fact that these objects have bounded VC Dimension.

Unique Set Cover. We show that UNIQUE SET COVER is W[1]-hard in the general setting (Lemma 5) and NP-complete when restricted to lines (Lemma 6). On the positive side, we show that the problem is FPT for families of bounded intersection (Lemma 7) and hyperplanes in d dimensions (Lemma 8).

2 Preliminaries

Parameterized Complexity. A parameterized problem Π is a subset of $\Sigma^* \times \mathbb{N}$. Given a parameterized decision problem with input $x \in \Sigma^*$ of size n, and an integer parameter k, the goal in parameterized complexity is to design a deterministic algorithm which decides the membership of the instance (x, k) in Π in time $f(k)n^{\mathcal{O}(1)}$, where f is a function of k alone. Problems which admit such algorithms are said to be fixed parameter tractable (FPT). We call an algorithm, with a running time of $f(k)n^{\mathcal{O}(1)}$, an FPT algorithm, and such a running time, an FPT running time. The theory of parameterized complexity was developed by Downey and Fellows [3]. For recent developments, see the book by Flum and Grohe [4].

Definition 1. *A parameterized problem Π FPT-many-one reduces to another parameterized problem Γ, if there is a polynomial p, computable functions $f, g : N \rightarrow N$, and a Turing machine T such that, given any input instance (x, k), T outputs an instance (x', k') within $f(k)p(|x|)$ time, with $(x, k) \in \Pi$ if and only if $(x', k') \in \Gamma$, and $k' \leq g(k)$.*

There is a hierarchy of problems in parameterized complexity. For the purpose of this paper we will define the class $W[1]$ with respect to a hard problem in this class. The class $W[1]$ is the class of all parameterized problems that FPT-many-one reduce to the k-Clique problem parameterized by k. A parameterized problem is W[1]-hard if there is a FPT-many-one reduction from k-Clique, parameterized by k, to the given problem. It is widely believed that FPT $\subset$ W[1]. For further understanding of the various parameterized classes, refer to Flum and Grohe [4].

Kernelization. A *kernelization* algorithm for a parameterized problem Π is a polynomial time procedure which takes as input an instance (x, k), where k is the parameter, and returns an instance (x', k') such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and $|x'| \leq g(k)$ and $k' \leq h(k)$, for some computable functions g, h. The returned instance (x', k') is said to be the kernel for the instance (x, k) of Π.

Problem Definitions. A set system is a pair $(U, \mathcal{F})$, where U is a universe of n elements and $\mathcal{F}$ is a family of m subsets of U. Given a set system $(U, \mathcal{F})$, a set S is said to cover an element $p \in U$ if $p \in S$. An element p is said to be covered *uniquely* by a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ if there is exactly one set in $\mathcal{F}'$ which contains p. The SET COVER problem asks for a smallest collection of subsets whose union covers every element in the universe. We are now ready to define some of the variations of this problem that we consider in our work.

EXACT COVER **Parameter:** k
Input: A set system $(U, \mathcal{F})$ of n elements and m sets, and a positive integer k.
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k that covers every element of U such that each element is contained in exactly one set in $\mathcal{F}'$?

UNIQUE COVER **Parameter:** k
Input: A set system $(U, \mathcal{F})$ of n elements and m sets, and a positive integer k.
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that, among the set of elements covered by $\mathcal{F}'$, there is a subset $S \subset U$, $|S| \geq k$ with each element of S being contained in exactly one set in $\mathcal{F}'$?

Unique Set Cover **Parameter:** k
Input: A set system $(U, \mathcal{F})$ of n elements and m sets, and a positive integer k.
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ that covers every element of U such that there is a subset $S \subseteq U$, $|S| \geq k$ where each element of S is contained in exactly one set in $\mathcal{F}'$?

We consider some notions that will be useful in defining special set systems that we will encounter during our study of these problems. A set system $(U, \mathcal{F})$ is said to have *bounded intersection* when there is a universal constant c such that for any pair of sets $F_1, F_2 \in \mathcal{F}$, $|F_1 \cap F_2| \leq c$. A set system $(U, \mathcal{F})$ is said to be a set system of squares (lines) if U is a subset of n points in $\mathbb{R}^2$ and each set $F \in \mathcal{F}$ is the maximal set of points of U that are contained in a square(line) defined on $\mathbb{R}^2$. Similarly, a set system $(U, \mathcal{F})$ is said to be a set system of hyperplanes if U is a set of n points in $\mathbb{R}^d$, for a fixed positive integer d, and each set $F \in \mathcal{F}$ is the maximal set of points of U that are contained in a hyperplane defined on $\mathbb{R}^d$.

Given a set system $(U, \mathcal{F})$ of n elements and m sets, for every subset $A \subseteq U$ we define the family of sets $\mathcal{F}_A = \{S \cap A | S \in \mathcal{F}\}$.

Definition 2 (VC Dimension). *Let $(U, \mathcal{F})$ represent a set system. A subset $A \subseteq U$ is said to be* shattered *if for every $B \subseteq A$, there exists $F \in \mathcal{F}$ such that $F \cap A = B$. The* Vapnik-Chervonenkis dimension*(or* VC dimension *of $(U, \mathcal{F})$ is the supremum of the sizes of all shattered subsets of U.*

Therefore, in general, the VC dimension of a set system could be infinite. However, set systems of several geometric objects are known to have bounded VC dimension. We refer the reader to [9] for further details on VC Dimension. The following result is known from [13] about set systems of finite VC dimension.

Proposition 1. *Let $(U, \mathcal{F})$ be a set system with $|U| = n$ and VC dimension d. Then $|\mathcal{F}| \leq \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{d}$*

Hyperplanes An i-flat in $\mathbb{R}^d$ is the affine hull of $i+1$ affinely independent points. The dimension of a (possibly infinite) set of points P, denoted as $dim(P)$, is the minimum i such that the entire set P is contained in an i-flat of $\mathbb{R}^d$ [7].

Observation 1. *[7] For a pair of i-flat H_1 and j-flat H_2 , $1 \leq j, i \leq d-1$, if $H_1 \not\subset H_2$ and $H_1 \not\subset H_2$, then $dim(H_1 \cap H_2) < min\{i, j\}$.*

In this paper we refer to $(d-1)$-flats of $\mathbb{R}^d$ as hyperplanes.

3 Exact Cover

In this section, we consider the Exact Cover problem, parameterized by the number k of sets in a solution family. Since this problem is known to be W[1]-hard, it is natural to introduce properties on the input set family and see whether the added structure makes the problem easier in these special cases. Here, we restrict ourselves to geometric versions, where the universe is a set of points in a real space $\mathbb{R}^d$, for an appropriate integer d, while the set family is such that every set satisfies a particular geometric property.

The W[1]-hardness of Exact Cover was shown in [11]. We give an alternative proof for this W[1]-hardness. This proof, with a little bit of modification, shows that Exact Cover on set systems of unit squares is also W[1]-hard.

Proposition 2 ($\star$). Exact Cover *is* W[1]-*hard.*[1]

Lemma 1 ($\star$). Exact Cover *on set systems of unit squares is* W[1]-*hard.*

Kernels for Exact Cover *with Lines and Hyperplanes.* In contrast with the hardness results that we have seen so far, we now turn to some algorithmic results. First, when we consider our input universe to be a set of n points in $\mathbb{R}^2$ and our sets to be maximal sets of collinear points, we obtain a quadratic kernel using a "high degree" reduction rule. This version of Exact Cover is also NP-complete, and the proof for NP-hardness is very similar to the proof of Lemma 6 given in the Appendix.

Let $(P, \mathcal{L})$ be the input set system. In our discussion, we use the terms sets and lines interchangeably. The input family could contain sets containing single points. Our first reduction rule is taken directly from [7]. We remove all lines (but for one) that pass through exactly one point.

Reduction Rule 1. *For any input point p, from the set $\mathcal{L}_p = \{L | L \in \mathcal{L}$ and $L \cap P = \{p\}\}$, delete all lines but one (say L_p).*

Proposition 3. *Reduction Rule 1 is sound.*

Reduction Rule 2. *In an instance$(P, \mathcal{L}, k)$, if there is a line L that contains at least $k+1$ input points then delete L and all lines intersecting with points on L and decrease k by one. All points contained in L are deleted from the universe U. Formally, the reduced instance is $(P \setminus L, \mathcal{L} \setminus \{T \mid L \cap T \neq \emptyset\}, k-1)$.*

A similar reduction was given in [7] to exhibit a polynomial kernel for Point Line Cover. We show the correctness of this reduction in this case.

Claim. Reduction Rule 2 is sound.

Proof. Suppose there is a solution, $\mathcal{L}'$, for $(P, \mathcal{L}, k)$ which does not contain L. Since $\mathcal{L}'$ is a set cover of size at most k that excludes L, it includes at least $(k+1)$ other lines to cover the points on L, as two lines intersect at at most one point. This contradicts that $\mathcal{L}'$ is a solution of the instance $(P, \mathcal{L}, k)$. Note that this argument shows that *any* valid solution of the instance $(P, \mathcal{L}, k)$ must contain L. Now, consider the subfamily of $\mathcal{L}$ comprising of lines that contain points belonging to L:

$$\mathcal{L}'' := \{T \in \mathcal{L} \mid T \neq L, T \cap L \neq \emptyset\}.$$

[1] The proofs of results marked with a $\star$ are in the Appendix.

Clearly, any solution that contains L does not contain any element of $\mathcal{L}''$, therefore, it is safe to remove these sets from the instance, establishing the correctness of Reduction Rule 2. $\square$

The reduction is very robust, in the sense that a line as described above will belong to any solution of the Exact Cover instance. On exhaustive application of this reduction, a YES instance can have at most k^2 remaining input points, since k lines can only cover k^2 points when each line has at most k points.

Therefore, if our reduced instance has more than k^2 points, we correctly return NO. Otherwise, due to Reduction Rule 1 and by the property of lines, we can also bound the number of lines in the reduced instance to at most k^4. Thus, we have shown the following.

Lemma 2. Exact Cover *on set systems of lines is* FPT*, with a polynomial kernel.*

A natural question is to consider input instances of Exact Cover which are set systems of hyperplanes in $\mathbb{R}^d$. We parameterize the Exact Cover problem in this case by $k+d$. The following Lemma is obtained by extending the reduction rules in [7]. In particular, it is shown in [7] that for any $1 \leq i \leq d-1$, if an i-flat covers more than $k^i + 1$ points, then these points can be replaced with one representative. The crux of the argument is that all of these points are covered “together” by a single hyperplane in any valid solution. In our argument, we further this reduction rule by deleting all hyperplanes that contain a strict subset of these points, because such hyperplanes are automatically forbidden from being a part of any valid solution of Exact Cover. We refer the reader to the Appendix and [7] for further details.

Lemma 3 ($\star$). Exact Cover *on set systems of hyperplanes in* $\mathbb{R}^d$ *is* FPT *parameterized by* $k+d$*.*

4 Unique Cover

The Unique Cover problem was studied in [11] and was found to be FPT. However, the problem does not have a polynomial kernel unless $NP \subseteq coNP/poly$, as shown in [2]. In this section, we exhibit polynomial sized kernels for several geometric versions, exploiting the geometric property satisfied by each set of the input set system.

Recall that the Unique Cover problem is parameterized by the number of elements that we desire to cover uniquely. To begin with, in the abstract setting, we show that the number of elements in the universe can be bounded by k^2. Note that it is straightforward to bound the sizes of the individual sets in an instance of Unique Cover, with the following observation.

Observation 2. *If there exists* $F_i \in \mathcal{F}$ *such that* $|F_i| \geq k$*, then the given instance is a* YES *instance, with* $\{F_i\}$ *serving as a valid solution.*

We now turn to an argument for bounding the size of the universe in an instance of Unique Cover.

Lemma 4 (⋆). Unique Cover *admits a quadratic element kernel.*

The following result regarding sets of bounded VC Dimension are now implied by Proposition 1 and Lemma 4.

Corollary 1. Unique Cover *on set systems of VC Dimension bounded by a constant d admits a polynomial kernel.*

As an immediate consequence, we get the existence of polynomial kernels in special geometric cases, since these geometric set families have constant VC Dimension.

Corollary 2 (⋆). Unique Cover *admits a polynomial kernel for set systems of lines, hyperplanes, axis-parallel rectangles and disks.*

5 Unique Set Cover

We show that Unique Set Cover is W[1]-hard. However, as with the other problems, assuming geometric properties on the set family provides positive algorithmic results.

Lemma 5 (⋆). Unique Set Cover *is* W[1]*-hard.*

The reduction also shows that Unique Set Cover is NP-hard. In fact, even when we consider the special case when the universe U is a set of n points in $\mathbb{R}^2$ and each set is a line, the Unique Set Cover problem turns out to be NP-hard, as we show in our next lemma.

Lemma 6 (⋆). Unique Set Cover *on set systems of lines is* NP*-complete.*

This NP-hardness reduction is very similar to the NP-hardness reduction for Point Line Cover [10]. Unlike [10], we reduce the problem from 1-IN-3-SAT, instead of 3-SAT.

This implies that Unique Set Cover on set systems with bounded intersection and Unique Set Cover on set systems of hyperplanes are also NP-hard. In the parameterized context, although the problem is W[1]-hard in the general setting, we look at some special cases where this problem can be solved in FPT time. First, we make a few observations.

Observation 3. *Let $\mathcal{F}'$ be a solution for* Unique Set Cover*. Then any minimal set cover contained in $\mathcal{F}'$ is also a solution for* Unique Set Cover*.*

Thus, it is enough to find a minimal set cover that covers at least k elements uniquely.

Observation 4. *Any minimal set cover of size at least k is a* Unique Set Cover *solution for a given instance. In particular, if the minimum set cover of the given instance is of size at least k then it is a* YES *instance for the* Unique Set Cover *problem.*

Now, we turn to algorithmic results for special cases of Unique Set Cover.

Sets of Bounded Intersection. First, we consider set systems $(U, \mathcal{F})$ where $\mathcal{F}$ has the property that for any pair of sets $F_1, F_2 \in \mathcal{F}$, $|F_1 \cap F_2| \leq c$.

Lemma 7. Unique Set Cover *for sets of bounded intersection c is* FPT *when parameterized by $c + k$.*

Proof. Construct a minimal set cover $\mathcal{S}$ for the instance. If the size of $\mathcal{S}$ is at least k, then it is a solution for Unique Set Cover, by Observation 4. Similarly, if there is a set $S \in \mathcal{S}$ that contains at least k private elements, then too $\mathcal{S}$ is a solution for Unique Set Cover. Suppose there are at most $k-1$ sets in $\mathcal{S}$ and each set has at most $k-1$ private elements. By pigeonhole principle, there must be a set $S \in \mathcal{S}$ which contains at least $n/(k-1)$ elements of U. The number of elements in S that can belong to other sets of $\mathcal{S}$ is at most $c(k-2)$, because of the bounded intersection property. If $\frac{n}{k-1} - c(k-2) \geq k$ then the given instance is a YES instance. Otherwise, $\frac{n}{k-1} - c(k-2) \leq k-1$, which implies that $n \leq (1+c)(k-1)^2$.

Since the sets have bounded pairwise intersection of at most c, any subset of $c+1$ elements can appear together in at most one set. Therefore, the number of sets are bounded by $n^{c+1} \leq ((1+c)(k-1)^2)^{c+1}$. We can now guess the uniquely covered elements, and the distribution of the uniquely covered elements in a Unique Set Cover solution. Finally, we can check whether there are sets in $\mathcal{F}$ to validate the guess in polynomial time. As the number of guesses is an FPT function, the running time of this algorithm is FPT. □

Hyperplanes in $\mathbb{R}^d$. Next, we consider a geometric set system $(U, \mathcal{F})$ where U is a set of n points in $\mathbb{R}^d$ and sets in $\mathcal{F}$ are defined by hyperplanes in $\mathbb{R}^d$. When $d = 2$, these are lines and this is a special case of sets with bounded intersection. For $d > 2$, hyperplanes do not have this property. Nonetheless, we obtain an FPT algorithm for hyperplanes, by reducing the given instance to an instance of Unique Set Cover for sets with bounded intersection.

Lemma 8. Unique Set Cover *on set systems of hyperplanes in $\mathbb{R}^d$ is* FPT.

Proof. Let U be the universe of n elements and $\mathcal{F}$ be the family of m hyperplanes in $\mathbb{R}^d$. The following Reduction Rule aims at reducing the number of points, while maintaining the Unique Set Cover solution if there exists one.

Reduction Rule 3. *for i from 1 to $d-1$:*
Suppose $P \subseteq U$ is a set of at least k^i points such that $dim(P) = i$, and P is contained in at least one hyperplane of $\mathcal{F}$. Suppose $\mathcal{F}_P \subseteq \mathcal{F}$ is the set of all hyperplanes that contain P. If $\mathcal{F} \setminus \mathcal{F}_P$ is a set cover for U, then we say YES *for our input instance and exit. Otherwise, we delete all but k^i points of P from the universe. If a hyperplane becomes empty, we delete that hyperplane from $\mathcal{F}$.*

We prove the correctness of this reduction rule by induction on i. When $i = 1$, P is a line with more than k points. We abuse notation and also use P to refer to this collection of points. Suppose $\mathcal{F} \setminus \mathcal{F}_P$ is a set cover for the instance, let $\mathcal{G}$ be a minimal set cover obtained from $\mathcal{F} \setminus \mathcal{F}_P$. Then, by Observation 1, any set in $\mathcal{G}$ can contain at most one point of P. To cover all elements of P, there must be at least $k+1$ sets in $\mathcal{G}$. By Observation 4, we correctly say YES. If no such set cover exists, then we know that any set cover for the input instance must contain at least one hyperplane from $\mathcal{F}_P$. Let $P' \subset P$ be the set of all but k points that are deleted by the Reduction Rule and $P'' = P \setminus P'$. Also, let $\mathcal{F}'$ be the family of hyperplanes that became empty and got deleted from $\mathcal{F}$. Suppose $\mathcal{G} = \{H_1, \dots, H_l\}$ is a minimal solution for $(U, \mathcal{F}, k)$. Since P is a line with more than k points, there must be at least one H_i that contains all of P. Now, the following cases can occur:

1. Suppose there are two planes $H_i, H_j, 1 \leq i \neq j \leq l$, both of which contain the set P. Then none of the points in P are uniquely covered by this solution. The points which are uniquely covered are not deleted as a result of this Reduction Rule. Also, by definition of minimality, no hyperplane of $\mathcal{G}$ could have become empty after this Reduction Rule was applied. Hence, $\mathcal{G}$ remains a solution for Unique Set Cover in $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.
2. Suppose $l > k$. Since we have dealt with the case when P is covered be at least 2 hyperplanes of $\mathcal{G}$, we can assume that there is exactly one hyperplane in $\mathcal{G}$ that contains P. There are at least k remaining hyperplanes in $\mathcal{G}$. Since, $\mathcal{G}$ is a minimal solution for Unique Set Cover, each of these remaining hyperplanes cover a point uniquely, and none of these uniquely covered points belong to P. Hence, $\mathcal{G}$ remains a solution for $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.
3. Finally, suppose $l \leq k$, and as before, let $H \in \mathcal{G}$ be the hyperplane that contains all points in P. Then at most $l-1$ points of P are not uniquely covered. All other points of P must be uniquely covered. In particular, at most $l-1$ points of $P \setminus P'$ are not uniquely covered , which implies that at least $k-l+1$ points in $P \setminus P'$ are uniquely covered by $\mathcal{G}$. By minimality, for each hyperplane H' in $\mathcal{G} \setminus \{H\}$ there is a point $p_{H'}$ that is uniquely covered by H'. Thus, at least $k-l+1$ points from $P \setminus P'$ and $l-1$ points from $\mathcal{G} \setminus \{H\}$ are covered uniquely. Again, by minimality, no hyperplane of $\mathcal{G}$ could have become empty because of application of the Reduction Rule. Hence, $\mathcal{G}$ is a solution in $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

On the other hand, let $\mathcal{G}'$ be a minimal solution for $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$. Assume $\mathcal{G}'$ is not a solution for $(U, \mathcal{F}, k)$. Then each point in P'' is covered by a different set in $\mathcal{G}'$. Let this subfamily of $\mathcal{G}'$, with at least k hyperplanes, be $\mathcal{H}'$.Let $G \in \mathcal{F}_P$. Consider $\mathcal{G}' \cup G$, which is clearly a set cover for $(U, \mathcal{F})$. Let $\mathcal{S} \subset \mathcal{G}' \cup G$ be a minimal set cover of $(U, \mathcal{F})$. Suppose $P_1 \subseteq (U \setminus P')$ was a set of k points, such that for each hyperplane H in $\mathcal{H}'$ there is a point in P_1 that it uniquely covers with respect to $\mathcal{G}'$. Let $P_2 \subseteq P_1$ be uniquely covered by $\mathcal{S}$. Each point in $P_1 \setminus P_2$ has exactly one hyperplane in $\mathcal{S}$, other than G, containing it. By the minimality of $\mathcal{S}$, this hyperplane has a point that is uniquely covered by it. Therefore, for all of the points in $P_2 \setminus P_1$, either that point is uniquely covered by G or a hyperplane (other than G) in $\mathcal{S}$ containing it is uniquely covering another point. Therefore $\mathcal{S}$ still covers at least k points uniquely and is a solution for $(U, \mathcal{F}, k)$.

Now, assume $i > 1$ and the Induction Hypothesis is true for all $j < i$. By arguments similar to the base case, the reduction rule is sound for i. The full proof is in the Appendix. □

We exhaustively apply this Reduction Rule. At the end, any hyperplane contains at most k^{d-1} points. Let $\mathcal{G}$ be a minimal set cover for the instance. If there are at least $k+1$ hyperplanes in $\mathcal{G}$, then due to Observation 4, we correctly say YES. Otherwise, there are at most k hyperplanes in the set cover $\mathcal{G}$, which implies that $|U| \leq k^{(d-1)} \cdot k$. The number of hyperplanes that can contain these points is at most $(k^d)^d$. Thus, we have a kernel for the problem. For the algorithm, we guess k points $P \subseteq U$ that are uniquely covered by a solution and the family $\mathcal{G}$ of at most k hyperplanes that are responsible for this unique coverage. Let $\mathcal{F}^P = \{H | H \in \mathcal{F} \setminus \mathcal{G}, \exists p \in P \text{ s.t } p \in H\}$. We check whether the family $\mathcal{F} \setminus \mathcal{F}^P$ is a set cover or not. There are at most $\mathcal{O}(k^{kd^2})$ possible pairs $(P, \mathcal{G})$. Thus the problem is FPT. □

References

1. Demaine, E.D., Feige, U., Hajiaghayi, M., Salavatipour, M.R.: Combination Can Be Hard: Approximability of the Unique Coverage Problem. SIAM J. Comput. () **38**(4), 1464–1483 (2008)
2. Dom, M., Lokshtanov, D., Saurabh, S.: Kernelization Lower Bounds Through Colors and IDs. ACM Trans. Algorithms **11**(2), 13:1–13:20 (2014). doi:10.1145/2650261
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity, p. 530. Springer-Verlag (1999)
4. Flum, J., Grohe, M.: Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series). Springer-Verlag New York Inc (2006)
5. Hochbaum, D.S., Maass, W.: Fast approximation algorithms for a nonconvex covering problem. Journal of algorithms **8**(3), 305–323 (1987)
6. Karp, R.M.: Reducibility Among Combinatorial Problems. Complexity of Computer Computations, pp. 85–103 (1972)
7. Langerman, S., Morin, P.: Covering things with things. Discrete & Computational Geometry **33**(4), 717–729 (2005)
8. Marx, Dániel: Efficient approximation schemes for geometric problems? In: Brodal, Gerth Stølting, Leonardi, Stefano (eds.) ESA 2005. LNCS, vol. 3669, pp. 448–459. Springer, Heidelberg (2005)
9. Matoušek, J.: Lectures on discrete geometry, vol. 108. Springer, New York (2002)
10. Megiddo, N., Tamir, A.: On the Complexity of Locating Linear Facilities in the Plane. Oper. Res. Lett. **1**(5), 194–197 (1982). doi:10.1016/0167-6377(82)90039-6
11. Misra, N., Moser, H., Raman, V., Saurabh, S., Sikdar, S.: The Parameterized Complexity of Unique Coverage and Its Variants. Algorithmica, 517–544 (2013)
12. Mustafa, N., Ray, S.: PTAS for geometric hitting set problems via local search. In: Proceedings of the 25th annual symposium on Computational geometry, pp. 17–22. ACM (2009)
13. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability & Its Applications **16**(2), 264–280 (1971)

Linear Time Approximation Schemes for Geometric Maximum Coverage

Jian Li[1(✉)], Haitao Wang[2], Bowei Zhang[1], and Ningye Zhang[1]

[1] Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing 100084, China
lijian83@mail.tsinghua.edu.cn, {bw-zh14,zhangny12}@mails.tsinghua.edu.cn
[2] Department of Computer Science, Utah State University, Utah 84322, USA
haitao.wang@usu.edu

Abstract. We study approximation algorithms for the following geometric version of the maximum coverage problem: Let $\mathcal{P}$ be a set of n weighted points in the plane. We want to place m $a \times b$ rectangles such that the sum of the weights of the points in $\mathcal{P}$ covered by these rectangles is maximized. For any fixed $\varepsilon > 0$, we present efficient approximation schemes that can find a $(1-\varepsilon)$-approximation to the optimal solution. In particular, for $m = 1$, our algorithm runs in linear time $O(n \log(\frac{1}{\varepsilon}))$, improving over the previous result. For $m > 1$, we present an algorithm that runs in $O(\frac{n}{\varepsilon}\log(\frac{1}{\varepsilon}) + m(\frac{1}{\varepsilon})^{O(\min(\sqrt{m},\frac{1}{\varepsilon}))})$ time.

Keywords: Maximum coverage · Geometric set cover · Polynomial-time approximation scheme

1 Introduction

The maximum coverage problem is a classic problem in theoretical computer science and combinatorial optimization. In this problem, we are given a universe $\mathcal{P}$ of weighted elements, a family of subsets and a number k. The goal is to select at most k of these subsets such that the sum of the weights of the covered elements in $\mathcal{P}$ is maximized. It is well-known that the most natural greedy algorithm achieves an approximation factor of $1 - 1/e$, which is essentially optimal (unless P=NP) [17,20,25]. However, for several geometric versions of the maximum coverage problem, better approximation ratios can be achieved (we will mention some of such results below). In this paper, we mainly consider the following geometric maximum coverage problem:

Jian Li, Bowei Zhang and Ningye Zhang's research was supported in part by the National Basic Research Program of China Grant 2015CB358700, 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61202009, 61033001, 61361136003. Haitao Wang's research was supported in part by NSF under Grant CCF-1317143.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 559–571, 2015.
DOI: 10.1007/978-3-319-21398-9_44

Definition 1. *($\mathsf{MaxCov_R}(\mathcal{P}, m)$) Let $\mathcal{P}$ be a set of n points in a 2-dimensional Euclidean plane $\mathbb{R}^2$. Each point $p \in \mathcal{P}$ has a given weight $w_p \geq 0$. The goal of our geometric max-coverage problem (denoted as $\mathsf{MaxCov_R}(\mathcal{P}, m)$) is to place m $a \times b$ rectangles such that the sum of the weights of the covered points by these rectangles is maximized. More precisely, let S be the union of m rectangles we placed. Our goal is to maximize*

$$\mathsf{Cover}(\mathcal{P}, S) = \sum_{p \in \mathcal{P} \cap S} w_p.$$

We also study the same coverage problem with unit disks, instead of rectangles. We denote the corresponding problem as $\mathsf{MaxCov_D}(\mathcal{P}, m)$. One natural application of the geometric maximum coverage problem is the facility placement problem. In this problem, we would like to locate a certain number of facilities to serve the maximum number of clients. Each facility can serve a region (depending on whether the metric is L_1 or L_2, the region is either a square or a disk).

1.1 $m = 1$

Previous Results: We first consider $\mathsf{MaxCov_R}(\mathcal{P}, 1)$. Imai and Asano [21], Nandy and Bhattacharya [24] gave two different exact algorithms for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$, both running in time $O(n \log n)$. It is also known that solving $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ exactly in algebraic decision tree model requires $\Omega(n \log n)$ time [4]. Tao et al. [26] proposed a randomized approximation scheme for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$. With probability $1 - 1/n$, their algorithm returns a $(1 - \varepsilon)$-approximate answer in $O(n \log(\frac{1}{\varepsilon}) + n \log \log n)$ time. In the same paper, they also studied the problem in the external memory model.

Our Results: For $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ we show that there is an approximation scheme that produces a $(1 - \varepsilon)$-approximation and runs in $O(n \log(\frac{1}{\varepsilon}))$ time, improving the result by Tao et al. [26].

1.2 General $m > 1$

Previous Results: Both $\mathsf{MaxCov_R}(\mathcal{P}, m)$ and $\mathsf{MaxCov_D}(\mathcal{P}, m)$ are NP-hard if m is part of the input [22]. The most related work is de Berg, Cabello and Har-Peled [12]. They mainly focused on using unit disks (i.e., $\mathsf{MaxCov_D}(\mathcal{P}, m)$). They proposed a $(1-\varepsilon)$-approximation algorithm for $\mathsf{MaxCov_D}(\mathcal{P}, m)$ with time complexity $O(n(m/\varepsilon)^{O(\sqrt{m})})$.

[1] We note that their algorithm can be easily extended to $\mathsf{MaxCov_R}$ with the same time complexity.

We are not aware of any explicit result for $\mathsf{MaxCov_R}(\mathcal{P}, m)$ for general $m > 1$.

[1] They were mainly interested in the case where m is a constant. So the running time becomes $O(n(1/\varepsilon)^{O(\sqrt{m})})$ (which is the bound claimed in their paper) and the exponential dependency on m does not look too bad for $m = O(1)$. Since we consider the more general case, we make the dependency on m explicit.

It is known [12] that the problem admits a PTAS via the standard shifting technique [19]. [2]

Our Results: Our main result is an approximation scheme for $\mathsf{MaxCov_R}(\mathcal{P}, m)$ which runs in time

$$O\left(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon} + m\left(\frac{1}{\varepsilon}\right)^{\Delta}\right),$$

where $\Delta = O(\min(\sqrt{m}, \frac{1}{\varepsilon}))$. Our algorithm can be easily extended to other shapes. The algorithm for approximating approximating $\mathsf{MaxCov_D}(\mathcal{P}, m)$ can be found in the full version of this paper. [3] The running time of our algorithm is

$$O\left(n\Big(\frac{1}{\varepsilon}\Big)^{O(1)} + m\Big(\frac{1}{\varepsilon}\Big)^{\Delta}\right).$$

Following the convention of approximation algorithms, ε is a fixed constant. Hence, the second term is essentially $O(m)$ and the overall running time is essentially linear $O(n)$. Our algorithm follows the standard shifting technique [19], which reduces the problem to a smaller problem restricted in a constant size cell. The same technique is also used in de Berg et al. [12]. They proceeded by first solving the problem exactly in each cell, and then use dynamic programming to find the optimal allocation for all cells. [4]

Our improvement comes from another two simple yet useful ideas. First, we apply the shifting technique in a different way and make the side length of grids much smaller ($O(\frac{1}{\varepsilon})$, instead of $O(m)$ in de Berg et al.'s algorithm [12]). Second, we solve the dynamic program approximately. In fact, we show that a simple greedy strategy (along with some additional observations) can be used for this purpose, which allows us to save another $O(m)$ term.

1.3 Other Related Work

There are many different variants for this problem. We mention some most related problems here.

Barequet et al. [3], Dickerson and Scharstein [13] studied the max-enclosing polygon problem which aims to find a position of a given polygon to cover maximum number of points. This is the same as $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ if a polygon is a rectangle. Imai et al. [21] gave an optimal algorithm for the max-enclosing rectangle problem with time complexity $O(n \log n)$.

$\mathsf{MaxCov_D}(\mathcal{P}, m)$ was introduced by Drezner [15]. Chazelle and Lee [9] gave an $O(n^2)$-time exact algorithm for the problem $\mathsf{MaxCov_D}(\mathcal{P}, 1)$. A Monte-Carlo

[2] Hochbaum and Maass [19] obtained a PTAS for the problem of covering given points with a minimal number of rectangles. Their algorithm can be easily modified into a PTAS for $\mathsf{MaxCov_R}(\mathcal{P}, m)$ with running time $n^{O(1/\epsilon)}$.

[3] The full version of this paper can be found on CS arXiv.

[4] In fact, their dynamic programming runs in time at least $\Omega(m^2)$. Since they focused on constant m, this term is negligible in their running time. But if $m > \sqrt{n}$, the term can not be ignored and may become the dominating term.

$(1-\varepsilon)$-approximation algorithm for $\mathsf{MaxCov}_\mathsf{D}(\mathcal{P},1)$ was shown in [1], where $\mathcal{P}$ is an unweighted point set. Aronov and Har-Peled [2] showed that for unweighted point sets an $O(n\varepsilon^{-2}\log n)$ time Monte-Carlo $(1-\varepsilon)$-approximation algorithm exists, and also provided some results for other shapes. de Berg et al. [12] provided an $O(n\varepsilon^{-3})$ time $(1-\varepsilon)$-approximation algorithm.

For $m > 1$, $\mathsf{MaxCov}_\mathsf{D}(\mathcal{P},m)$ has only a few results. For $m = 2$, Cabello et al. [7] gave an exact algorithm for this problem when the two disks are disjoint in $O(n^{8/3}\log^2 n)$ time. de Berg et al. [12] gave $(1-\varepsilon)$-approximation algorithms that run in $O(n\varepsilon^{-4m+4}\log^{2m-1}(1/\varepsilon))$ time for $m > 3$ and in $O(n\varepsilon^{-6m+6}\log(1/\varepsilon))$ time for $m = 2, 3$.

The dual of the maximum coverage problem is the classical set cover problem. The geometric set cover problem has enjoyed extensive study in the past two decades. The literature is too vast to list exhaustively here. See e.g., [6,8,10,16,23,27] and the references therein.

2 Preliminaries

We first define some notations and mention some results that are needed in our algorithm. Denote by $G_\delta(a,b)$ the square grid with mesh size δ such that the vertical and horizontal lines are defined as follows

$$G_\delta(a,b) = \{(x,y)\in\mathbb{R}^2 \mid y = b + k\cdot\delta, k\in\mathbb{Z}\} \cup \{(x,y)\in\mathbb{R}^2 \mid x = a + k\cdot\delta, k\in\mathbb{Z}\}$$

Given $G_\delta(a,b)$ and a point $p=(x,y)$, we call the integer pair $(\lfloor x/\delta\rfloor, \lfloor y/\delta\rfloor)$ the *index* of p (the index of the cell in which p lies in).

Perfect Hashing: Dietzfetbinger et al. [14] shows that if each basic algebraic operation (including $\{+,-,\times,\div,\log_2,\exp_2\}$) can be done in constant time, we can get a perfect hash family so that each insertion and membership query takes $O(1)$ expected time. In particular, using this hashing scheme, we can hash the indices of all points, so that we can obtain the list of all non-empty cells in $O(n)$ expected time. Moreover, for any non-empty cell, we can retrieve all points lies in it in time linear in the number of such points.

Linear Time Weighted Median and Selection: It is well known that finding the weighted median for an array of numbers can be done in deterministic worst-case linear time. The setting is as follows: Given n distinct elements $x_1, x_2, ..., x_n$ with positive weights $w_1, w_2, ..., w_n$. Let $w = \sum_{i=1}^n w_i$. The *weighted median* is the element x_k satisfying $\sum_{x_i<x_k} w_i < w/2$ and $\sum_{x_i>x_k} w_i \le w/2$. Finding the kth smallest elements for any array can also be done in deterministic worst-case linear time. See e.g., [11].

An Exact Algorithm for $\mathsf{MaxCov}_\mathsf{R}(\mathcal{P},1)$: As we mentioned previously, Nandy and Bhattacharya [24] provided an $O(n\log n)$ exact algorithm for the $\mathsf{MaxCov}_\mathsf{R}(\mathcal{P},1)$ problem. We are going to use this algorithm as a subroutine in our algorithm.

3 A Linear Time Algorithm for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$

Notations: Without loss of generality, we can assume that $a = b = 1$, i.e., all the rectangles are 1×1 squares, (by properly scaling the input). We also assume that all points are in general positions. In particular, all coordinates of all points are distinct. For a unit square r, we use $w(r)$ to denote the sum of the weights of the points covered by r. We say a unit square r is located at (x, y) if the top-left corner of r is (x, y).

Now we present our approximation algorithm for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$.

3.1 Grid Shifting

Recall the definition of a grid $G_\delta(a, b)$ (in Section 2). Consider the following four grids: $G_2(0,0)$,$G_2(0,1)$,$G_2(1,0)$,$G_2(1,1)$ with $\delta = 2$. We can easily see that for any unit square r, there exists one of the above grids that does not intersect r (i.e., r is inside some cell of the grid). This is also the case for the optimal solution.

Now, we describe the overall framework, which is similar to that in [26]. Our algorithm differs in several details. MaxCovCell(c) is a subroutine that takes a 2×2 cell c as input and returns a unit square r that is a (1-ε)-approximate solution if the problem is restricted to cell c. We present the details of MaxCovCell in the next subsection.

Algorithm 1. $\mathsf{MaxCov_R}(\mathcal{P}, 1)$

```
$w_{\max} \leftarrow 0$
for each $G \in \{G_2(0,0), G_2(0,1), G_2(1,0), G_2(1,1)\}$ do
    Use perfect hashing to find all the non-empty cells of $G$.
    for each non-empty cell $\mathsf{c}$ of $G$ do
        $r \leftarrow$ MaxCovCell($\mathsf{c}$).
        If $w(r) > w_{\max}$, then $w_{\max} \leftarrow w(r)$ and $r_{\max} \leftarrow r$.
    end for;
end for;
return $r_{\max}$;
```

As we argued above, there exists a grid G such that the optimal solution is inside some cell $\mathsf{c}^\star \in G$. Therefore, MaxCovCell($\mathsf{c}^\star$) should return a (1-ε)-approximation for the original problem $\mathsf{MaxCov_R}(\mathcal{P}, 1)$.

3.2 MaxCovCell

In this section, we present the details of the subroutine MaxCovCell. Now we are dealing with the problem restricted to a single 2×2 cell c. Denote the number of point in c by n_c, and the sum of the weights of points in c by W_c. We distinguish two cases, depending on whether n_c is larger or smaller than $\left(\frac{1}{\varepsilon}\right)^2$. If $n_\mathsf{c} < \left(\frac{1}{\varepsilon}\right)^2$, we simply apply the $O(n \log n)$ time exact algorithm. [24]

Algorithm 2. PARTITION($\{x_1, x_2, ..., x_n\}$)

```
Find the weighted median $x_k$ (w.r.t. $w$-weight);
$\mathcal{L} = \mathcal{L} \cup \{x_k\}$;
Generate $S = \{x_i \mid w_i < x_k\}$, $L = \{x_i \mid w_i > x_k\}$;
If the sum of the weights of the points in $S$ is lager than $w_d$, run PARTITION(S);
If the sum of the weights of the points in $L$ is lager than $w_d$, run PARTITION(L);
```

The other case requires more work. In this case, we further partition cell c into many smaller cells. First, we need the following simple lemma.

Lemma 1. *Given n points in $\mathbb{R}^2$ with positive weights $w_1, w_2, ..., w_n$, $\sum_{i=1}^{n} w_i = w$. Assume that $x_1, x_2, ..., x_n$ are their distinct x-coordinates. We are also given a value w_d such that $\max(w_1, w_2, ..., w_n) \leq w_d \leq w$, Then, we can find at most $2w/w_d$ vertical lines such that the sum of the weights of points strictly between (we do not count the points on these lines) any two adjacent lines is at most w_d in time $O(n \log(w/w_d))$.*

Proof. See Algorithm 2. In this algorithm, we apply the weighted median algorithm recursively. Initially we have a global variable $\mathcal{L} = \emptyset$, which upon termination is the set of x-coordinates of the selected vertical lines. Each time we find the weighted median x_k and separate the point with the vertical line $x = x_k$, which we add into $\mathcal{L}$. The sum of the weights of points in either side is at most half of the sum of the weights of all the points. Hence, the depth of the recursion is at most $\lceil \log(w/w_d) \rceil$. Thus, the size of $\mathcal{L}$ is at most $2^{\lceil \log(w/w_d) \rceil} \leq 2w/w_d$, and the running time is $O(n \log(w/w_d))$. □

Now, we describe how to partition cell c into smaller cells. First we partition c with some vertical lines. Let $\mathcal{L}_v$ to denote a set of vertical lines. Initially, $\mathcal{L} = \emptyset$. Let $w_d = \frac{\varepsilon \cdot W_{\mathsf{c}}}{16}$. We find all the points whose weights are at least w_d. For each such point, we add to $\mathcal{L}_v$ the vertical line that passes through the point. Then, we apply Algorithm 2 to all the points with weights less than w_d. Next, we add a set $\mathcal{L}_h$ of horizontal lines in exactly the same way.

Lemma 2. *The sum of the weights of points strictly between any two adjacent lines in $\mathcal{L}_v$ is at most $w_d = \frac{\varepsilon \cdot W_{\mathsf{c}}}{16}$. The number of vertical lines in $\mathcal{L}_v$ is at most $\frac{32}{\varepsilon}$. Both statements hold for $\mathcal{L}_h$ as well.*

Proof. The first statement is straightforward from the description of the algorithm. We only need to prove the upper bound of the number of the vertical lines. Assume the sum of the weights of those points considered in the first (resp. second) step is W_1(resp. W_2), $W_1 + W_2 = W_{\mathsf{c}}$. The number of vertical lines in $\mathcal{L}_v$ is at most

$$W_1 / \left(\frac{\varepsilon \cdot W_{\mathsf{c}}}{16} \right) + 2W_2 / \left(\frac{\varepsilon \cdot W_{\mathsf{c}}}{16} \right) \leq \frac{32}{\varepsilon}.$$

The first term is due to the fact that the weight of each point we found in the first step has weight at least $\frac{\varepsilon \cdot W_{\mathsf{c}}}{16}$, and the second term directly follows from Lemma 1. □

We add both vertical boundaries of cell c into $\mathcal{L}_v$ and both horizontal boundaries of cell c into $\mathcal{L}_h$. Now $\mathcal{L} = \mathcal{L}_v \cup \mathcal{L}_h$ forms a grid of size at most $(\frac{32}{\varepsilon} + 2) \times (\frac{32}{\varepsilon} + 2)$. Assume $\mathcal{L} = \{(x, y) \in \mathbb{R}^2 \mid y = y_j, j \in \{1, ..., u\}\} \cup \{(x, y) \in \mathbb{R}^2 \mid x = x_i, i \in \{1, ..., v\}\}$, with both $\{y_i\}$ and $\{x_i\}$ are sorted. $\mathcal{L}$ partitions c into *small cells*. The final step of our algorithm is simply enumerating all the unit squares located at $(x_i, y_j), i \in \{1, ..., u\}, j \in \{1, ..., v\}$, and return the one with the maximum coverage. However, computing the coverage exactly for all these unit squares is expensive. Instead, we only calculate the weight of these unit square approximately as follows. For each unit square r, we only count the weight of points that are in some small cell fully covered by r. Now, we show this can be done in $O\left(n_{\mathsf{c}} \log\left(\frac{1}{\varepsilon}\right) + \left(\frac{1}{\varepsilon}\right)^2\right)$ time.

After sorting $\{y_i\}$ and $\{x_i\}$, we can use binary search to identify which small cell each point lies in. So we can calculate the sum of the weights of points at the interior, edges or corners of all small cells in $O(n_{\mathsf{c}} \log\left(\frac{1}{\varepsilon}\right))$ times.

Thus searching the unit square with the maximum (approximate) coverage can be done with a standard incremental algorithm in $O\left(\frac{1}{\varepsilon}\right)^2$ time. Due to space constraints, we omit the details which can be found in the full version of this paper.

Putting everything together, we conclude that if $n_{\mathsf{c}} \geq \left(\frac{1}{\varepsilon}\right)^2$, the running time of MaxCovCell(c) is $O\left(n_c \log\left(\frac{1}{\varepsilon}\right) + \left(\frac{1}{\varepsilon}\right)^2\right)$. We can conclude the main result of this section with the following theorem.

Theorem 1. *Algorithm 1 returns a (1-ε)-approximate answer for* $\mathsf{MaxCov}_{\mathsf{R}}(\mathcal{P}, 1)$ *in* $O(n \log\left(\frac{1}{\varepsilon}\right))$ *time.*

Proof. We only show the proof of the approximation guarantee of the algorithm. The complete proof can be found in the full version of this paper. We only need to prove that MaxCovCell(c) returns a (1-ε)-approximation for cell c. The case $n_{\mathsf{c}} < \left(\frac{1}{\varepsilon}\right)^2$ is trivial since we apply the exact algorithm. So we only need to prove the case of $n_{\mathsf{c}} \geq \left(\frac{1}{\varepsilon}\right)^2$.

Suppose the optimal unit square is r. Denote by Opt the weight of the optimal solution. The size of c is 2×2, so we can use 4 unit squares to cover the entire cell. Therefore, $\mathsf{Opt} \geq (\frac{W_{\mathsf{c}}}{4})$. Suppose r is located at a point p, which is in the strict interior of a small cell B separated by $\mathcal{L}$. [5] Suppose the index of B is (i, j). We compare the weight of r with $I(i, j)$ (which is the approximate weight of the unit square located at the top-left corner of B). By the rule of our partition, the weight difference is at most 4 times the maximum possible weight of points between two adjacent parallel lines in $\mathcal{L}$. So $I(i, j) \geq \mathsf{Opt} - 4 \cdot \frac{\varepsilon \cdot W_{\mathsf{c}}}{16} \geq (1 - \varepsilon)\mathsf{Opt}$. This completes the proof. $\square$

[5] If p lies on the boundary of B, the same argument still works.

4 Linear Time Algorithms for $\mathsf{MaxCov_R}(\mathcal{P}, m)$

4.1 Grid Shifting

For general m, we need the shifting technique [19]. Consider grids with a different side length: $G_{6/\varepsilon}(a, b)$. We shift the grid to $\frac{6}{\varepsilon}$ different positions: $(0, 0), (1, 1), ..., (\frac{6}{\varepsilon} - 1, \frac{6}{\varepsilon} - 1)$. (For simplicity, we assume that $\frac{1}{\varepsilon}$ is an integer and no point in $\mathcal{P}$ has an integer coordinate, so points in $\mathcal{P}$ will never lie on the grid line. Let

$$\mathbb{G} = \left\{G_{6/\varepsilon}(0, 0), ..., G_{6/\varepsilon}(6/\varepsilon - 1, 6/\varepsilon - 1)\right\}.$$

The following lemma is quite standard. The proof can be found in the full version of this paper.

Lemma 3. *There exist $G^\star \in \mathbb{G}$ and a $(1-\frac{2\varepsilon}{3})$-approximate solution R such that none of the unit squares in R intersects $G^\star$.*

We present a subroutine in section 4.4 which can approximately solve the problem for a grid, and apply it to each non-empty grid in $\mathbb{G}$. Then, in order to compute our final output from those obtained solutions, we apply a dynamic programming algorithm or a greedy algorithm which are shown in the next two sections.

4.2 Dynamic Programming

Now consider a fixed grid $G \in \mathbb{G}$. Let $\mathsf{c}_1, \ldots, \mathsf{c}_t$ be the cells of grid G and Opt be the optimal solution that does not intersect G. Obviously, $(\frac{6}{\varepsilon})^2$ unit squares are enough to cover an entire $\frac{6}{\varepsilon} \times \frac{6}{\varepsilon}$ cell. Thus the maximum number of unit squares we need to place in one single cell is $m_c = \min\{m, (\frac{6}{\varepsilon})^2\}$.

Let $\mathsf{Opt}(\mathsf{c}_i, k)$ be the maximum weight we can cover with k unit squares in cell c_i. For each nonempty cell c_i and for each $k \in [m_c]$, we find a $(1 - \frac{\varepsilon}{3})$-approximation $\mathsf{F}(\mathsf{c}_i, k)$ to $\mathsf{Opt}(\mathsf{c}_i, k)$. We will show how to achieve this later. Now assume that we can do it.

Let $\mathsf{Opt}_\mathsf{F}(m)$ be the optimal solution we can get from the values $\mathsf{F}(\mathsf{c}_i, k)$. More precisely,

$$\mathsf{Opt}_\mathsf{F}(m) = \max_{k_1,\ldots,k_t \in [m_c]} \left\{ \sum_{i=1}^{t} \mathsf{F}(\mathsf{c}_i, k_i) \;\middle|\; \sum_{i=1}^{t} k_i = m \right\} \tag{1}$$

We can see that $\mathsf{Opt}_\mathsf{F}(m)$ must be a $(1-\frac{\varepsilon}{3})$-approximation to Opt. We can easily use dynamic programming to calculate the exact value of $\mathsf{Opt}_\mathsf{F}(m)$. Denote by $A(i, k)$ the maximum weight we can cover with k unit squares in cells $\mathsf{c}_1, \mathsf{c}_2, ..., \mathsf{c}_i$. We have the following DP recursion:

$$A(i, k) = \begin{cases} \max_{j=0}^{\min(k, m_c)} \{A(i-1, k-j) + \mathsf{F}(\mathsf{c}_i, j)\} & \text{if} \quad i > 1 \\ \mathsf{F}(c_1, k) & \text{if} \quad i = 1 \end{cases}$$

The running time of the above simple dynamic programming is $O(m^2 \cdot m_c)$. One may notice that each step of the DP is computing a $(+, \max)$ convolution. However, existing algorithms (see e.g., [5,28]) only run slightly better than quadratic time. So the improvement would be quite marginal. But in the next section, we show that if we would like to settle for an approximation to $\mathsf{Opt}_\mathsf{F}(m)$, the running time can be dramatically improved to linear.

4.3 A Greedy Algorithm

We first apply our $\mathsf{MaxCov}_\mathsf{R}(\mathcal{P}, 1)$ algorithm in Section 3 to each cell c_i, to compute a $(1 - \frac{\varepsilon^2}{9})$-approximation of $\mathsf{Opt}(\mathsf{c}_i, 1)$. Let $f(\mathsf{c}_i, 1)$ be the return values. [6] This takes $O(n \log \frac{1}{\varepsilon})$ time. Then, we use the selection algorithm to find out the m cells with the largest $f(\mathsf{c}_i, 1)$ values. Assume that those cells are $\mathsf{c}_1, ..., \mathsf{c}_m, \mathsf{c}_{m+1}, ..., \mathsf{c}_t$, sorted from largest to smallest by $f(\mathsf{c}_i, 1)$.

Lemma 4. *Let* Opt' *be the maximum weight we can cover using* m *unit squares in* $\mathsf{c}_1, ..., \mathsf{c}_m$. *Then* $\mathsf{Opt}' \geq (1 - \frac{\varepsilon^2}{9})\mathsf{Opt}$

Proof. Let k be the number of unit squares in Opt that are chosen from $\mathsf{c}_{m+1}, \ldots, \mathsf{c}_t$. This means there must be at least k cells in $\{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$ such that Opt does not place any unit square. Therefore we can always move all k unit squares placed in $\mathsf{c}_{m+1}, \ldots, \mathsf{c}_t$ to these empty cells such that each empty cell contains only one unit square. Denote the weight of this modified solution by A. Obviously, $\mathsf{Opt}' \geq A$. For any i,j such that $1 \leq i \leq m < j \leq t$, we have $\mathsf{Opt}(\mathsf{c}_i, 1) \geq f(\mathsf{c}_i, 1) \geq f(\mathsf{c}_j, 1) \geq (1 - \frac{\varepsilon^2}{9})\mathsf{Opt}(\mathsf{c}_j, 1)$. Combining with a simple observation that $\mathsf{Opt}(\mathsf{c}_i, k) \leq k\mathsf{Opt}(\mathsf{c}_i, 1)$, we can see that $A \geq (1 - \frac{\varepsilon^2}{9})\mathsf{Opt}$. Therefore, $\mathsf{Opt}' \geq (1 - \frac{\varepsilon^2}{9})\mathsf{Opt}$. □

Hence, from now on, we only need to consider the first m cells $\{\mathsf{c}_1, ..., \mathsf{c}_m\}$. We distinguish two cases. If $m \leq 324(\frac{1}{\varepsilon})^4$, we just apply the dynamic program to $\mathsf{c}_1, ..., \mathsf{c}_m$. The running time of the above dynamic programming is $O((\frac{1}{\varepsilon})^{O(1)})$.

If $m > 324(\frac{1}{\varepsilon})^4$, we can use a greedy algorithm to find a answer of weight at least $(1 - \frac{\varepsilon^2}{9})\mathsf{Opt}_\mathsf{F}(m)$.

Let $\mathsf{b} = (\frac{6}{\varepsilon})^2$. For each cell c_i, we find the upper convex hull of 2D points $\{(0, \mathsf{F}(\mathsf{c}_i, 0))$,$(1, \mathsf{F}(\mathsf{c}_i, 1))$, ..., $(\mathsf{b}, \mathsf{F}(\mathsf{c}_i, \mathsf{b}))\}$. See Figure 1. Suppose the convex hull points are $\{(t_{i,0}, \mathsf{F}(\mathsf{c}_i, t_{i,0}))$, $(t_{i,1}, \mathsf{F}(\mathsf{c}_i, t_{i,1}))$, ... , $(t_{i,s_i}, \mathsf{F}(\mathsf{c}_i, t_{i,s_i}))\}$, where $t_{i,0} = 0$,$t_{i,s_i} = \mathsf{b}$. For each cell, since the above points are already sorted from left to right, we can compute the convex hull in $O(\mathsf{b})$ time by Graham's scan[18]. Therefore, computing the convex hulls for all these cells takes $O(m\mathsf{b})$ time.

For each cell c_i, we maintain a value p_i representing that we are going to place t_{i,p_i} squares in cell c_i. Initially for all $i \in [m]$, $p_i = 0$. In each stage, we find the cell c_i such that current slope (the slope of the next convex hull edge)

$$\frac{\mathsf{F}(\mathsf{c}_i, t_{i,p_i+1}) - \mathsf{F}(\mathsf{c}_i, t_{i,p_i})}{t_{i,p_i+1} - t_{i,p_i}}$$

[6] Both $f(\mathsf{c}_i, 1)$ and $\mathsf{F}(\mathsf{c}_i, 1)$ are approximations of $\mathsf{Opt}(\mathsf{c}_i, 1)$, with slightly different approximation ratios.

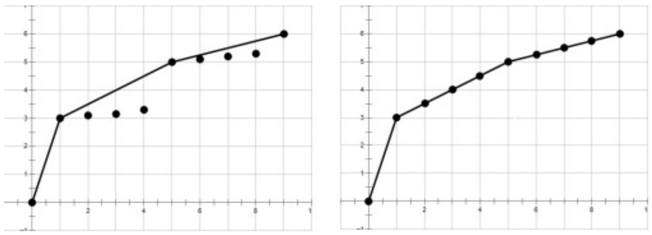


Fig. 1. $\mathsf{F}(\mathsf{c}_i, k)$ (left) and $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ (right)

is maximized. Then we add 1 to p_i, or equivalently we assign $t_{i,p_i+1} - t_{i,p_i}$ more squares into cell c_i. We repeat this step until we have already placed at least $m - \mathsf{b}$ squares. We can always achieve this since we can place at most b squares in one single cell in each iteration. Let m' the number of squares we have placed ($m = \mathsf{b} \leq m' \leq m$). For the remaining $m - m'$ squares, we allocate them arbitrarily. We denote the algorithm by Greedy and let the value obtained be $\mathsf{Greedy}(m')$. Having the convex hulls, the running time of the greedy algorithm is $O(m)$.

Now we analyze the performance of the greedy algorithm.

Lemma 5. *The above greedy algorithm computes an* $(1 - \varepsilon^2/9)$*-approximation to* $\mathsf{Opt}_{\mathsf{F}}(m)$.

Proof. Define an auxiliary function $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ as follows: If $k = t_{i,j}$ for some j, $\widehat{\mathsf{F}}(\mathsf{c}_i, k) = F(\mathsf{c}_i, k)$. Otherwise, suppose $t_{i,j} < k < t_{i,j+1}$, then

$$\widehat{\mathsf{F}}(\mathsf{c}_i, k) = F(\mathsf{c}_i, t_{i,j}) + \frac{F(\mathsf{c}_i, t_{i,j+1}) - F(\mathsf{c}_i, t_{i,j})}{t_{i,j+1} - t_{i,j}} \times (k - t_{i,j}).$$

Intuitively speaking, $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$(See Figure 1) is the function defined by the upper convex hull at integer points. [7] Thus, for all $i \in [m]$, $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ is a concave function. Obviously, $\widehat{\mathsf{F}}(\mathsf{c}_i, k) \geq \mathsf{F}(\mathsf{c}_i, k)$ for all $i \in [m]$ and all $k \in [\mathsf{b}]$.

Let $\mathsf{Opt}_{\widehat{\mathsf{F}}}(i)$ be the optimal solution we can get from the values $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ by placing i squares. By the convexity of $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$, the following greedy algorithm is optimal: as long as we still have budget, we assign 1 more square to the cell which provides the largest increment of the objective value. In fact, this greedy algorithm runs in almost the same way as Greedy. The only difference is that Greedy only picks an entire edge of the convex hull, while the greedy algorithm here may stop in the middle of an edge (only happen for the last edge). Since the marginal increment never increases, we can see that $\mathsf{Opt}_{\widehat{\mathsf{F}}}(i)$ is concave.

By the way of choosing cells in our greedy algorithm, we make the following simple but important observation:

$$\mathsf{Greedy}(m') = \mathsf{Opt}_{\widehat{\mathsf{F}}}(m') = \mathsf{Opt}_{\mathsf{F}}(m').$$

[7] At first sight, it may appear that $\mathsf{F}(\mathsf{c}_i, k)$ should be a concave function. However, this is not true. A counter-example is provided in the full version of this paper.

So, our greedy algorithm is in fact optimal for m'. Combining with $m - m' \leq \mathsf{b}$ and the concavity of $\mathsf{Opt}_{\widehat{\mathsf{F}}}$, we can see that

$$\mathsf{Opt}_{\widehat{\mathsf{F}}}(m') \geq \frac{m - \mathsf{b}}{m} \mathsf{Opt}_{\widehat{\mathsf{F}}}(m) \geq \left(1 - \frac{\varepsilon^2}{9}\right) \mathsf{Opt}_{\mathsf{F}}(m).$$

The last inequality holds because $\mathsf{Opt}_{\widehat{\mathsf{F}}}(i) \geq \mathsf{Opt}_{\mathsf{F}}(i)$ for any i. □

4.4 Computing $\mathsf{F}(\mathsf{c}, k)$

Now we show the subroutine MaxCovCellM for computing $\mathsf{F}(\mathsf{c}, k)$. We use a similar partition algorithm as Section 3.2. The only difference is that this time we need to partition the cell finer so that the maximum possible weight of points between any two adjacent parallel partition lines is $(\frac{\varepsilon^3 W_{\mathsf{c}}}{864})$. After partitioning the cell, we enumerate all the possible ways of placing k unit squares at the grid point. Similarly, for each unit square r, we only count the weight of points that are in some cell fully covered by r.

We can adapt the algorithm in [12] to enumerate these possible choices in $O((\frac{1}{\varepsilon})^{\Delta})$ time where $\Delta = O(\min(\sqrt{m}, \frac{1}{\varepsilon}))$. The details can be found in the full version of this paper. Now we prove the correctness of this algorithm.

Lemma 6. MaxCovCellM *returns a* $(1 - \frac{\varepsilon}{3})$ *approximate answer for* $\mathsf{Opt}(\mathsf{c}_i, k)$.

Proof. We can use $(\frac{6}{\varepsilon})^2$ unit squares to cover the entire cell, so $\mathsf{Opt}(\mathsf{c}_i, k) \geq \frac{k\varepsilon^2 W_{\mathsf{c}}}{72}$. By the same argument as in Theorem 1, the difference between $\mathsf{Opt}(\mathsf{c}_i, k)$ and the answer we got are at most $4k$ times the maximum possible weight of points between two adjacent parallel partition lines. Therefore, the algorithm returns a $(1 - \frac{\varepsilon}{3})$-approximate answer of $\mathsf{Opt}(\mathsf{c}_i, k)$. □

Now we can conclude the following theorem.

Theorem 2. *Let P be a set of n weighted point, for any $0 < \varepsilon < 1$ we can find a $(1 - \varepsilon)$-approximate answer for* $\mathsf{MaxCov}_{\mathsf{R}}(\mathcal{P}, m)$ *in time*

$$O\left(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon} + m \left(\frac{1}{\varepsilon}\right)^{\Delta}\right),$$

where $\Delta = O(\min(\sqrt{m}, \frac{1}{\varepsilon}))$.

The proof is similar to Theorem 1, and it is given in the full version of this paper.

References

1. Agarwal, P.K., Hagerup, T., Ray, R., Sharir, M., Smid, M., Welzl, E.: Translating a planar object to maximize point containment. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, p. 42. Springer, Heidelberg (2002)
2. Aronov, B., Har-Peled, S.: On approximating the depth and related problems. SICOMP **38**(3), 899–921 (2008)
3. Barequet, G., Dickerson, M., Pau, P.: Translating a convex polygon to contain a maximum number of points. Computational Geometry **8**(4), 167–179 (1997)
4. Ben-Or, M.: Lower bounds for algebraic computation trees. In: STOC, pp. 80–86. ACM (1983)
5. Bremner, D., Chan, T.M., Demaine, E.D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Taslakian, P.: Necklaces, convolutions, and $X + Y$. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 160–171. Springer, Heidelberg (2006)
6. Brönnimann, H., Goodrich, M.: Almost optimal set covers in finite vc-dimension. DCG **14**(1), 463–479 (1995)
7. Cabello, S., Díaz-Báñez, J.M., Seara, C., Sellares, J.A., Urrutia, J., Ventura, I.: Covering point sets with two disjoint disks or squares. Computational Geometry **40**(3), 195–206 (2008)
8. Chan, T.M., Grant, E., Könemann, J., Sharpe, M.: Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In: SODA, SODA 2012, pp. 1576–1585. SIAM (2012)
9. Chazelle, B.M., Lee, D.-T.: On a circle placement problem. Computing **36**(1–2), 1–16 (1986)
10. Clarkson, K.L., Varadarajan, K.: Improved approximation algorithms for geometric set cover. DCG **37**(1), 43–58 (2007)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)
12. de Berg, M., Cabello, S., Har-Peled, S.: Covering many or few points with unit disks. Theory of Computing Systems **45**(3), 446–469 (2009)
13. Dickerson, M., Scharstein, D.: Optimal placement of convex polygons to maximize point containment. Computational Geometry **11**(1), 1–16 (1998)
14. Dietzfelbinger, M., Hagerup, T., Katajainen, J., Penttonen, M.: A reliable randomized algorithm for the closest-pair problem. Journal of Algorithms **25**(1), 19–51 (1997)
15. Drezner, Z.: Noteon a modified one-center model. Management Science **27**(7), 848–851 (1981)
16. Even, G., Rawitz, D., Shahar, S.M.: Hitting sets when the vc-dimension is small. IPL **95**(2), 358–362 (2005)
17. Feige, U.: A threshold of ln n for approximating set cover. JACM **45**(4), 634–652 (1998)
18. Graham, R.L.: An efficient algorith for determining the convex hull of a finite planar set. Information processing letters **1**(4), 132–133 (1972)
19. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and vlsi. JACM **32**(1), 130–136 (1985)
20. Hochbaum, D.S., Pathria, A.: Analysis of the greedy approach in problems of maximum k-coverage. Naval Research Logistics **45**(6), 615–627 (1998)
21. Imai, H., Asano, T.: Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. Journal of algorithms **4**(4), 310–323 (1983)

22. Megiddo, N., Supowit, K.J.: On the complexity of some common geometric location problems. SICOMP **13**(1), 182–196 (1984)
23. Mustafa, N.H., Ray, S.: Ptas for geometric hitting set problems via local search. In: SCG, pp. 17–22. ACM (2009)
24. Nandy, S.C., Bhattacharya, B.B.: A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. Computers & Mathematics with Applications **29**(8), 45–61 (1995)
25. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions. Mathematical Programming **14**(1), 265–294 (1978)
26. Tao, Y., Hu, X., Choi, D.-W., Chung, C.-W.: Approximate maxrs in spatial databases. Proceedings of the VLDB Endowment **6**(13), 1546–1557 (2013)
27. Varadarajan, K.: Weighted geometric set cover via quasi-uniform sampling. In: STOC, pp. 641–648. ACM (2010)
28. Williams, R.: Faster all-pairs shortest paths via circuit complexity. In: STOC, pp. 664–673. ACM (2014)

Complexity and Security

Private Certificate-Based Remote Data Integrity Checking in Public Clouds

Huaqun Wang[1](✉) and Jiguo Li[2]

[1] School of Information Engineering, Dalian Ocean University, Dalian, China
wanghuaqun@aliyun.com

[2] College of Computer and Information Engineering, Hohai University, Nanjing, China

Abstract. When the clients store their data in public cloud, the stored data is out of their control. In order to ensure the remote data integrity, the remote data integrity checking (RDIC) is indispensable. In order to overcome the certificate management problem in PKI and the key escrow problem in identity-based setting, we propose a novel remote data integrity checking model, *i.e.*, certificate-based remote data integrity checking (Cert-RDIC). This paper formalizes the private Cert-RDIC model which consists of system model and security model. Then, this paper presents the first private Cert-RDIC protocol whose security is based on the hardness of the standard computational Diffie-Hellman (CDH) problem. Our concrete private Cert-RDIC protocol can also be transformed into public Cert-RDIC protocol. At the same time, by using our proposed private Cert-RDIC protocol, the malicious clients can also be detected by the public cloud servers (PCS).

Keywords: Remote data integrity checking · Certificate-based cryptography · Cloud computing

1 Introduction

Along with the development of networking and computing technology, the cloud computing has become a reality. Cloud computing can be used to process the big data in order to mine the useful information. Cloud computing is a novel and economic computing paradigm. The clients consign the cloud computing server to manage their stored data. Thus, the clients avoid the capital expenditure on hardware, software, and personnel maintenances, *etc.* Nowadays, security problems have become the main barriers which affect the cloud computing development. Due to the lack of physical control over the outsourced data, the clients will be sensitive to their remote data integrity. Secure and efficient remote data integrity checking is an inevitable security issue. At the same time, in order to get rid of the certificate management and key escrow problem, certificate-based cryptography is an important primitive designed to address this issue. In the certificate-based cryptography, this paper studies the remote data integrity model in public clouds.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 575–586, 2015.
DOI: 10.1007/978-3-319-21398-9_45

1.1 Related Work

When the client stores their own data on PCS, RDIC can be used to verify whether PCS keeps the clients' remote data integrity. In 2007, an important probabilistic remote data integrity checking model is proposed by Ateniese *et al.* [1]. Their model is called the provable data possession (PDP). At the same time, they designed two provably secure PDP protocols. By using the PDP model, RDIC drastically reduces communication cost and enhances efficiency. When the clients' data dynamically change, the dynamic RDIC protocol is inevitable [2]. After Ateniese *et al.*'s pioneering work, many different PDP models and protocols were proposed. These PDP protocols can be used in different application environments [3–13]. When the verifier not only can verify the remote data integrity, but also it can also retrieve the remote data, Shacham *et al.* proposed the model of proof of retrievability (POR) which was also studied by Juels *et al.* [14,15]. Many different POR protocols were proposed [16–19]. Based on the role of verifier, RDIC protocol is divided into two cases: private RDIC protocol and public RDIC protocol. In private RDIC protocol, the verifier must possess some secret information. In public RDIC protocol, any client can perform the RDIC protocol. Private RDIC protocols and public RDIC protocols have their respective application fields.

In order to eliminate the burden of certificate management, identity-based cryptography [20] are introduced into the cloud computing data management. In 2013, by using the identity-based cryptography, Han *et al.* proposed identity-based data storage in cloud computing [21,22]. In 2014, Wang *et al.* proposed identity-based remote possession checking in public clouds [23].

By using the cloud computing, the stored remote data can be accessed by the clients with independent geographical locations. The clients maybe some end devices which are mobile and limited in computation and storage. Thus, it is necessary to design the efficient and secure RDIC protocols in cloud computing. Since PDP protocols are more suitable for the clients with mobile end devices, our private Cert-RDIC architecture and protocol are based on the PDP.

1.2 Motivation and Contribution

In PKI architecture, the trusted third party will have to deal with the certificate management which includes certificate generation, certificate usage, and certificate revocation. In order to get rid of the certificate management, identity-based cryptography protocol is introduced into cloud computing. Regretfully, identity-based RDIC protocol suffers from the key escrow problem. This paper focuses on RDIC architecture and protocol in certificate-based cloud storage.

In 2003, Gentry introduced the notion of certificate-based encryption [24]. In this model, a certificate acts not only as a certificate but also as a decryption key. Certificate-based cryptography proposed by Gentry [24] combines the merit of traditional public key cryptography and identity based cryptography, without certificate management problem and key escrow problem. In 2004, Kang *et al.* proposed the security notion of certificate-based signature [25]. Furthermore, Li *et al.* refined the security model of certificate-based signature and proposed two concrete certificate-based signature schemes [26, 27]. In public clouds, remote data integrity checking will face many disadvantages when RDIC protocols were designed in PKI or identity-based setting, such as certificate management, key escrow, *etc.* In order to solve the above problem, we investigate a new RDIC model incorporating certificate-based cryptography, *i.e.*, the private Cert-RDIC model. Our contributions are given from the three respects:

– Firstly, we formalize the private Cert-RDIC model.
– Secondly, we realize the first private Cert-RDIC protocol.
– Thirdly, by making some parameters public, our private Cert-RDIC protocol can be transformed into public Cert-RDIC protocol.

1.3 Paper Organization

The rest of this paper is organized below. Section 2 formalizes the private Cert-RDIC model. Section 3 presents our private Cert-RDIC protocol. Section 4 analyzes the proposed private Cert-RDIC protocol's properties of security and transformation. Finally, Section 5 concludes the paper.

2 Private Cert-RDIC Model

System model and security definition of private Cert-RDIC protocol are given in the section. In the system model, there exist four entities which are described below:

1. CA. This is an entity that issues digital certificates. It is trusted by the clients and the PCS. The issued digital certificate certifies the ownership of a public key by the named subject of the certificate. For example, the public key can be the identity in identity-based public key cryptology. In this model of trust relationships, CA is the trusted third party that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate. CA is characteristic of many public key infrastructure (PKI) schemes.
2. Client. This is an entity that has massive data to be uploaded to PCS. They maybe either individual consumers or group consumers.
3. PCS. This is an entity that is managed by the cloud service provider. PCS has significant storage space and computation resource to manage the clients' data.
4. Verifier. This is an entity that issues the remote data integrity checking challenge and gets PCS's response. Then, it verifies the response in order to verify the remote data integrity.

Next, we give a formal private Cert-RDIC scheme in public clouds. In the definition, the verifier is the client itself, *i.e.*, it belongs to the private RDIC in public clouds.

Definition 1 (Private Cert-RDIC). *A private Cert-RDIC protocol consists of seven polynomial time algorithms (*Setup*,* UserKeyGen*,* Certify*,* TagGen*,* CheckTag*,* GenProof *and* CheckProof*) which run below:*

1. $(params, X, Z, x, z) \leftarrow Setup(1^k)$ *is the parameter-generation algorithm. When the security parameter k is input, the algorithm outputs the system public parameters* params*, CA's public key* X*, PCS's public key* Z*, CA's master secret key x and PCS's secret key* z.
2. $(pk_{ID}, sk_{ID}) \leftarrow UserKeyGen(1^k, params)$ *is a probabilistic polynomial time algorithm that is run by the client ID. It outputs the client's secret key/public key pair (sk_{ID}, pk_{ID}).*
3. $Cert_{ID} \leftarrow Certify(ID, x)$ *is a probabilistic polynomial time certificate generation algorithm that is run by CA. When the client identity ID and CA's master secret key x are input, it outputs the client's certificate $Cert_{ID}$.*
4. $T_m \leftarrow TagGen(sk_{ID}, Cert_{ID}, m)$ *is a probabilistic polynomial time algorithm that is run by the client ID. When the secret key sk_{ID}, the certificate $Cert_{ID}$ and a file block m are input, it outputs the tag T_m.*
5. $\{\text{"}success\text{"}, \text{"}failure\text{"}\} \leftarrow CheckTag(z, X, pk_{ID}, Z, cert_{ID}, ID, T_m)$ *is run by PCS to check the tag T_m. When the PCS's secret key z, CA's public key X, the client's public key pk_{ID} and certificate $cert_{ID}$, PCS's public key Z, the client's identity ID, and a tag T_m are input, it returns "success" or "failure" representing that T_m is a correct tag or not.*
6. $V \leftarrow GenProof(params, ID, F, chal, \Sigma)$ *is run by PCS to generate the remote data integrity proof. When a challenge chal, an ordered collection Σ of the verification tags, a stored block collection F and $\{params, ID\}$ are input, it returns V.*
7. $\{\text{"}success\text{"}, \text{"}failure\text{"}\} \leftarrow CheckProof(sk_{ID}, Y, X, Z, ID, chal, cert_{ID}, V)$ *is run by the client to validate a proof of remote data integrity. When the parameters $\{sk_{ID}, pk_{ID}, X, Z, I, chal, cert_{ID}, V\}$ are input, it returns "success" or "failure" representing that V is a correct proof or not.*

In our security model, CA is a secure and trusted entity. On the other hand, a private Cert-RDIC protocol has to stay secure even if the prover (i.e., PCS) is malicious. A malicious prover is interested in proving knowledge of some data that she does not entirely know. The security of a private Cert-RDIC protocol is defined below.

Definition 2 (Unforgeability). *For any probabilistic polynomial time adversary $\mathcal{A}$, if the probability that $\mathcal{A}$ wins the private Cert-RDIC game is negligible, then the private Cert-RDIC protocol is called secure. Between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$, the private Cert-RDIC game is given below:*

1. *Setup: In the phase, $\mathcal{C}$ creates the system parameters $\{params, (z, Z), (x, X), (sk_{ID}, pk_{ID}), Cert_{ID}\}$. The symbols have the same meanings as in the Definition 1. $\mathcal{C}$ sends $(params, z, Z, X, pk_{ID}, Cert_{ID})$ to $\mathcal{A}$ and keeps the CA's secret key x and the client's secret key sk_{ID} confidential.*
2. *First-Phase: $\mathcal{A}$ adaptively makes Hash and Tag queries to $\mathcal{C}$ below:*

   – Hash. *Upon receiving $\mathcal{A}$'s query, $\mathcal{C}$ responds with the* hash *values .*
   – Tag. *Upon receiving $\mathcal{A}$'s query m, $\mathcal{C}$ responds with the tag $T_m \leftarrow TagGen(sk_{ID}, Cert_{ID}, m)$. Without loss of generality, denote $\mathbb{I}_1$ as the subscript set of queried block-tag pairs, i.e.,$\{(m_i, T_i) : i \in \mathbb{I}_1\}$.*
3. *Challenge: $\mathcal{C}$ generates a challenge chal which defines the challenged blocks' subscript set $\{i_1, i_2, \cdots, i_c\} \nsubseteq \mathbb{I}_1$ where c is an positive integer. $\mathcal{A}$ is required to provide a proof of remote data integrity checking for chal.*
4. *Second-Phase: It is similar to the First-Phase. Denote $\mathbb{I}_2$ as the subscript set of queried block-tag pairs, i.e., $\{(m_i, T_i) : i \in \mathbb{I}_2\}$. The restriction is that $\{i_1, i_2, \cdots, i_c\} \nsubseteq \mathbb{I}_1 \cup \mathbb{I}_2$.*
5. *Forge: $\mathcal{A}$ computes a remote data integrity checking proof V which corresponds to chal and returns V.*

We say that $\mathcal{A}$ wins the private Cert-RDIC game if

$$CheckProof(sk_{ID}, Y, X, Z, ID, chal, cert_{ID}, V) = \text{“}success\text{”}$$

In order to state clearly the status of the unchallenged blocks, we give the following definition.

Definition 3 ((ρ, δ) security). *A private Cert-RDIC protocol is (ρ, δ) secure if, given a fraction ρ of PCS-corrupted blocks, the probability that the corrupted blocks are detected is at least δ.*

3 Proposed Private Cert-RDIC Protocol

3.1 Bilinear Pairings

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two cyclic multiplicative groups with the same prime order q. Let $e : \mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$ be a bilinear map, which satisfies the following properties:

1. Bilinearity: $\forall g_1, g_2 \in \mathcal{G}_1$ and $a, b \in \mathcal{Z}_q$, $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$.
2. Non-degeneracy: $\exists g_3, g_4 \in \mathcal{G}_1$ such that $e(g_3, g_4) \neq 1_{\mathcal{G}_2}$.
3. Computability: $\forall g_5, g_6 \in \mathcal{G}_1$, there is an efficient algorithm to calculate $e(g_5, g_6)$.

Our private Cert-RDIC protocol is constructed on the Gap Diffie-Hellman group, where the computational Diffie-Hellman (CDH) problem is hard while the decisional Diffie-Hellman (DDH) problem is easy. Gap Diffie-Hellman group is defined below.

Definition 4 (Gap Diffie-Hellman (GDH) Group). *Let g be a generator of $\mathcal{G}_1$. Given $g, g^a, g^b, g^c \in \mathcal{G}_1$ and unknown $a, b, c \in \mathcal{Z}_q^*$, an efficient algorithm exists to determine whether $ab = c \bmod q$ holds by verifying $e(g^a, g^b) = e(g, g)^c$ in polynomial time (DDH problem), while no efficient algorithm exists to compute $g^{ab} \in \mathcal{G}_1$ with non-negligible probability within polynomial time (CDH problem). A group $\mathcal{G}_1$ is a (t, ϵ)-GDH group if the DDH problem can be efficiently solved, while no algorithm (t, ϵ)-breaks the CDH problem.*

3.2 Private Cert-RDIC Protocol Construction

The concrete private Cert-RDIC protocol comprises the procedures *Setup*, *UserKeyGen*, *Certify*, *TagGen*, *CheckTag*, *GenProof* and *CheckProof*. When only the client can perform the Cert-RDIC protocol, it is private Cert-RDIC protocol. Suppose that the client has stored n message blocks. Let the n blocks be $\{m_1, m_2, \cdots, m_n\}$. The private Cert-RDIC scheme can be described below.

– *Setup*: Let g be a generator of the group $\mathcal{G}_1$. CA chooses a random number $x \in \mathcal{Z}_q^*$ and gets its secret/public key pair (x, X) where $X = g^x$. PCS chooses a random number $z \in \mathcal{Z}_q^*$ and gets its secret/public key pair (z, Z) where $Z = g^z$. $\mathcal{G}_1, \mathcal{G}_2$ and e are the same as the section 3.1. Three cryptographic hash functions H, h, h_1, a pseudo-random function f and pseudo-random permutation π are given below:

$$H : \{0,1\}^* \to \mathcal{Z}_q^*, \quad h : \mathcal{Z}_q^* \to \mathcal{G}_1^*, \quad h_1 : \{0,1\}^* \to \mathcal{Z}_q^*$$

$$f : \mathcal{Z}_q^* \times \{1, 2, \cdots, n\} \to \mathcal{Z}_q^*, \;\; \pi : \mathcal{Z}_q^* \times \{1, 2, \cdots, n\} \to \{1, 2, \cdots, n\}$$

The parameters $\{\mathcal{G}_1, \mathcal{G}_2, e, q, g, X, Z, H, h, h_1, f, \pi\}$ are made public.
– *UserKeyGen*: The client picks a secret value $y \in \mathcal{Z}_q^*$ and gets his secret/public key pair (y, Y, u) where $Y = g^y$, $u \in \mathcal{G}_1$.
– *Certify*. For the client with public key Y and the identity ID, to construct ID's certificate, CA picks $r \in \mathcal{Z}_q^*$ and computes

$$R = g^r, \qquad \sigma = r + xH(ID, u, R, Y) \bmod q$$

CA sends the certificate (R, σ) to the client. The client sends the certificate (R, σ) to PCS. Note that a correctly generated certificate should satisfy the following equality: $g^{\sigma} = R \cdot X^{H(ID,u,R,Y)}$
– *TagGen*: Based on the counter i, the client creates the tags sequentially. The client computes $ck = Z^{y+\sigma}$. For the block m_i whose name, property, *etc.* are contained in N_i, the client computes $\hat{m}_i = h_1(m_i)$ and the tag $T_i = (h(ck, N_i, i)u^{\hat{m}_i})^{\sigma+y}$. The client sends the block-tag pair (m_i, T_i) to PCS.
– *CheckTag*: Given the block-tag pair set $\{(m_i, T_i), 1 \le i \le n\}$, PCS performs the procedures for every i $(1 \le i \le n)$ below:
  1. PCS computes $ck = (YRX^{H(ID,u,R,Y)})^z$ and $\hat{m}_i = h_1(m_i)$.
  2. PCS verifies whether $e(T_i, g) \overset{?}{=} e(h(ck, N_i, i)u^{\hat{m}_i}, YRX^{H(ID,u,R,Y)})$ holds. If it holds, then PCS accepts it; otherwise, PCS rejects it and queries the client for new block-tag pair.
– *GenProof*: Upon receiving the challenge $chal = (c, k_1, k_2)$, PCS performs the procedures below:
  1. For $1 \le j \le c$, $i_j = \pi_{k_1}(j), a_j = f_{k_2}(j)$ are computed as the subscripts and coefficients of the blocks for which the proof is generated. In fact, the challenge *chal* determines an ordered set $\{c, i_1, \cdots, i_c, a_1, \cdots, a_c\}$.
  2. Compute $T = \prod_{j=1}^{c} T_{i_j}^{a_j}$, $\hat{m} = \sum_{j=1}^{c} a_j h_1(m_{i_j})$.

3. Output $V = (T, \hat{m})$ as the response to the *chal* query.

– *CheckProof*: The client verifies the response V from the PCS below:
  1. Compute $ck = Z^{y+\sigma}$, $i_j = \pi_{k_1}(j)$ and $a_j = f_{k_2}(j)$.
  2. Check the equation $e(T, g) \overset{?}{=} e(\prod_{j=1}^{c} h(ck, N_{i_j}, i_j)^{a_j} u^{\hat{m}},\ YRX^{H(ID,u,R,Y)})$. If it holds, output "success". Otherwise, output "failure".

A private Cert-RDIC protocol must be workable and correct. That is, if the CA, the Client and the PCS are honest and follow the specified procedures, the response V can pass the *CheckProof* phase. The correctness of our private Cert-RDIC protocol is given below:

$$\begin{aligned} e(T, g) &= e(\textstyle\prod_{j=1}^{c} T_{i_j}^{a_j}, g) = e(\prod_{j=1}^{c}(h(ck, N_{i_j}, i_j)u^{\hat{m}_{i_j}})^{(y+\sigma)a_j}, g) \\ &= e(\textstyle\prod_{j=1}^{c}(h(ck, N_{i_j}, i_j)u^{\hat{m}_{i_j}})^{a_j}, g^{y+\sigma}) \\ &= e(\textstyle\prod_{j=1}^{c} h(ck, N_{i_j}, i_j)^{a_j} u^{\hat{m}}, YRX^{H(ID,u,R,Y)}) \end{aligned}$$

4 Security Analysis

The security of the above private Cert-RDIC protocol is guaranteed by the following lemmas and theorems.

Lemma 1. *Suppose $\mathcal{A}$ is a (t, ϵ, q_h, q_t)-forger against our TagGen protocol, i.e., $\mathcal{A}$ can forge a valid tag with probability ϵ within time t. The groups $\mathcal{G}_1$ and $\mathcal{G}_2$ have the prime order q. Then, the CDH problem can be solved with probability ϵ' within time t' satisfying $\epsilon' \geq \frac{1}{9}$ and $t' \leq \frac{23q_h(t+(q_t+q_h)t_{exp}+q_h t_{mul})(q_t+1)}{\epsilon}\left(1+\frac{1}{q_t}\right)^{q_t}$ where t_{exp} denotes the time needed to perform a modular exponentiation in $\mathcal{G}_1$, t_{mul} denotes the time needed to perform a multiplication in $\mathcal{G}_1$, and q_h, q_t denote the number of queries to the h-Oracle and TagGen-Oracle, respectively.*

Proof. Let the uploaded subscript-block pairs be $\{(1, m_1), (2, m_2), \cdots, (n, m_n)\}$. Taking the adversary $\mathcal{A}$ as a sub-routine, we construct a probabilistic polynomial time algorithm $\mathcal{F}$. If $\mathcal{A}$ can succeed to forge a valid tag, $\mathcal{F}$ can solve a given instance of the CDH problem. Thus, $\mathcal{F}$ simulates the environment of $\mathcal{A}$ below.

The parameters $(q, \mathcal{G}_1, \mathcal{G}_2, e, g, g^a, g^b)$ are made public. The goal of $\mathcal{F}$ is to compute the value g^{ab}. Let (σ, R) be the client's certificate and (x, X) be the CA's secret/public key pair. First, $\mathcal{F}$ picks a random $u \in \mathcal{G}_1$ and defines $(Y = g^a, u)$ as the client's public key and maintains four tables Tab_1, Tab_2, Tab_3 and Tab_4, which are initialized empty. The client's secret key a is unknown to $\mathcal{F}$.

Hash functions H and h_1. In the simulation, H and h_1 are the real hash function and are not oracles. When $\mathcal{A}$ queries H with (ID, u, R, Y), $\mathcal{F}$ responds $H(ID, u, R, Y)$. When $\mathcal{A}$ queries h_1 with m_i, $\mathcal{F}$ responds $\hat{m}_i = h_1(m_i)$.

h-Oracle. When $\mathcal{A}$ queries h with (ck, N_i, i), $\mathcal{F}$ responds below:

1. If the query (ck, N_i, i) has been stored in the h-list Tab_1, *i.e.*, $(ck, N_i, i, z_i, b_i, d_i) \in \text{Tab}_1$ and $1 \leq i \leq n$, $\mathcal{F}$ responds with $h(ck, N_i, i) = z_i u^{-h_1(m_i)}$.
2. Otherwise, based on the bivariate distribution $\Pr[d_i = 0] = \frac{1}{q_h+1}$, $\Pr[d_i = 1] = 1 - \frac{1}{q_h+1}$, $\mathcal{F}$ generates a random coin $d_i \in \{0, 1\}$.

(a) $\mathcal{F}$ picks a random $b_i \in \mathcal{Z}_q^*$. If $d_i = 1$, $\mathcal{F}$ computes $z_i = g^{b_i}$. If $d_i = 0$, $\mathcal{F}$ computes $z_i = (g^b)^{b_i}$.
(b) $\mathcal{F}$ adds the tuple $(ck, N_i, i, z_i, b_i, d_i)$ to the h-list Tab$_1$ and responds $h(cn, N_i, i) = z_i u^{-h_1(m_i)}$.

TagGen-Oracle. $\mathcal{A}$ queries *TagGen-Oracle* with (ck, N_i, i, m_i). $\mathcal{F}$ responds $\mathcal{A}$ below:

1. $\mathcal{F}$ runs h-Oracle and obtains $h(cn, N_i, i)$. Let $(ck, N_i, i, z_i, b_i, d_i)$ be the corresponding tuple in table h-list Tab$_1$.
2. If $d_i = 0$, $\mathcal{F}$ reports failure and terminates. Otherwise, $d_i = 1$, define $\sigma_i = (g^a g^\sigma)^{b_i}$. Observe that $T_i = (g^a g^\sigma)^{b_i} = (g^{b_i})^{a+\sigma} = (h(ck, N_i, i)u^{h_1(m_i)})^{a+\sigma}$ under the public parameters (g^a, u). Then, $\mathcal{F}$ sends T_i to $\mathcal{A}$.

Forge and Output. Finally, $\mathcal{A}$ forges a valid block-tag pair (m_j, T_j) where (ck, N_j, j, m_j) has never been queried to *TagGen-Oracle*. If there is no tuple in the h-list Tab$_1$ containing (ck, N_j, j), $\mathcal{F}$ queries *h-Oracle* to ensure that such a tuple exists. Next, $\mathcal{F}$ searches for the tuple $(ck, N_j, j, z_j, b_j, d_j)$ in Tab$_1$. If $d_j = 1$ then $\mathcal{F}$ reports failure and terminates. Otherwise $(d_j = 0)$, (ck, N_j, j, m_j, T_j) satisfies $e(T_j, g) = e(h(ck, N_j, j)u^{h_1(m_j)}, Yg^\sigma)$.

By using the oracle-replay technique [28], $\mathcal{F}$ can obtain another tag $(m_j, \hat{T}_j)$ by using a different hash function $\hat{h}$. Then, $\mathcal{F}$ can get

$$e(\hat{T}_j, g) = e(h(ck, N_j, j)u^{h_1(m_j)}, Rg^\sigma)$$

Based on the simulation process of *h-Oracle*, $\mathcal{F}$ knows the corresponding $b_j, \hat{b}_j$ that satisfy $h(ck, N_j, j) = (g^b)^{b_j} u^{-h_1(m_j)}$, $\hat{h}(ck, N_j, j) = (g^b)^{\hat{b}_j} u^{-h_1(m_j)}$. Thus, $\mathcal{F}$ can get $e(T_j, g) = e((g^b)^{b_j}, Y \cdot g^\sigma)$, $e(\hat{T}_j, g) = e((g^b)^{\hat{b}_j}, Y \cdot g^\sigma)$. From the above procedure, $\mathcal{F}$ gets $e(T_j \hat{T}_j^{-1}, g) = e((g^b)^{b_j - \hat{b}_j}, Y \cdot g^\sigma)$ and obtains

$$g^{ab} = \left[\left(T_j \hat{T}_j^{-1}\right)\left(g^b\right)^{-\left(b_j - \hat{b}_j\right)\sigma}\right]^{\frac{1}{b_j - \hat{b}_j}}$$

Probability analysis. The above proof comes from the $\mathcal{F}$'s perfect simulation. Based on the simulation process, the probability that $\mathcal{F}$ succeeds to answer $\mathcal{A}$'s queries is $P_1 = (1 - \frac{1}{q_t+1})^{q_t}$. Thus, the attacker $\mathcal{A}$ can forge a valid tag with probability $\hat{\epsilon} \geq \frac{1}{q_t+1} P_1 \epsilon = \frac{1}{q_t+1}(1 - \frac{1}{q_t+1})^{q_t} \epsilon$ within the time $\hat{t} \approx t + (q_t + q_h)t_{exp} + q_h t_{mul}$. By using the oracle replay technique [28], $\mathcal{A}$ can get two different tags on the same message and randomness with the probability $\epsilon' \geq \frac{1}{9}$ within time $t' \leq 23 q_h \hat{t} \hat{\epsilon}^{-1}$, *i.e.*, $t' \leq \frac{23 q_h (t + (q_t + q_h)t_{exp} + q_h t_{mul})(q_t + 1)}{\epsilon} \left(1 + \frac{1}{q_t}\right)^{q_t}$.

Based on the known assumption that the CDH problem is computationally difficult, the following corollary is implied by the above Lemma 1.

Corollary 1 (Single-tag existential unforgeability). *A single tag is existentially unforgeable if the CDH problem in $\mathcal{G}_1$ is hard.*

Lemma 2. *Based on the unforgeability of a single tag (Corollary 1), the grouped message-tag pair* $(\hat{m}, T)$ *can be forged only with negligible probability.*

Proof. We will prove this lemma by contradiction. Suppose the forged message-tag pair $(\hat{m}, T)$ can pass the client's integrity checking, *i.e.*,

$$e(T, g) = e(\prod_{j=1}^{c} h(ck, N_{i_j}, i_j)^{a_j} u^{\hat{m}}, YRX^{H(ID,u,R,Y)})$$

where $a_j = f_{k_2}(j)$ are random numbers, $1 \le j \le c$. Then,

$$\prod_{j=1}^{c} e(T_{i_j}, g)^{a_j} = \prod_{j=1}^{c} e(h(ck, N_{i_j}, i_j) u^{h_1(m_{i_j})}, YRX^{H(ID,u,R,Y)})^{a_j}$$

Suppose that the generator of $\mathcal{G}_2$ is d, and

$$e(T_{i_j}, g) = d^{x_j},\ e(h(ck, N_{i_j}, i_j) u^{h_1(m_{i_j})},\ YRX^{H(ID,u,R,Y)}) = d^{y_j}$$

then we can get

$$d^{\sum_{j=1}^{c} a_j x_j} = d^{\sum_{j=1}^{c} a_j y_j},\ \sum_{j=1}^{c} a_j x_j = \sum_{j=1}^{c} a_j y_j,\ \sum_{j=1}^{c} a_j (x_j - y_j) = 0$$

According to Corollary 1, a single tag is existentially unforgeable. So, there exist at least two different indices j such that $x_j \neq y_j$. Suppose there are $s \le c$ pairs (x_j, y_j) such that $x_j \neq y_j$. Then, there exist q^{s-1} tuples $(a_1, a_2, \cdots, a_c)$ satisfying the above equation. As $(a_1, a_2, \cdots, a_c)$ is a random vector, the probability that the tuple satisfies the above equation is not greater than $q^{s-1}/q^c \le q^{c-1}/q^c = q^{-1}$. This probability is negligible.

Lemma 3. *Assume some block-tag pair* (m_l, T_l) *is modified and PCS substitutes another valid block-tag pair* $(m_{\hat{l}}, T_{\hat{l}})$ *for* (m_l, T_l)*. If PCS forges a response* $(T, \hat{m})$ *using the substituted block-tag, this response passes CheckProof only with negligible probability.*

Proof. Without loss of generality, we assume that the client sends the challenge $chal = (c, k_1, k_2)$, where $l \in \{1, 2, \cdots, c\}$, $\hat{l} \notin \{1, 2, \cdots, c\}$.

Since the block-tag pair (m_l, T_l) is modified, the public cloud server may substitute another valid block-tag pair $(m_{\hat{l}}, T_{\hat{l}})$ for (m_l, T_l). Then,

$$T = \prod_{j=1, j\neq l}^{c} T_{i_j}^{a_j} T_{m_{i_{\hat{l}}}}^{a_l},\quad \hat{m} = \sum_{j=1, j\neq l}^{c} a_j m_{i_j} + a_l m_{i_{\hat{l}}}$$

where $a_i = f_{k_2}(i)$ is random number, $1 \le i \le c$.

If the forged response passed the checking, then

$$e(T, g) = e(\prod_{j=1}^{c} h(ck, N_{i_j}, i_j)^{a_j} u^{\hat{m}}, YRX^{H(ID,u,R,Y)})$$

i.e.,

$$\begin{aligned} & e(\prod_{j=1, j\neq l}^{c} T_{i_j}^{a_j} T_{m_{i_{\hat{l}}}}^{a_l}, g) \\ = & e(\prod_{j=1, j\neq l}^{c} [h(ck, N_{i_j}, i_j)^{a_j}] u^{\sum_{j=1, j\neq l}^{c} a_j h_1(m_{i_j}) + a_l h_1(m_{i_{\hat{l}}})}, YRX^{H(ID,u,R,Y)}) \end{aligned}$$

Since the other block-tag pair is valid, we can get

$$e(T_{m_{i_{\hat{l}}}}^{a_l}, g) = e(h(ck, N_{i_l}, i_l)^{a_l} u^{a_l h_1(m_{i_{\hat{l}}})}, YRX^{H(ID,u,R,Y)})$$

According to the tag generation process, we know that

$$\begin{aligned} &e(h(ck, N_{i_{\hat{l}}}, i_{\hat{l}})^{a_l} u^{a_l h_1(m_{i_{\hat{l}}})}, YRX^{H(ID,u,R,Y)}) \\ = &e(h(ck, N_{i_l}, i_l)^{a_l} u^{a_l h_1(m_{i_{\hat{l}}})}, YRX^{H(ID,u,R,Y)}) \end{aligned}$$

i.e., $h(ck, N_{i_{\hat{l}}}, i_{\hat{l}}) = h(ck, N_{i_l}, i_l)$.

Since h is collision-free hash function, the probability of $h(ck, N_{i_{\hat{l}}}, i_{\hat{l}}) = h(ck, N_{i_l}, i_l)$ is $\frac{1}{q}$, *i.e.*, it is negligible.

Corollary 1 states that an untrusted PCS cannot forge individual tags to cheat the client. Lemma 2 implies that the untrusted PCS cannot aggregate fake tags to cheat the client. Lemma 3 shows that the untrusted PCS cannot replace some tags to cheat the client. Hence, from the above results, we have the following claim.

Theorem 1. *The proposed private Cert-RDIC protocol is unforgeable in the random oracle model if the CDH problem in $\mathcal{G}_1$ is hard.*

Theorem 2. *Suppose that the client has stored n block-tag pairs $((m_1, T_1), (m_2, T_2), \cdots, (m_n, T_n))$ in PCS and the PCS has modified t block-tag pairs. Let the challenge be $chal = (c, k_1, k_2)$, the probability P_X of detecting the modification satisfies:*

$$1 - (\frac{n-t}{n})^c \leq P_X \leq 1 - (\frac{n-c+1-t}{n-c+1})^c$$

We omit the detailed proof since it is similar to the proof of deletion detection [1].

Theorem 3. *Our proposed Cert-RDIC protocol is private Cert-RDIC protocol. When the client is willing to publish some parameters, it can be transformed into public Cert-RDIC protocol.*

Proof. In the phase *CheckProof*, the parameter $ck = Z^{y+\sigma}$ cannot be computed by other entities except of the client and PCS. Thus, other entities cannot perform the verification formula $e(T, g) = e(\prod_{j=1}^{c} h(ck, N_{i_j}, i_j)^{a_j} u^{\hat{m}}, YRX^{H(ID,u,R,Y)})$. Thus, except of the client and the PCS, the other entities cannot the Cert-RDIC protocol. Our proposed Cert-RDIC protocol is private Cert-RDIC protocol.

When the client decides to publish the parameter $ck = Z^{y+\sigma}$, every entity can perform the Cert-RDIC protocol. Thus, it is transformed into public Cert-RDIC protocol.

On the other hand, in the phase *CheckTag*, PCS can check every received block-tag pair. When some pairs are invalid, PCS will reject them. Thus, PCS can avoid of the malicious clients and protect his own benefits. In order to increase the verification efficiency, PCS can take use of the batch processing technique and verify many block-tag pairs at the same time.

5 Conclusion

Cert-RDIC combines the advantage of both PKC-RDIC and ID-RDIC as it avoids the usage of certificates and does not suffer from the key escrow. Based on these advantages, this paper formalizes the private Cert-RDIC model which comprises system model and security model. Then, we present the first concrete private Cert-RDIC protocol which is provably secure under the assumption that the CDH problem is hard. When the client publishes some parameters, our private Cert-RDIC protocol can be transformed into public Cert-RDIC protocol.

Acknowledgments. This work was partly supported by the Natural Science Foundation of China (No.61272522, 61272542), by the Natural Science Foundation of Liaoning Province (No. 2014020147), by the Program for Liaoning Excellent Talents in University (No. LR2014021), by the Research Project of Liaoning Provincial Education Department (No. L2012259) and by the Fundamental Research Funds for the Central Universities(No. 2013B07014).

References

1. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: CCS 2007, pp. 598–609 (2007)
2. Ateniese, G., DiPietro, R., Mancini, L.V., Tsudik, G.: Scalable and Efficient Provable Data Possession. http://eprint.iacr.org/2008/114
3. Erway, C.C., Kupcu, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: CCS 2009, pp. 213–222 (2009)
4. Sebé, F., Domingo-Ferrer, J., Martínez-Ballesté, A., Deswarte, Y., Quisquater, J.: Efficient Remote Data Integrity checking in Critical Information Infrastructures. IEEE Transactions on Knowledge and Data Engineering **20**(8), 1034–1038 (2008)
5. Wang, H.: Identity-Based Distributed Provable Data Possession in Multi-Cloud Storage. IEEE Transactions on Services Computing **8**(2), 328–340 (2015)
6. Wang, H., Zhang, Y.: On the Knowledge Soundness of a Cooperative Provable Data Possession Scheme in Multicloud Storage. IEEE Transactions on Parallel and Distributed Systems **25**(1), 264–267 (2014)
7. Wang, H., Wu, Q., Qin, B., Josep, D.: FRR: Fair Remote Retrieval of Outsourced Private Medical Records in Electronic Health Networks. Journal of Biomedical Informatics **50**, 226–233 (2014)
8. Wang, H.: Anonymous Multi-Receiver Remote Data Retrieval for Pay-TV in Public Clouds. IET Information Security **9**(2), 108–118 (2015)
9. Wang, H.: Proxy Provable Data Possession in Public Clouds. IEEE Transactions on Services Computing **6**(4), 551–559 (2013)
10. Hao, Z., Yu, N.: A multiple-replica remote data possession checking protocol with public verifiability. In: 2010 Second International Symposium on Data, Privacy, and E-Commerce, pp. 84–89 (2010)
11. Barsoum, A.F., Hasan, M.A.: On Verifying Dynamic Multiple Data Copies over Cloud Servers. IACR eprint report 447 (2011). http://eprint.iacr.org/2011/447.pdf
12. Wang, H., Zhang, Y.: On the Knowledge Soundness of a Cooperative Provable Data Possession Scheme in Multicloud Storage. IEEE Transactions on Parallel and Distributed Systems **25**(1), 264–267 (2014)

13. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing. IEEE Transactions on Parallel and Distributed Systems **22**(5), 847–859 (2011)
14. Juels, A., Kaliski, B.S.: PORs: proofs of retrievability for large files. In: CCS 2007, pp. 584–597 (2007)
15. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
16. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: CCSW 2009, pp. 43–54 (2009)
17. Zheng, Q., Xu, S.: Fair and dynamic proofs of retrievability. In: CODASPY 2011, pp. 237–248 (2011)
18. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 109–127. Springer, Heidelberg (2009)
19. Zhu, Y., Wang, H., Hu, Z., Ahn, G.J., Hu, H.: Zero-Knowledge Proofs of Retrievability. Sci. China Inf. Sci. **54**(8), 1608–1617 (2011)
20. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
21. Han, J., Susilo, W., Mu, Y.: Identity-Based Data Storage in Cloud Computing. Future Generation Computer Systems **29**(3), 673–681 (2013)
22. Han, J., Susilo, W., Mu, Y.: Identity-Based Secure DistributedData Storage Schemes. IEEE Trans. Computers **63**(4), 941–953 (2014)
23. Wang, H., Wu, Q., Qin, B., Domingo-Ferrer, J.: Identity-based remote data possession checking in public clouds. IET Information Security **8**(2), 114–121 (2014)
24. Gentry, C.: Certificate-based encryption and the certificate revocation problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 272–293. Springer, Heidelberg (2003)
25. Kang, B.G., Park, J.H., Hahn, S.G.: A certificate-based signature scheme. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 99–111. Springer, Heidelberg (2004)
26. Li, J., Huang, X., Mu, Y., Susilo, W., Wu, Q.: Certificate-based signature: security model and efficient construction. In: López, J., Samarati, P., Ferrer, J.L. (eds.) EuroPKI 2007. LNCS, vol. 4582, pp. 110–125. Springer, Heidelberg (2007)
27. Li, J., Huang, X., Mu, Y., Susilo, W., Wu, Q.: Constructions of Certificate-Based Signature Secure Against Key Replacement Attacks. Journal of Computer Security **18**(3), 421–449 (2010)
28. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology **13**(3), 361–396 (2000)

Maximal and Maximum Transitive Relation Contained in a Given Binary Relation

Sourav Chakraborty[1], Shamik Ghosh[2], Nitesh Jha[1](✉), and Sasanka Roy[1]

[1] Chennai Mathematical Institute, Chennai, India
{sourav,nj,sasanka}@cmi.ac.in
[2] Department of Mathematics, Jadavpur University, Kolkata, India
sghosh@math.jdvu.ac.in

Abstract. We study the problem of finding a *maximal* transitive relation contained in a given binary relation. Given a binary relation of size m defined on a set of size n, we present a polynomial time algorithm that finds a maximal transitive sub-relation in time $O(n^2 + nm)$.

We also study the problem of finding a *maximum* transitive relation contained in a binary relation. For the class of triangle-free relations (directed graphs), we present a 0.874-approximation via the problem of maximum directed cut.

1 Introduction

All relations considered in this study are binary relations. We represent a relation alternately as a digraph to simplify the presentation at places (see Section 2 for definitions). Transitivity is a fundamental property of relations. Given the importance of relations and the transitivity property, it is not surprising that various related problems have been studied in detail and have found widespread application in different fields of study.

Some of the fundamental problems related to transitivity that have been long studied are - given a relation ρ, checking whether ρ is transitive, finding the transitive closure of ρ, finding the maximum transitive relation contained in ρ, partitioning ρ into smallest number of transitive relations. Various algorithms have been proposed for these problems and some hardness results have also been proved.

In this paper, we study two related problems on transitivity. First - given a relation, obtain a *maximal* transitive relation contained in it. It is straightforward to see that this can be solved in poly-time, hence our goal is to do this as efficiently as possible. Second - given a relation, obtain a *maximum* transitive relation contained it. This problem was proven to be NP-complete in [11]. Here our approach is to find approximate solutions.

The problem of finding a *maximum* transitive relation contained in a given relation is a generalisation of well-studied hard problems. For the class of triangle-free graphs, the problem of finding a maximum transitive subgraph in a directed graph is the same problem as the MAX-DICUT problem (see Section 4). MAX-DICUT has well known inapproximability results.

DOI: 10.1007/978-3-319-21398-9_46

We can also relate it to a problem of optimisation on a 3SAT instance. We look at the relation as a directed graph $G = (V, E)$, where $|V| = n$. For every pair for distinct vertices (i, j) in V, create a boolean variable x_{ij}. Consider the following 3SAT formula.

$$C = \bigwedge_{1 \leqslant i < j < k \leqslant n} (x_{ij} \vee \overline{x}_{ik} \vee \overline{x}_{kj})$$

Let C' be a formula derived from C such that any literal with variable x_{ij} is removed if $(i, j) \notin E$. It is easy to see that a solution to C' represents a subgraph of G. Specifically, a solution to C' is also transitive. To see this, observe that for every triplet (i, j, k), if a clause $(x_{ij} \vee \overline{x}_{ik} \vee \overline{x}_{kj})$ is satisfied, then either the edge (i, j) is included or at least one of the edges (i, k) or (k, j) is excluded. To get the maximum transitive subgraph, the solution must maximize the number of variables set to 1. To conclude, the maximum transitive subgraph problem is same as the problem of finding a satisfying solution to a 3SAT formula that also maximizes the number of variables assigned the value 'true'.

1.1 Our Results

The usual greedy algorithm for finding a maximal substructure - satisfying a given property $\mathcal{P}$ - starts with the empty set and incrementally grows the substructure while maintaining the property $\mathcal{P}$. Finally it ends when the set becomes maximal. Thus checking for maximality is a subroutine for the usual greedy algorithm.

In the case of finding a maximal transitive relation contained in a given relation the usual greedy algorithm takes $O(n^5)$ time, where n is the size of the set on which the binary relation is defined. Using matrix multiplication as a subroutine for checking maximality one can improve the running time of the greedy algorithm to $O(n^{\omega+2})$, where the matrix-multiplication of two $n \times n$ matrices takes $O(n^{\omega})$ time.

The other greedy approach for finding a maximal substructure could be to start with an object $\mathcal{O}$ and slowly shrink the object until it satisfies the property $\mathcal{P}$. Unfortunately, this technique may not yield a maximal substructure - the maximality may not be satisfied at the end.

In this paper we design an algorithm, for finding a maximal transitive subrelation in a given relation ρ. Our algorithm runs in time $O(n^3)$ where n is the size of set on which the relation ρ is defined. Our algorithm does not use any subroutine for checking for maximality. In fact the best known algorithm for checking maximality in this case has running time $O(n^{\omega+1})$ which is clearly more than the running time of our algorithm.

Instead our algorithm follows the approach of the second kind of greedy algorithms discussed above. Given the fact that usually this approach does not guarantee that the output is maximal, we have to make some clever modification. The algorithm as such is simple but the key is the proof of correctness, which is quite involved.

In fact we present an algorithm that runs in time $O(nm+n^2)$ where, n is the size of the set on which the relation is defined and m is the size of the relation ρ. To the best of our knowledge, no better algorithm for finding a maximal transitive sub-relation is known. We present the algorithms and the proof of correctness related to the following theorem in Section 3.

Theorem 1. *Let ρ be a binary relation on a set of size n and let m be the size of ρ. There is an algorithm that given ρ, outputs a maximal transitive relation contained in ρ, in time $O(n^2 + nm)$.*

In the case of finding a *maximum* transitive relation contained in a binary relation, we present the following results.

Theorem 2. *There exists a 0.25-approximation algorithm for obtaining a maximum transitive relation contained in a binary relation.*

Theorem 3. *There exists a 0.874-approximation algorithm for finding the maximum transitive relation contained in a triangle-free relation (triangle-free digraph).*

These results have been arrived at by connecting the the problem of maximum transitive 'sub-relations' to the well known problems of MAX-CUT and MAX-DICUT. This has been detailed in Section 4.

1.2 Related Results

The transitive property is a fundamental property of binary relations. Various important algorithmic problems with respect to transitive property has been studied and used. One very important and well studied problem is finding the transitive closure of a binary relation ρ (that is the smallest binary relation which contains ρ and is transitive). This problem of finding transitive closure has been studied way back in 1960s. Warshall [9] gave an algorithm to find the transitive closure is time $O(n^3)$, where n is the size of the set on which the binary relation is defined. Using different techniques [5] gave an $O(n^2 + nm)$ algorithms, where m is the number of relations in ρ. Modifying the algorithm of Warshall, Nuutila [8] connected the problem of finding transitive closure with matrix multiplication. With the latest knowledge of matrix multiplication ([4] and [10]) we can compute the transitive closure of a binary relation on n elements using $O(n^{2.37})$ time complexity.

Another important problem connected to transitive property is the finding the transitive reduction of a binary relation. Transitive reduction of a binary relation ρ is the minimal sub-relation whose transitive closure is same as the transitive closure of ρ. This was introduce by Aho et al [1] and they also gave the tight complexity bounds. A closely related concept to the transitive reduction is the maximal equivalent graph, introduced by Moyles [7].

Given a binary relation, partitioning it as a union of transitive relations is another very important related problem (see [2]). A plethora of work has been done on this problem in recent times as this problem has found application in biomedical studies.

2 Notations

Let $S = \{1, 2, \ldots, n\}$, where n is a natural number. A binary relation ρ on S is a subset of the cross product $S \times S$. We only consider binary relations in this study. Any relation ρ on S can be represented by a $(0, 1)$ matrix $A = (a_{ij})_{n\times n}$ of size $n \times n$, where

$$a_{ij} = \begin{cases} 1, \text{ if } (i, j) \in \rho \\ 0, \text{ otherwise.} \end{cases}$$

Similarly, a relation ρ on S can represented by a directed graph with S as the vertex set and elements of ρ as the arcs of the directed graph.

In this paper we do not distinguish between a relation and its matrix representation or its directed graph representation. So for a given relation ρ, if $(i, j) \in \rho$, we sometimes refer to it as the arc (i, j) being present and sometimes as the adjacency matrix entry $\rho_{ij} = 1$.

If ρ is a binary relation on S then the size of ρ (denoted by m) is the number of arcs in the directed graph corresponding to ρ. In other words, it is the number of pairs $(i, j) \in S \times S$ such that $(i, j) \in \rho$.

If ρ is a binary relation on S we say ρ' is contained in ρ (or is a sub-relation) if for all $i, j \in S$, $(i, j) \in \rho'$ implies $(i, j) \in \rho$.

Definition 1. *A binary relation ρ on S is called* transitive *if for all $a, b, c \in S$, $(a, b) \in \rho, (b, c) \in \rho$ implies $(a, c) \in \rho$.*

For a binary relation ρ on S a sub-relation α is said to be a *maximal transitive* relation contained in ρ if there does not exist any transitive relation β such that α is strictly contained in β and β is contained in ρ. A *maximum transitive* relation contained in ρ is a largest relation contained in ρ.

3 Maximal Transitive Relation Finding Algorithms

We first present an algorithm which finds a maximal transitive relation contained in a given binary relation in $O(n^3)$ and then we improve it to obtain another algorithm for this with time complexity $O(n^2 + mn)$.

3.1 $O(n^3)$ Algorithm for Finding Maximal Transitive Sub-relation

Theorem 4. ***Algorithm 1*** *correctly finds a maximal transitive sub-relation in a given relation in time $O(n^3)$.*

Proof. It is easy to see that the time complexity of the algorithm is $O(n^3)$. For the proof of correctness, all we need to prove is that the output T of the algorithm is transitive and maximal. The transitivity of the output T is proved in Lemma 2 and the maximality of T is proved in Lemma 3.

Algorithm 1. Finding a maximal transitive sub-relation

```
Input  : An $n \times n$ matrix $A = (a_{ij})$ representing a relation.
Output: A matrix $T = (t_{ij})$ which is a maximal transitive sub-relation
        contained in $A$.
1  for $i \leftarrow 1$ to $n$ do
2      for $j \leftarrow 1$ to $n$, $j \neq i$ do
3          if $a_{ij} = 1$ then
4              for $k = 1$ to $n$ do
5                  if $k \neq j$ and $a_{ik} = 0$ then
6                      set $a_{jk} = 0$
7                  end
8                  if $k \neq i$ and $a_{kj} = 0$ then
9                      set $a_{ki} = 0$
10                 end
11             end
12         end
13     end
14 end
15 return A
```

3.2 Proof of Correctness of Algorithm 1

Before we prove the correctness of **Algorithm 1**, let us make some simple observations about the algorithm. In this section we will treat the binary relation on a set S as a directed graph with vertex set S. So the **Algorithm 1** takes a directed graph A on n vertices (labelled 1 to n) and outputs a directed transitive subgraph T that is maximal, that is, one cannot add arcs from G to T to obtain a bigger transitive graph. In the algorithm, note that changing an entry a_{ij} from 1 to 0 implies deletion of the arc (i, j).

Definition 2. *At any stage of the **Algorithm 1** we say the arc (a, b) is visited if at some earlier stage of the algorithm when $i = a$ in Line 1 and $j = b$ in Line 2 we had $a_{ij} = 1$.*

Remark 1. We first note the following obvious but important facts of the **Algorithm 1**:

(1) No new arc is created during the algorithm because it never changes an entry a_{ij} in the matrix A from 0 to 1. It only deletes arcs.
(2) Line 1, 2 and 3 of the algorithm implies that the algorithm visits the arcs one by one (in a particular order). And while visiting an arc it decides whether or not to delete some arcs.
(3) Since in Line 1 the i increases from 1 to n so the algorithm first visits the arcs starting from vertex 1 and then the arcs starting from vertex 2 and then the arcs starting from vertex 3 and so on.

(4) Arcs are deleted only in Line 6 and Line 9 in the algorithm.
(5) While the for loop in Line 1 is in the i-th iteration (that is when the algorithm is visiting an arc starting at i) no arc starting from the i is deleted. In Line 6 only arcs starting from j are deleted and $j \neq i$ from Line 2. And in Line 9 only arcs ending in i are deleted.
(6) In Line 2 the condition $j \neq i$ is given just for ease of understanding the algorithm. As such even if the condition was not there the algorithm would have the same output because if $j = i$ in Line 2 and the algorithm pass line 3 (that is $a_{ii} = 1$) then Line 6 would read as "if $a_{ik} = 0$ write $a_{ik} = 0$" and Line 9 would read as "if $a_{ki} = 0$ write $a_{ki} = 0$", both of which are no action statement.
(7) Similarly, in Line 5 the condition $k = j$ is given just for ease of understanding of the algorithm. If the condition was not there even then the algorithm would have produced the same result because from Line 3 we already have $a_{ij} = 1$ and thus if $k = j$ then $a_{ik} = a_{ij} \neq 0$.
(8) Similarly, the condition $k \neq i$ in Line 9 has no particular role in the algorithm.

One of the most important lemma for the proof of correctness is the following:

Lemma 1. *An arc once visited in* ***Algorithm 1*** *cannot be deleted later on.*

Proof. Let us prove by contradiction.

Suppose at a certain point in the algorithm's run the arc (i, j) has already been visited, and then when the algorithm is visiting some other arc starting from vertex r the algorithm decides to delete the arc (i, j).

If such an arc (i, j) which is deleted after being visited exists then there must a first one also. Without loss of generality we can assume that the arc (i, j) is the first such arc: that is when the algorithm decides to delete the arc (i, j) no other arc that has been visited by the algorithm has been deleted.

By point number 3 in Remark 1, $r \geq i$. From point number 5 in Remark 1 we can say that $r \neq i$. So we have $r > i$.

We now consider two cases depending on whether the algorithm decides to delete the arc (i, j) is Line 6 or Line 9.

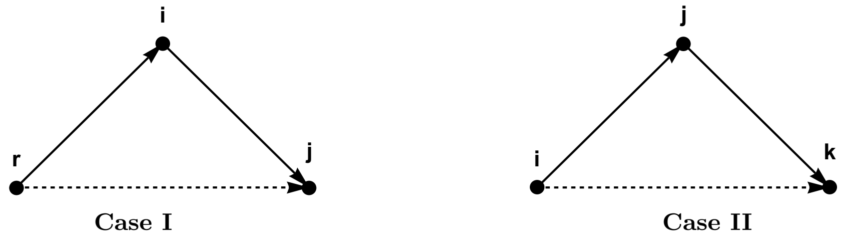


Fig. 1. Diagrams of the two cases for Lemma 1

Case I. Suppose (i, j) is deleted in Line 6, when the algorithm was visiting an arc starting from vertex r. Since the algorithm is deleting (i, j) in Line 6 so from Line 3 and Line 5 we have, at that stage, $a_{ri} = 1$ and $a_{rj} = 0$ (just like in Figure 1(left)).

Since no arc is ever created by the algorithm (point 1 in Remark 1), a_{ri} was 1 when the arc (i, j) was visited. So at the stage when the algorithm was visiting arc (i, j), a_{rj} must be 1, otherwise (r, i) would be deleted by Line 9. Thus (r, j) was deleted after visiting the arc (i, j) and but before time (i, j) is being deleted.

By Remark 1(5), (r, j) cannot be deleted when visiting an arc starting from r. So (r, j) must have been deleted when visiting an arc starting from vertex r_1 and $r_1 < r$.

We now split this case into two cases depending on whether $r_1 = j$ or $r_1 \neq j$.

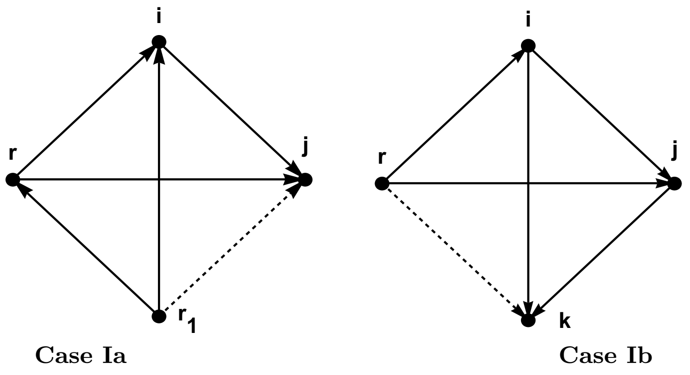


Fig. 2. Diagram for subcases of Case 1 for Lemma 1

Case Ia: $(r_1 \neq j)$

By Remark 1(5) we know at the set of arcs starting from vertex r_1 must have remained unchanged during the r_1-th iteration of Line 1.

But since in the r_1-th iteration of Line 1 the arc (r, j) was deleted so (r_1, r) must have been present while (r_1, j) was absent. Also if $a_{r_1 i} = 0$ when visiting the arc (r_1, r), the algorithm would have found $a_{r_1 r} = 1$ and $a_{r_1 i} = 0$ and in that case would have deleted (r, i) is Line 6. That would contradict that fact the the arc (r, i) was present when the arc (i, j) was being deleted. Thus at the start of the r_1-th iteration of Line 1 the situation would have been like in Figure 2(left)).

But in that case, when visiting (r_1, i) the algorithm would have found $a_{r_1 i} = 1$ and $a_{r_1 j} = 0$ and then would have deleted the arc (i, j). But by assumption the arc (i, j) is deleted when visiting arc (r, i) and not an arc starting at r_1. So we get a contradiction. And thus if $s \neq j$ we have a contradiction.

Case Ib: $(r_1 = j)$

Let the arc (r, j) be deleted when the algorithm was visiting the arc (r_1, k) (that is (j, k)) for some k. Since the arc (j, k) is deleted after the arc (i, j) is visited and before the arc (r, i) is visited, so $i < j < r$.

Now consider the stage when the arc (j, k) is visited by the algorithm. If arc (i, j) is not present at that time then the arc (i, j) would have been deleted which would contradict the assumption that the arc (i, j) is deleted when the algorithm was visiting (r, i). So just before the stage when the algorithm was visiting arc (j, k) the situation would have been like in Figure 2(right)).

So the arc (i, k) was present when the algorithm was visiting the arc (j, k). But since $i < j$ so the arc (i, k) must have been visited already. By the minimality condition that (i, j) is the first arc that is visited and then deleted and since the arc (i, j) is deleted when visiting arc (r, i), so when the algorithm just started visiting the arc (r, i) the arc (i, k) must be present. Also at that stage the arc (r, k) was absent as it was absent when visiting the arc (j, k) and $j < r$. So when the algorithm just started to visit (r, i) the situation would have been like in Figure 2(right)) except the arc (r, j) would also have been missing.

When the algorithm was visiting the arc (j, k) the arc (r, k) was not there. But when the algorithm visited the arc (i, j) the arc (r, k) must have been there, else the arc (r, i) would have been deleted at that stage, which would contradict our assumption that (i, j) was deleted when visiting (r, i). So the arc (r, k) must have been deleted after the arc (i, k) was visited but before the arc (j, k) was visited.

If the arc is deleted when visiting some arc starting with k then it means that $i < k < j$. Now consider the stage when the algorithm was visiting (r, i). As described earlier the situation would have been like in Figure 2(right)) except the arc (r, j) would also have been missing. Since $k < j$ so the algorithm would have deleted (i, k) before it deleted (i, j). And since the algorithm has also visited (i, k) earlier so this contradicts the the minimality condition of (i, j) being the first visited arc to be deleted.

The other case being the arc deleted when visiting the some arc ending in r, say (t, r), where $i < t < j$. Thus during the t-th iteration of Line 1 the arcs (t, r) is present while the arc (t, k) is absent. Now, since in the t-th iteration the arc (r, j) is not deleted thus it means that the arc (t, j) was present during the t-th iteration of Line 1. But in that case since arcs (t, j) and (j, k) are present while (t, k) is not present the algorithm would have deleted the arc (j, k) in the t-th iteration of Line 1, this contradicts the assumption that the arc (r, j) is deleted in the j-th iteration of Line 1 when visiting the arc (j, k).

Thus the arc (i, j) cannot be deleted by the algorithm in Line 6 when visiting an arc starting from r.

Case II. Suppose (i, j) is deleted in Line 9, when the algorithm was visiting an arc starting from vertex r. In this case $j = r$. And since $r > i$ so $j > i$. Say the arc (i, j) is deleted when visiting arc (j, k), for some vertex k. Since the algorithm is deleting (i, j) in Line 9 so from Line 3 and Line 8 we have, at that stage, $a_{jk} = 1$ and $a_{ik} = 0$ (cf. Figure 1(left)).

Now if a_{ik} was 0 when the algorithm visited the arc (i, j) then the algorithm would have found $a_{ik} = 0$ and $a_{i,j} = 1$ and in that case would have deleted the arc (j, k) in Line 6. That would give a contradiction as in a later stage of the algorithm (in particular in the j-th iteration of Line 1, with $j > i$) the arc (j, k) is present. So when the arc (i, j) was visited the arc (i, k) was present.

Since by Remark 1(5) the arc (i, k) cannot be deleted in the ith iteration of Line 1, so the arc (i, j) must have been visited in the i-th iteration of Line 1 and must have been deleted by the algorithm at a later time but before the arc (i, j) is deleted. But this would contradict the minimality of the arc (i, j).

Hence even in this case also we get a contradiction. So this completes the proof.

The second lemma we need is the following. Because of limitation of space we move the proof to the appendix.

Lemma 2. *The matrix T output by the **Algorithm 1** is transitive.*

Using Lemma 2 and Lemma 1 we can finally prove the correctness of the algorithm. Unfortunately because of the shortage of space we have to push the proof of the following lemma to the appendix.

Lemma 3. *The matrix T output by the **Algorithm 1** is a maximal transitive relation contained in A.*

3.3 Better Running Time Analysis of Algorithm 1

If we do a better analysis of the running time of the **Algorithm 1** we can see that the algorithm has running time $O(n^2 + nm)$. To see it more formally consider a new pseudocode of the algorithm that we present as Algorithm 2. It is not hard to see that both the algorithms are basically same.

Theorem 5. *Algorithm 2 correctly finds a maximal transitive relation contained in a given binary relation in $O(n^2 + mn)$, where m is the number of 1's in A.*

Proof. The proof for correctness is same as in Theorem 4. We calculate only the time complexity of the algorithm and it is given by

$$\sum_{i=1}^{n}(n + k_i n), \text{ (where } k_i \text{ is the number of 1's in the } i^{\text{th}} \text{ row)}$$

$$= n^2 + n\sum_{i=1}^{n} k_i = n^2 + mn.$$

Algorithm 2. Finding a maximal transitive sub-relation

```
Input  : An $n \times n$ matrix $A = (a_{ij})$ representing a binary relation.
Output: A matrix $T = (t_{ij})$ which is a maximal transitive relation contained in
        the given binary relation $A$.
for $i \leftarrow 1$ to $n$ do
    Initialize $B_i = \emptyset$
    for each $s \leftarrow 1$ to $n$, $j \neq i$ do
        if $a_{ij} = 1$ then
            Include $j$ in $B_i$
        end
    end
    for each $j \in B_i$ do
        for each $k = 1$ to $n$ do
            if $k \neq j$ and $a_{ik} = 0$ then
                Make $a_{jk} = 0$
            end
            if $k \neq i$ and $a_{kj} = 0$ then
                Make $a_{ki} = 0$
            end
        end
    end
end
```

4 Maximum Transitive Relation

In this section, we study the problem of obtaining a maximum transitive relation contained in a binary relation. We will be using the notation of directed graphs for binary relations. In the following discussion, a 'graph' is a directed graph unless otherwise stated.

First, we state a well known result from graph theory.

Lemma 4. *There exists a bipartite subgraph of size $m/2$ in any graph with m edges.*

Obtaining such a bipartite graph deterministically in poly-time is a folklore result. Observe that every bipartite graph is transitive. Given a cut of size k in a directed graph, we can always obtain a transitive subgraph of size at least $k/2$ by considering all the edges in one direction - the direction which has more number of edges. This gives the following.

Theorem 6. *There exists a poly-time algorithm to obtain an $m/4$ sized transitive subgraph in any directed graph. This gives a $1/4$-approximation algorithm for maximum transitive subgraph problem.*

The obvious question is - can we do better than $m/4$? We claim that the constant $1/4$ can not be improved in poly-time. For this we consider the class of triangle-free graphs. From a recent result [3], we have the following result.

Theorem 7. *For every m, there exists a triangle-free graph with m edges in which the size of any directed cut is at most $m/4 + cm^{4/5}$.*

Obtaining a transitive subgraph of size better than $m/4$ (in the constant multiple) would contradict the theorem - since we could input the counterexample triangle-free directed graph and improve our cut size.

In order to improve upon the approximation factor, we focus on the class of *triangle-free* directed graphs. First we make the following simple observation about triangle-free directed graphs.

Lemma 5. *In any transitive subgraph of a triangle-free graph, there are no directed paths of length two.*

Let G be a graph and U, V be a partition of the vertex set of H. A *directed cut* (U, V) is the set of edges with a starting in U and ending point in V. The MAX-DICUT problem is the problem of obtaining a largest directed cut in a graph. This is NP-hard. [6] gives an approximation algorithm for the MAX-DICUT problem.

Theorem 8 (see [6]). *There exists a 0.874-approximation algorithm for the MAX-DICUT problem.*

As a corollary of Lemma 5, we have the following.

Lemma 6. *In a triangle-free graph, every directed cut is also a transitive subgraph.*

This implies that finding the maximum transitive subgraph is same as the MAX-DICUT problem for triangle free graphs.

Theorem 9. *There exists a 0.874-approximation algorithm for finding the maximum transitive subgraph in a triangle-free graph.*

5 Conclusion

We have presented an algorithm that given a directed graph on n vertices and m arcs outputs a maximal transitive sub-graph is time $O(n^2 + nm)$. This is the first algorithm for finding maximal transitive subgraph that we know of, that does better than the usual greedy algorithm. Although it might be the case that this is an optimal algorithm, we are unable to prove a lower bound for this problem.

There are many related problems for which one might expect similar kind of algorithm - that is $O(n^3)$ time algorithm that does better than the usual greedy algorithm. We would like to present them as open problems:

1. Given a directed graph G on n vertices and a transitive subgraph H of G, check if H is a maximal transitive subgraph of G.
2. Given a directed graph G on n vertices and a subgraph H of G, find a maximal transitive subgraph of G that also contains H.

Obviously an algorithm for the second problem would also give an algorithm for the first problem.

Appendix

Proof of Lemma 2

Lemma 2. *The matrix T output by the **Algorithm 1** is transitive.*

Proof. Suppose $t_{ij} = 1 = t_{jk}$. By Remark 1(1) no arc is created. So at all stages and in particular, at the initial stage $a_{ij} = a_{jk} = 1$. Suppose $a_{ik} = 0$ at the initial stage. Then when the algorithm visited (i, j) or (j, i) (whichever comes first), the arc (j, k) or (i, j) (respectively) will be deleted for the lack of the arc (i, k), as $a_{ij} = a_{jk} = 1$ throughout (cf. Figure 3).

Thus suppose the arc (i, k) is deleted at some stage, say, r-th iteration of Line 1. Now $r > i, j$ for otherwise the arc (i, k) would be deleted before the i-th or j-th iteration of Line 1. And in that case in the i-th or j-th iteration of Line 1 (depending on which of i and j is smaller) of Line 1 either (j, k) or (i, j) would be deleted. And then at the end at least one of t_{ij} and t_{ik} must be 0.

But then the arc (i, k) during the i-th iteration of Line 1 (as $i < r$). Since no arc is deleted once it is visited by Lemma 1, we have $t_{ik} = 1$. Therefore T is transitive.

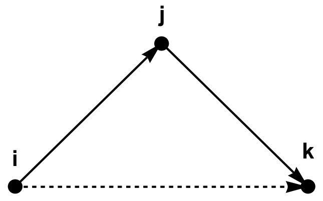


Fig. 3. Diagram for Lemma 2

Proof of Lemma 3

Lemma 3. *The matrix T output by the **Algorithm 1** is a maximal transitive relation contained in A.*

Proof. T is transitive by Lemma 2. Also by Remark 1(1) the output matrix is contained in A. So the only thing remaining to prove is that the output matrix T is maximal.

Now if T is not a maximal transitive sub-relation then there must be some arc (say (a, b)) such that the transitive closure of $T \cup \{(a, b)\}$ is also contained in A.

Now by Lemma 1, an arc once visited can never be deleted. Also the algorithm is visiting every undeleted arc. Thus T is the collection of visited arcs and these arcs are present at every stage of the algorithm.

Thus, every arc in the transitive closure of $T \cup \{(a, b)\}$ that is not in T must have been deleted in some iteration of Line 1. Let (i, j) be the first arc to be deleted among all the arcs that are in the of transitive closure of $T \cup \{(a, b)\}$ but not in T.

Clearly the transitive closure of $T \cup \{(i, j)\}$ is also contained in A, and all the arcs in the transitive closure of $T \cup \{(i, j)\}$ either is never deleted or is deleted after the arc (i, j) is deleted. Suppose the arc (i, j) is deleted in the r-th iteration of Line 1. We have $r \neq i$ by Remark 1(5) and by Lemma 1 we have $r < i$.

We now consider two cases depending on whether r is j or not.

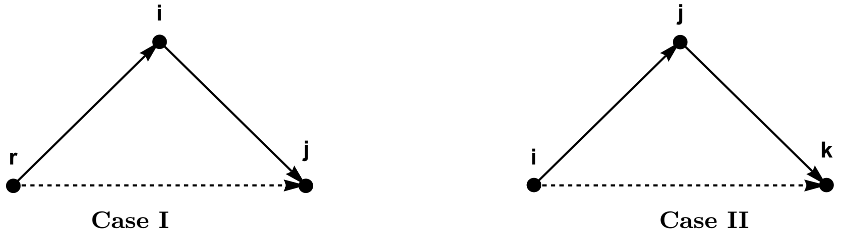


Fig. 4. Diagrams of the two cases for Lemma 3

Case I: $r \neq j$

In this case, since the arc (i, j) was deleted in the r-iteration of Line 1, the arc (i, j) must have been deleted when the algorithm was visiting the arc (r, i). So at the stage when the arc (i, j) was deleted, the arc (r, j) must not have been there (else the algorithm wouldn't have deleted the arc (i, j)).

If $a_{rj} = 0$ in A, then $t_{rj} = 0$ (by Remark 1(1)). But by Lemma 1 $t_{ri} = 1$ as the arc (r, i) is being visited. So $T \cup \{(i, j)\}$ is not transitive (cf. Figure 4(left)), and the transitive closure of $T \cup \{(i, j)\}$ must contain the arc (r, j). Thus $a_{rj} = 1$ in A, but the arc (r, j) is deleted in some stage of the algorithm but before the visit of the r-th iteration of Line 1, say, at r_1-th iteration of Line 1, with $r_1 < r$.

Thus the arc (r, j) is in the transitive closure of $T \cup \{(i, j)\}$ and it got deleted before the deletion of arc (i, j). This is a contradiction to the fact that the arc (i, j) was the first arc to be deleted. So when $r \neq j$ we have a contradiction.

Case II: $r = j$

In this case, since the arc (i, j) was deleted in the j-iteration of Line 1, the arc (i, j) must have been deleted when the algorithm was visiting some arc (j, k), for some vertex k. So at the stage when the arc (i, j) was deleted, the arc (i, k) must not have been there (else the algorithm wouldn't have deleted the arc (i, j)).

If $a_{ik} = 0$ in A, then $t_{ik} = 0$ (by Remark 1(1)). But by Lemma 1 $t_{jk} = 1$ as the arc (j, k) is being visited. So $T \cup \{(i, j)\}$ is not transitive (cf. Figure 4(right)), and the transitive closure of $T \cup \{(i, j)\}$ must contain the arc (i, k). Thus $a_{ik} = 1$ in A, but the arc (i, k) is deleted in some stage of the algorithm but before the visit of the j-th iteration of Line 1, say, at r_1-th iteration of Line 1, with $r_1 < j$.

Thus the arc (i, k) is in the transitive closure of $T \cup \{(i, j)\}$ and it got deleted before the deletion of arc (i, j). This is a contradiction to the fact that the arc (i, j) was the first arc to be deleted. So when $r = j$ we have a contradiction.

Since in both the case we face a contradiction so we have that the output T is a maximal transitive relation contained in A.

References

1. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. **1**(2), 131–137 (1972)
2. Alvarado, F.L., Pothen, A., Schreiber, R.: Highly parallel sparse triangular solution. In: George, A., Gilbert, J.R., Liu, J.W.H. (eds.) Graph Theory and Sparse Matrix Computation. The IMA Volumes in Mathematics and its Applications, vol. 56, pp. 141–157. Springer, New York (1993)
3. Chakraborty, S., Jha, N.: On the size of maximum directed cuts in triangle free graphs (2015) (unpublished)
4. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990)
5. Purdom Jr., P.W.: A transitive closure algorithm. BIT **10**, 76–94 (1970)
6. Lewin, M., Livnat, D., Zwick, U.: Improved rounding techniques for the MAX 2-sat and MAX DI-CUT problems. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 67–82. Springer, Heidelberg (2002)
7. Moyles, D.M., Thompson, G.L.: An algorithm for finding a minimum equivalent graph of a digraph. J. ACM **16**(3), 455–460 (1969)
8. Nuutila, E.: Efficient transitive closure computation in large digraphs. Acta Polytechnica Scandinavia: Math. Comput. Eng. **74**, 1–124 (1995)
9. Warshall, S.: A theorem on boolean matrices. J. ACM **9**(1), 11–12 (1962)
10. Williams, V.V.: Multiplying matrices faster than coppersmith-winograd. In: Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012, pp. 887–898 (2012)
11. Yannakakis, M.: Node- and edge-deletion np-complete problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1–3, 1978, San Diego, California, USA, pp. 253–264 (1978)

An Improved Kernel for the Complementary Maximal Strip Recovery Problem

Shuai Hu, Wenjun Li(✉), and Jianxin Wang

School of Information Science and Engineering, Central South University, Changsha 410083, People's Republic of China
liwenjun@csu.edu.cn

Abstract. We study the parameterized complexity of the complementary maximal strip recovery problem (CMSR), which is to delete the minimum number of gene markers from two genetic maps so that the remaining markers in the maps can be partitioned into matched strips. It is known that the CMSR problem has a kernel of size bounded by $78k$, and a question has been raised whether this bound can be further improved. In this paper, we answer this question by presenting an improved kernel of size $58k$ for the CMSR problem. Our results are based on the techniques of building a weighted bipartite graph from a given instance of the CMSR problem so that three additional and more powerful reduction rules can be applied to further reduce the kernel size.

1 Introduction

In comparative genomics, a critical task is to decompose two or more genomes into syntenic blocks, which are segments of chromosomes with similar contents. This task is non-trivial when the genetic maps include noise and ambiguities. The noise and ambiguities should be removed if we want to construct an accurate syntenic block decomposition.

Motivated by this, a combinatorial problem has been formulated. We start with some terminologies. A (gene) *marker* is given as a distinct symbol that is signed either positive or negative (the symbol represents a family of the gene, and the sign represents the orientation). A *genetic map* is a sequence of distinct markers. A *strip* is a substring in a genetic map consisting of at least 2 markers. Two strips *match* if either they are the same or one can be obtained from the other by first reversing the sequence then negating all symbols (note that negating a negative symbol gives a positive symbol). Since all markers in a genetic map are distinct, two matched strips must come from two genetic maps. The *maximal strip recovery* problem (MSR) [5] is to remove markers from two given genetic maps, with an optimization objective of keeping the maximum number of markers, so that the resulting sequences can be partitioned into strips such that there is a one-to-one mapping that maps each of the strips in one sequence to a

This work is supported by the National Natural Science Foundation of China under Grants (61232001, 61472449, 61420106009, 61402054).

DOI: 10.1007/978-3-319-21398-9_47

matched strip in the other sequence. The *complementary maximal strip recovery* problem (CMSR) is the same problem but with a complementary optimization objective of minimizing the total number of removed markers.

For example, consider two genetic maps (where a symbol without bar is a positive symbol and a symbol with a bar is a negative symbol):

$$G_1 = (a_1, a_2, \bar{a}_3, a_4, a_5, a_6, \bar{a}_7, \bar{a}_8, a_9, a_{10}, a_{11}),$$
$$G_2 = (\bar{a}_9, \bar{a}_4, a_7, \bar{a}_6, a_8, a_1, a_3, a_2, \bar{a}_{11}, \bar{a}_{10}, \bar{a}_5).$$

After removing markers $\bar{a}_3$, a_4, a_5, $\bar{a}_8$ from G_1 and markers $\bar{a}_4$, a_8, a_3, $\bar{a}_5$ from G_2, G_1 becomes $G_1' = (a_1, a_2, a_6, \bar{a}_7, a_9, a_{10}, a_{11})$, which can be partitioned into three strips (a_1, a_2), $(a_6, \bar{a}_7, a_9)$, (a_{10}, a_{11}), and G_2 becomes $G_2' = (\bar{a}_9, a_7, \bar{a}_6, a_1, a_2, \bar{a}_{11}, \bar{a}_{10})$, which can be partitioned into three strips $(\bar{a}_9, a_7, \bar{a}_6)$, (a_1, a_2), $(\bar{a}_{11}, \bar{a}_{10})$. It is easy to see that there is a one-to-one mapping that maps each of the strips in G_1' to a matched strip in G_2'.

The complexity of the MSR and CMSR problems has been studied. It is known that both problems are NP-hard [6]. Recently, it has also been shown that both MSR and CMSR are APX-hard [7–9]. Heuristic algorithms based on MIS and Max Clique have been developed for the problems [5,10]. The MSR problem has a polynomial-time 4-approximation algorithm [11]. The CMSR problem has a polynomial-time 3-approximation algorithm [12], and this ratio has been improved recently to 2.33 [13].

The parameterized complexity of the problems has also been considered. Formally, an instance of the (parameterized) CMSR problem takes the form $(G_1, G_2; k)$, where G_1 and G_2 are genetic maps of the same length, and one needs to decide whether k markers can be removed from each of G_1 and G_2 so that the resulting sequences can be partitioned into strips such that there is a one-to-one mapping that maps each of the strips in one sequence to a matched strip in the other sequence. There is an $O^*(3^k)$-time parameterized algorithm for CMSR [12], and this upper bound has been improved recently to $O^*(2.36^k)$ [14].

An issue closely related to parameterized algorithms is kernelization. A *kernelization algorithm* for CMSR is a polynomial-time algorithm that on an instance $(G_1, G_2; k)$ produces another instance $(G_1', G_2'; k')$ such that (1) $(G_1, G_2; k)$ is a YES-instance if and only if $(G_1', G_2'; k')$ is a YES-instance; (2) $k' \leq k$, and (3) the length of G_1' plus the length of G_2' is bounded by a function $h(k)$ of the parameter k. The instance $(G_1', G_2'; k')$ is called the *kernel*, and the function $h(k)$ is the *size* of the kernel. Kernelization algorithms measure the effectiveness of polynomial-time preprocessing on a problem. Very recently, Jiang and Zhu presented a kernelization algorithm for the CMSR problem whose kernel size is bounded by $78k$ [15]. At the end of their work, Jiang and Zhu asked whether this bound on the kernel size can be further improved [15].

In the current paper, we are focused on kernelization algorithms for the CMSR problem. In particular, we present a kernelization algorithm for the CMSR problem that has an improved kernel of size bounded by $58k$. Our result is based on techniques of building a weighted bipartite graph from a given instance of the CMSR problem so that besides the reduction rules proposed by Jiang and

Zhu [15], three additional and more powerful reduction rules can be applied to further reduce the kernel size.

2 Known Rules and Properties of CMSR

In this section, we briefly introduce some known reduction rules and structure properties for CMSR that are going to be needed during our proofs. Rule 1-8 correspond to Rule 2.1-2.8 in [15] respectively.

Given an instance $(G_1, G_2; k)$ of CMSR, we can identify, in quadratic time, all maximal substrings in G_1 and G_2 that make matched strips. Such a maximal substring of length larger than 1 is called a *block*, and those of length 1 are called *isolators*. A maximal substring in either G_1 or G_2 that consists of only blocks but no isolators is called a *super-block*. Obviously, super-blocks and sequences of continuous isolators appear alternatively in G_1 and G_2 respectively.

Then, we can construct a weighted bipartite graph $G = (V_1, V_2, E)$, where each vertex in V_1 or V_2 represents a super-block of G_1 or G_2, and there is an edge $[v_1, v_2] \in E$ between two super-blocks $v_1 \in V_1$, $v_2 \in V_2$ iff they share two matched blocks of length 2 or 3. We mention that the weights of vertices and edges work only for the analysis of the kernel, therefore we define the weight function later.

Let Σ be the symbol set for the input maps G_1 and G_2, Σ_1 be a new symbol set, with $\Sigma_1 \cap \Sigma = \emptyset$. A new symbol in Σ_1 is used in the reduction rules, in order to indicate a surely retained block in some optimal solution.

In the subsequent kernelization process, when we change the two matched blocks of length 2 or 3 into a same new Σ_1 symbol , it also implies that deleting the corresponding edge.

Lemma 1. [12] *There exists an optimal CMSR solution S satisfying that (1) for any block b of length at least 4, b is totally disjoint with S; (2) for any block c of length 2 or 3, c is either totally included in S, or totally disjoint with S.*

Rule 1. For each pair of matched blocks of length at least 4, change them into a same new symbol in Σ_1 (and delete the corresponding old symbols in it from Σ whenever such a new symbol in Σ_1 is created).

Rule 2. For any pair of super-blocks $s_1 \in V_1$, $s_2 \in V_2$ which contain at least two pairs of length-2 or length-3 matched blocks, ie., there are multi-edges between s_1 and s_2 in the weighted bipartite graph, change each pair of the matched blocks into a same new symbol in Σ_1.

Rule 3. For any super-block (in V_1 or V_2) containing at least two length-3 blocks, change each length-3 block in this super-block and its matched block into a new symbol in Σ_1.

Rule 4. For any cycle in G, identify the length-2 or length-3 blocks involved in the cycle and change each matched pair of them into a new symbol in Σ_1.

Rule 5. Within any super-block, for all blocks between two symbols in Σ_1, change each of them and its matched block into a new symbol in Σ_1. For the leftmost (rightmost) super-block in G_1 and G_2, if there is no isolator on its left (right), for all blocks on the left (right) of symbols in Σ_1, change each of them and its matched block into a new symbol in Σ_1.

Rule 6. If there exists a length-3 block in a super-block which contains some Σ_1 symbol, then change this length-3 block and its matched block into a new Σ_1 symbol.

Rule 7. If there exists a length-3 block in the leftmost (resp. rightmost) super-block in G_1 or G_2, without any isolator on its left (resp. right), then change this length-3 block and its matched block into a new Σ_1 symbol.

Rule 8. For each sequence of Σ_1 symbols $\pi_1, \pi_2, ..., \pi_j$, where π_i corresponds to some block $C_i, 1 \leq i \leq j$.

Step (I): If the corresponding super-block is the leftmost (resp. rightmost) super-block in G_1 or G_2, without any isolator on its left (resp. right), then change $C_j = d_1 d_2 \cdots d_t$ (resp. $C_1 = c_1 c_2 \cdots c_r$) into a length-3 block $C_j = d_1 \beta d_t$ (resp. $C_1 = c_1 \alpha c_r$), and keep C_j (resp. C_1); Otherwise, if $j = 1$, then change $C_1 = c_1 c_2 \cdots c_r$ into a length-4 block $c_1 abc_r$, where a, b are new symbols not in Σ or Σ_1, and keep C_1. If $j > 1$, then change $C_1 = c_1 c_2 \cdots c_r$ into a length-3 block $C_1 = c_1 \alpha c_r$ and $C_j = d_1 d_2 \cdots d_t$ into a length-3 block $C_j = d_1 \beta d_t$, where α, β are new symbols not in Σ or Σ_1, and keep C_1 and C_j.

Step (II): Finally, delete all the blocks that are not kept in both G_1 and G_2.

Lemma 2. [15] *Let G contain m connected components $H_1, H_2, ..., H_m$, and let H_i have q_i edges. Then, in between the vertices in G, there are at least $\Sigma_{i=1}^{m} q_i + m - 2$ sequences of continuous isolators in G_1, G_2.*

Corollary 1. *G has at most $4k + 2$ vertices.*

3 New Rules

In this section, we introduce three new reduction rules.

Rule 9. (1) If there exist four super-blocks $v_{11}, v_{12}, v_{21}, v_{22}$ in G, such that (i) v_{11} and v_{21} share two matched length-3 blocks, (ii) v_{11} and v_{22} share two matched length-2 blocks, (iii) v_{12} and v_{22} share two matched length-3 blocks ; or (2) if there exist three super-blocks v_{11}, v_{21}, v_{22} in G, such that (i) v_{11} and v_{21} share two matched length-3 block, (ii) v_{11} and v_{22} share two matched length-2 blocks, (iii) v_{22} contains some Σ_1 symbol.

Then, change each pair of these matched blocks into a new symbol in Σ_1.

Rule 10. If there exist v_{11}, v_{21}, v_{22} in G, such that (i) v_{11} and v_{21} share two matched length-2 blocks, (ii) v_{11} and v_{22} share another two matched length-2 blocks, (iii) v_{21} and v_{22} respectively contain some Σ_1 symbol, then change each pair of these matched length-2 blocks into a new symbol in Σ_1.

Rule 11. If there exist two super-blocks v_{11}, v_{21} in G, such that (i) v_{11}, v_{21} share two matched length-2 blocks, (ii) v_{11}, v_{21} either contains some Σ_1 symbol or be the leftmost (rightmost) super-block with no isolators on its left (right), then change this pair of matched length-2 blocks into a new symbol in Σ_1.

Lemma 3. *Rule 9 is correct.*

Proof. If condition (1) is satisfied, without loss of generality, assume that v_{11} and v_{12} are in G_1, v_{21} and v_{22} are in G_2, v_{21} contains Q_1, v_{11} contains Q_1, P_1, v_{22} contains P_1, Q_2, v_{12} contains Q_2, where Q_i $(i = 1, 2)$ is some length-3 block, P_1 is some length-2 block. Obviously, Q_1 has at most 3 isolated neighbors (1 in G_1, 2 in G_2), Q_2 has at most 3 isolated neighbors (2 in G_1, 1 in G_2), P_1 has at most 2 isolated neighbors (one each in G_1 and G_2). We distinguish three cases. Case (i), if some optimal solution deletes only one of P_1, Q_1, Q_2, by the fact that Q_i is of length 3, P_1 is of length 2, we can keep P_1 or Q_1 or Q_2 as strips to have a solution at least as good as the assumed optimal solution. Case (ii), if some optimal solution deletes only two of P_1, Q_1, Q_2, by the fact that the length of the deleted blocks are not shorter than the total length of their isolated neighbors, we can keep P_1 and Q_i $(i \in \{1, 2\})$ (or Q_1 and Q_2) as strips to have a solution at least as good as the assumed optimal solution. Case (iii), if some optimal solution deletes P_1, Q_1 and Q_2, by the same fact as case (ii), we can keep P_1, Q_1 and Q_2 as strips to have a solution at least as good as the assumed optimal solution. If condition (2) is satisfied, the proof is quite the same as the proof when condition (1) is satisfied, hence omitted. □

Lemma 4. *Rule 10 is correct.*

Proof. Assume that v_{11} is in G_1, v_{21} and v_{22} are in G_2, the two pair of matched blocks involved are $P_1(P_2)$ and $P_3(P_4)$. Each of them has at most 2 isolated neighbors. If some optimal solution deletes one (or both) of them, by the fact that they are of length 2, we can keep one (or both) of them as strips to have a solution at least as good as the assumed optimal solution. □

Lemma 5. *Rule 11 is correct.*

Proof. Assume that v_{11} and v_{21} share two matched length-2 blocks, denoted by $P_1(P_2)$, Simultaneously, v_{11}, v_{21} either contains some Σ_1 symbol or be the leftmost (rightmost) super-block with no isolators on its left (right). Clearly, $P_1(P_2)$ has at most 2 isolated neighbors (1 in G_1, 1 in G_2). If there is some optimal solution deletes P_1 (i.e., P_2), by the fact P_1 is length-2 block, we can keep P_1 as strips to have a solution at least as good as the assumed optimal solution. □

4 An Improved Kernel for CMSR

To obtain the improved kernel, we need to apply the rules in the following order: 1, 2, 3, 4, 9, 6, 7, 10, 11, 5, 8. Let the resulting sequences be G'_1, G'_2.

In [15], Jiang and Zhu used an inverse amortized analysis. Here, we also utilize this method to divide G_1 or G_2 into four sets : A – the CMSR solution, which contains those symbols/blocks we remove from each of G_1 and G_2 (of a total length k); B – those isolators which were kept in some strips in $G_1 - A$ (also in $G_2 - A$); C – those blocks identified by our kernelization algorithm, which are changed into Σ_1 symbols; and D – the length-2/3 blocks that are not removed in A, neither identified in C, these D blocks correspond to the edges in the graph G.

To distinguish whether a super-block includes a Σ_1 symbol, we use a *solid vertex* to represent a super-block which includes one or more Σ_1 symbols, and use a *hollow vertex* to represent a super-block which includes no Σ_1 symbol.

Theorem 1. *CMSR has a kernel of size* $58k$*.*

Proof. The type-C set has been changed into new length-3/4 blocks in G'_1 (also G'_2) by Rule 8. Let $|A'|, |B'|, |C'|, |D'|$ denote the total length of type-A, B, C, D sets in G'_1 and G'_2 respectively.

From [15], we have known that

$$|A'| + |B'| = 2|A| + 2|B| \leq 10k \tag{1}$$

We firstly define weights of edges, $w : E \longrightarrow R$, $w(e) = 4$, if e corresponds to two matched blocks of length 2; $w(e) = 6$, if e corresponds to two matched blocks of length 3. Actually,

$$|D'| = \Sigma_{e \in G} w(e) \tag{2}$$

If Rule 1, 2, 3, 4, 9, 6, 7, 10, 11, 5 are applied (Rule 8 is a rule for recovery), the resulting graph G has many useful properties, we summarize below:

(1) G is a forest, according to Rule 2, 4.

(2) Every super-block could contains at most one length-3 block, ie., every vertex is adjacent to at most one edge of weight 6, according to Rule 3.

(3) The Σ_1 symbols appear continuously in any super-block, according to Rule 5.

(4) In any super-block, a Σ_1 symbol and a length-3 block can not exist at the same time, ie., a solid vertex can't be adjacent to an edge of weight 6, according Rule 6.

(5) A solid vertex can't be adjacent to a solid vertex or a vertex/super-block which is the leftmost (resp. rightmost) vertex/super-block with no isolators on its left (resp. right), according to (4), Rule 11.

(6) A hollow vertex is adjacent to at most one solid vertex, according to Rule 10. If a hollow vertex is adjacent to a solid vertex, then for each edge adjacent to this hollow vertex, its weight must be 4, according to (4), Rule 9.

Now we define weights of vertices, $w : V_1 \cup V_2 \longrightarrow R$ by the following steps:

(1) For each hollow vertex u, initially set $w(u) = 6$.

(2) For each solid vertex u, i) if u corresponds to the leftmost (resp. rightmost) super-block, without any isolators on its left (resp. right), then set $w(u) = 6$; otherwise, set $w(u) = 12$. ii) for each vertex v adjacent to this solid vertex u, add $w([u, v])$ to $w(v)$.

It's easy to verify that, by the definition of w, for any solid vertex u, if u corresponds to the leftmost (resp. rightmost) super-block, without any isolators on its left (resp. right), then $w(u) = 6$; otherwise, $w(u) = 12$; For any hollow vertex v, if it is adjacent to some solid vertex, then $w(v) = 10$; otherwise, $w(v) = 6$.

According to Rule 8, we have

$$|C'| \leq \Sigma_{v \in G,\ v\ is\ solid} w(v) \tag{3}$$

Next, we bound $|D'|$ by the total weights of hollow vertices, ie., we prove that

$$|D'| < \Sigma_{v \in G,\ v\ is\ hollow} w(v) \tag{4}$$

Do the following steps on $V_1 \cup V_2 \cup E$ to create another weight function w' and a modified new graph:

(1) For each edge e, initially set $w'(e) = w(e)$.

(2) For each hollow vertex u, initially set $w'(u) = 6$.

(3) For each solid vertex u, i) if u corresponds to the leftmost (resp. rightmost) super-block, without any isolators on its left (resp. right), then set $w'(u) = 6$; otherwise, set $w'(u) = 12$. ii) for each vertex v adjacent to this solid vertex u, create a new hollow vertex s, and shift the edge $[v, u]$ to $[v, s]$, ie. delete edge $[v, u]$, add edge $[v, s]$, set $w'([v, s]) = w([v, u])$, and set $w'(s) = w'[v, s]$.

Let the modified new weighted graph be G'. Then, we have

$$\Sigma_{e \in G'} w'(e) = \Sigma_{e \in G} w(e) \tag{5}$$

$$\Sigma_{v \in G',\ v\ is\ hollow} w'(v) = \Sigma_{v \in G,\ v\ is\ hollow} w(v) \tag{6}$$

After the weight function w' is generated, the new graph G' may have a few new vertices, but it is still a forest. For each connected component H (i.e. a tree), H either consists of only hollow vertices, or consists of only one isolated solid vertex. Note that every vertex's weight is no less than the maximum weight of its adjacent edges, thus we have

$$\Sigma_{e \in H} w'(e) < \Sigma_{v \in H} w'(v), H \neq \phi \tag{7}$$

For all connected components besides those isolated solid vertices, we have

$$\Sigma_{e \in G'} w'(e) < \Sigma_{v \in G',\ v\ is\ hollow} w'(v) \tag{8}$$

Then,

$$\begin{aligned} |D'| &= \Sigma_{e \in G} w(e) \\ &= \Sigma_{e \in G'} w'(e) \\ &< \Sigma_{v \in G',\ v\ is\ hollow} w'(v) \\ &= \Sigma_{v \in G,\ v\ is\ hollow} w(v) \end{aligned} \tag{9}$$

This verifies the correctness of inequality (4). Furthermore, we have

$$|C'| + |D'| \leq \Sigma_{v \in G} w(v) \leq 12 * (4k - 2) + 6 * 4 = 48k \tag{10}$$

The case of equality occurs when $|D'| = 0$, ie., there is no edge in the graph G, every super-block contains only Σ_1 symbols.

Then the size of the kernel is bounded by

$$|A'| + |B'| + |C'| + |D'| \leq 10k + 48k \leq 58k. \tag{11}$$

□

References

1. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer-Verlag (1999)
2. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
3. Niedermeier, R.: Invitation to fixed-parameter algorithms (2006)
4. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News **38**(1), 31–45 (2007)
5. Zheng, C., Zhu, Q., Sankoff, D.: Removing noise and ambiguities from comparative maps in rearrangement analysis. IEEE/ACM Transactions on Computational Biology and Bioinformatics **4**(4), 515–522 (2007)
6. Wang, L., Zhu, B.: On the tractability of maximal strip recovery. Journal of Computational Biology **17**(7), 907–914 (2010)
7. Bulteau, L., Fertin, G., Rusu, I.: Maximal strip recovery problem with gaps: hardness and approximation algorithms. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 710–719. Springer, Heidelberg (2009)
8. Jiang, M.: Inapproximability of maximal strip recovery. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 616–625. Springer, Heidelberg (2009)
9. Jiang, M.: Inapproximability of maximal strip recovery: II. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) FAW 2010. LNCS, vol. 6213, pp. 53–64. Springer, Heidelberg (2010)
10. Choi, V., Zheng, C., Zhu, Q., Sankoff, D.: Algorithms for the extraction of synteny blocks from comparative maps. In: Giancarlo, R., Hannenhalli, S. (eds.) WABI 2007. LNCS (LNBI), vol. 4645, pp. 277–288. Springer, Heidelberg (2007)
11. Chen, Z., Fu, B., Jiang, M., Zhu, B.: On recovering syntenic blocks from comparative maps. J. Comb. Optim. **18**(3), 307–318 (2009)
12. Jiang, H., Li, Z., Lin, G., Wang, L., Zhu, B.: Exact and approximation algorithms for the complementary maximal strip recovery problem. J. Comb. Optim. **23**(4), 493–506 (2012)
13. Li, Z., Goebel, R., Wang, L., Lin, G.: An improved approximation algorithm for the complementary maximal strip recovery problem. J. Comput. Syst. Sci. **78**(3), 720–730 (2012)
14. Bulteau, L., Fertin, G., Jiang, M., Rusu, I.: Tractability and approximability of maximal strip recovery. Theor. Comput. Sci. **440**(441), 14–28 (2012)
15. Jiang, H., Zhu, B.: A linear kernel for the complementary maximal strip recovery problem. Journal of Computer and System Sciences (2014)

On Symbolic Ultrametrics, Cotree Representations, and Cograph Edge Decompositions and Partitions

Marc Hellmuth[1,2](✉) and Nicolas Wieseke[3]

[1] Department of Mathematics and Computer Science, University of Greifswald, Walther-Rathenau-Strasse 47, 17487 Greifswald, Germany
mhellmuth@mailbox.org

[2] Center for Bioinformatics, Saarland University, Building E 2.1, 66041 Saarbrücken, Germany

[3] Parallel Computing and Complex Systems Group, Department of Computer Science, Leipzig University, Augustusplatz 10, 04109 Leipzig, Germany
wieseke@informatik.uni-leipzig.de

Abstract. Symbolic ultrametrics define edge-colored complete graphs K_n and yield a simple tree representation of K_n. We discuss, under which conditions this idea can be generalized to find a symbolic ultrametric that, in addition, distinguishes between edges and non-edges of arbitrary graphs $G = (V, E)$ and thus, yielding a simple tree representation of G. We prove that such a symbolic ultrametric can only be defined for G if and only if G is a so-called cograph. A cograph is uniquely determined by a so-called cotree. As not all graphs are cographs, we ask, furthermore, what is the minimum number of cotrees needed to represent the topology of G. The latter problem is equivalent to find an optimal cograph edge k-decomposition $\{E_1, \dots, E_k\}$ of E so that each subgraph (V, E_i) of G is a cograph. An upper bound for the integer k is derived and it is shown that determining whether a graph has a cograph 2-decomposition, resp., 2-partition is NP-complete.

Keywords: Symbolic ultrametric · Cograph · Cotree · Edge partition · NP-complete

1 Introduction

Given an arbitrary edge-colored complete graph $K_n = (V, E)$ on n vertices, Böcker and Dress [4] asked, whether there is a tree representation of this K_n, i.e., a rooted tree $T = (W, F)$ with leaf set V together with a labeling t of the non-leaf vertices in $W \setminus V$ so that the least common ancestor $\text{lca}(x, y)$ of distinct leaves x and y is labeled with the respective color of the edge $[x, y] \in E$. The pair (T, t) is then called symbolic representation of the edge-colored graph K_n. The authors showed, that there is a symbolic representation (T, t) if and only if the map δ that assigns colors or symbols to the edges in E fulfills the properties of a

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 609–623, 2015.
DOI: 10.1007/978-3-319-21398-9_48

so-called symbolic ultrametric [4]. Such maps are crucial for the characterization of relationships between genes or proteins, so-called orthology relations [13–15], that lie at the heart of many phylogenomic studies.

Inspired by the work of Böcker and Dress, we address the following problem: Does there exist, for an arbitrary undirected graph $G = (V, E)$ a symbolic ultrametric $\delta : V \times V \to M$ and thus, a symbolic representation (T, t) of G so that one can distinguish between edges and non-edges of G? In other words, we ask for a coloring δ of the edges $[x, y] \in E$, as well as the non-edges $[x, y] \notin E$, so that the topology of G can be displayed by a rooted vertex-labeled tree (T, t) s.t. for all distinct vertices $x, y \in V$ the labeling of the lowest common ancestor $\mathrm{lca}(x, y)$ is equal to $\delta(x, y)$. The first result of this contribution provides that such a symbolic ultrametric can only be defined for G if and only if G is a cograph. This, in particular, establishes another new characterization of cographs.

Cographs are characterized by the absence of induced paths P_4 on four vertices. Moreover, Lerchs [16,17] showed that each cograph $G = (V, E)$ is associated with a unique rooted tree $T(G)$, called *cotree*. Obviously, not all graphs are cographs and thus, don't have a cotree representation. Therefore, we ask for the minimum number of cotrees that are needed to represent the structure of a given graph $G = (V, E)$ in an unambiguous way. As it will turn out, this problem is equivalent to find a decomposition $\Pi = \{E_1, \ldots, E_k\}$ of E (the elements of Π need not necessarily be disjoint) for the least integer k, so that each subgraph $G_i = (V, E_i)$, $1 \leq i \leq k$ is a cograph. Such a decomposition is called cograph edge k-decomposition, or cograph k-decomposition, for short. If the elements of Π are in addition pairwise disjoint, we call Π a cograph k-partition. We will prove that finding the least integer $k \geq 2$ so that G has a cograph k-decomposition or a cograph k-partition is an NP-hard problem. Moreover, upper bounds for the integer k for any cograph k-decomposition are derived. These findings complement results known about so-called cograph vertex partitions [1,10,11,26].

2 Basic Definitions

Graph. In what follows, we consider undirected simple graphs $G = (V, E)$ with vertex set $V(G) = V$ and edge set $E(G) = E \subseteq \binom{V}{2}$. The *complement graph* $G^c = (V, E^c)$ of $G = (V, E)$, has edge set $E^c = \binom{V}{2} \setminus E$. The graph $K_{|V|} = (V, E)$ with $E = \binom{V}{2}$ is called *complete graph.* A graph $H = (W, F)$ is an *induced subgraph* of $G = (V, E)$, if $W \subseteq V$ and all edges $[x, y] \in E$ with $x, y \in W$ are contained in F. The *degree* $\deg(v) = |\{e \in E \mid v \in e\}|$ of a vertex $v \in V$ is defined as the number of edges that contain v. The maximum degree of a graph is denoted with Δ.

Rooted Tree. A connected graph T is a *tree*, if T does not contain cycles. A vertex of a tree T of degree one is called a *leaf* of T and all other vertices of T are called *inner* vertices. The set of inner vertices of T is denoted by V^0. A *rooted tree* $T = (V, E)$ is a tree that contains a distinguished vertex $\rho_T \in V$ called the *root.* The first inner vertex $\mathrm{lca}_T(x, y)$ that lies on both unique paths from distinct leaves x, resp., y to the root, is called *most recent common ancestor* *of x and y*. If there is no danger of ambiguity, we will write $\mathrm{lca}(x, y)$ rather then $\mathrm{lca}_T(x, y)$.

Symbolic Ultrametric and Symbolic Representation. In what follows, the set M will always denote a non-empty finite set, the symbol $\odot$ will always denote a special element not contained in M, and $M^{\odot} := M \cup \{\odot\}$. Now, suppose X is an arbitrary non-empty set and $\delta : X \times X \to M^{\odot}$ a map. We call δ a *symbolic ultrametric* if it satisfies the following conditions:

(U0) $\delta(x, y) = \odot$ if and only if $x = y$;
(U1) $\delta(x, y) = \delta(y, x)$ for all $x, y \in X$, i.e. δ is symmetric;
(U2) $|\{\delta(x, y), \delta(x, z), \delta(y, z)\}| \leq 2$ for all $x, y, z \in X$; and
(U3) there exists no subset $\{x, y, u, v\} \in \binom{X}{4}$ such that $\delta(x, y) = \delta(y, u) = \delta(u, v) \neq \delta(y, v) = \delta(x, v) = \delta(x, u)$.

Now, suppose that $T = (V, E)$ is a rooted tree with leaf set X and that $t : V \to M^{\odot}$ is a map such that $t(x) = \odot$ for all $x \in X$. To the pair (T, t) we associate the map $d_{(T,t)}$ on $X \times X$ by setting, for all $x, y \in X$,

$$d_{(T,t)} : X \times X \to M^{\odot}; d_{(T,t)}(x, y) = t(\mathrm{lca}_T(x, y)). \tag{1}$$

Clearly this map is symmetric and satisfies (U0). We call the pair (T, t) a *symbolic representation* of a map $\delta : X \times X \to M^{\odot}$, if $\delta(x, y) = d_{(T,t)}(x, y)$ holds for all $x, y \in X$. For a subset $W \subseteq X \times X$ we denote with $\delta(W)$ the restriction of δ to the set W.

Cographs and Cotrees. Complement-reducible graph, cographs for short, are defined as the class of graphs formed from a single vertex under the closure of the operations of union and complementation, namely: (i) a single-vertex graph is a cograph; (ii) the disjoint union of cographs is a cograph; (iii) the complement of a cograph is a cograph. Alternatively, a cograph can be defined as a P_4-free graph (i.e. a graph such that no four vertices induce a subgraph that is a path of length 3), although there are a number of equivalent characterizations of such graphs (see e.g. [6] for a survey). It is well-known in the literature concerning cographs that, to any cograph G, one can associate a canonical *cotree* $T(G) = (V, E)$. This is a rooted tree, leaf set equal to the vertex set $V(G)$ of G and inner vertices that represent so-called "join" and "union" operations together with a labeling map $t : V^0 \to \{0, 1\}$ such that for all $[x, y] \in E(G)$ it holds that $t(\mathrm{lca}(x, y)) = 1$, and $t(v) \neq t(w_i)$ for all $v \in V^0$ and all children $w_1, \ldots, w_k \in V^0$ of v, (cf. [8]).

Cograph k-Decomposition and Partition, and Cotree Representation. Let $G = (V, E)$ be an arbitrary graph. A decomposition $\Pi = \{E_1, \ldots E_k\}$ of E is a called *(cograph) k-decomposition*, if each subgraph $G_i = (V, E_i)$, $1 \leq i \leq k$ of G is a cograph. We call Π a *(cograph) k-partition* if $E_i \cap E_j = \emptyset$, for all distinct $i, j \in \{1, \ldots, k\}$. A k-decomposition Π is called *optimal*, if Π has the least number k of elements among all cograph decompositions of G. Clearly, for a cograph only k-decompositions with $k = 1$ are optimal. A k-decomposition $\Pi = \{E_1, \ldots E_k\}$ is *coarsest*, if no elements of Π can be unified, so that the resulting decomposition is a cograph l-decomposition, with $l < k$. In other words, Π is coarsest, if for all subsets $I \subseteq \{1,\ldots,k\}$ with $|I| > 1$ it holds that $(V, \cup_{i\in I} E_i)$ is not a cograph. Thus, every optimal k-decomposition is also always a coarsest one.

A graph $G = (V, E)$ is *represented by a set of cotrees* $\mathbb{T} = \{T_1,\ldots,T_k\}$, each T_i with leaf set V, if and only if for each edge $[x, y] \in E$ there is a tree $T_i \in \mathbb{T}$ with $t(\text{lca}_{T_i}(x, y)) = 1$.

The Cartesian (Graph) Product $G\Box H$ has vertex set $V(G\Box H) = V(G) \times V(H)$; two vertices (g_1, h_1), (g_2, h_2) are adjacent in $G\Box H$ if $[g_1, g_2] \in E(G)$ and $h_1 = h_2$, or $[h_1, h_2] \in E(H)$ and $g_1 = g_2$. It is well-known that the Cartesian product is associative, commutative and that the single vertex graph K_1 serves as unit element [12]. Thus, the product $\Box_{i=1}^n G_i$ of arbitrary many factors $G_1,\ldots,G_n$ is well-defined. For a given product $\Box_{i=1}^n G_i$, we define the G_i-layer G_i^w of G (through vertex w that has coordinates $(w_1,\ldots,w_n)$) as the induced subgraph with vertex set $V(G_i^w) = \{v = (v_1,\ldots,v_n) \in \times_{i=1}^n V(G_i) \mid v_j = w_j,$ for all $j \neq i\}$. Note, G_i^w is isomorphic to G_i for all $1 \leq i \leq n$, $w \in V(\Box_{i=1}^n G_i)$. The *n-cube* Q_n is the Cartesian product $\Box_{i=1}^n K_2$.

3 Symbolic Ultrametrics

Symbolic ultrametrics and respective representations as event-labeled trees, have been first characterized by Böcker and Dress [4].

Theorem 1 ([4,13]). *Suppose $\delta : V \times V \to M^{\odot}$ is a map. Then there is a symbolic representation of δ if and only if δ is a symbolic ultrametric. Furthermore, this representation can be computed in polynomial time.*

Let $\delta : V \times V \to M^{\odot}$ be a map satisfying Properties (U0) and (U1). For each fixed $m \in M$, we define an undirected graph $G_m := G_m(\delta) = (V, E_m)$ with edge set

$$E_m = \{\{x, y\} \mid \delta(x, y) = m,\ x, y \in V\}. \tag{2}$$

Thus, the map δ can be considered as an edge coloring of a complete graph $K_{|V|}$, where each edge $[x, y]$ obtains color $\delta(x, y)$. Hence, G_m denotes the subgraph of the edge-colored graph $K_{|V|}$, that contains all edges colored with $m \in M$. The following result establishes the connection between symbolic ultrametrics and cographs.

Theorem 2 ([13]). *Let $\delta : V \times V \to M^{\odot}$ be a map satisfying Properties (U0) and (U1). Then δ is a symbolic ultrametric if and only if*

(U2') For all $\{x, y, z\} \in \binom{V}{3}$ there is an $m \in M$ such that E_m contains two of the three edges $\{x, y\}$, $\{x, z\}$, and $\{y, z\}$.
(U3') G_m is a cograph for all $m \in M$.

Assume now, we have given an arbitrary subgraph $G = (V, E) \subseteq K_{|V|}$. Let δ be a map defined on $V \times V$ so that edges $e \in E$ obtain a different color then the non-edges $e \in E(K_{|V|}) \setminus E$ of G. The questions then arises, whether such a map fulfilling the properties of symbolic ultrametric can be defined and thus, if there is tree representation (T, t) of G. Of course, this is possible only if δ restricted to E, resp., E^c is a symbolic ultrametric, while it also a symbolic ultrametric on the complete graph $K_{|V|} = (V, E \cup E^c)$. The next theorem answers the latter question and, in addition, provides a new characterization of cographs.

Theorem 3. *Let $G = (V, E)$ be an arbitrary (possibly disconnected) graph, $W = \{(x, y) \in V \times V \mid [x, y] \in E\}$ and $W^c = \{(x, y) \in V \times V \mid [x, y] \notin E\}$. There is a symbolic ultrametric $\delta : V \times V \to M^{\odot}$ s.t. $\delta(W) \cap \delta(W^c) = \emptyset$ if and only if G is a cograph.*

Proof. First assume that G is a cograph. Set $\delta(x, x) = \odot$ for all $x \in V$ and set $\delta(x, y) = \delta(y, x) = 1$ if $[x, y] \in E$ and, otherwise, to 0. Hence, condition $(U0)$ and $(U1)$ are fulfilled. Moreover, by construction $|M| = 2$ and thus, condition $(U2')$ is trivially fulfilled. Furthermore, since $G_1(\delta)$ and its complement $G_0(\delta)$ are cographs, $(U3')$ is satisfied. Theorem 2 implies that δ is a symbolic ultrametric.

Now, let $\delta : V \times V \to M^{\odot}$ be a symbolic ultrametric with $\delta(W) \cap \delta(W^c) = \emptyset$. Assume for contradiction that G is not a cograph. Then G contains an induced path $P_4 = a - b - c - d$. Therefore, at least one edge e of this path P_4 must obtain a color $\delta(e)$ different from the other two edges contained in this P_4, as otherwise $G_{\delta(e)}(\delta)$ is not a cograph and thus, δ is not a symbolic ultrametric (Theorem 2). For all such possible maps δ "subdividing" this P_4 we always obtain that two edges of at least one of the underlying paths $P_3 = a - b - c$ or $b - c - d$ must have different colors. W.l.o.g. assume that $\delta(a, b) \neq \delta(b, c)$. Since $[a, c] \notin E$ and $\delta(W) \cap \delta(W^c) = \emptyset$ we can conclude that $\delta(a, c) \neq \delta(a, b)$ and $\delta(a, c) \neq \delta(b, c)$. But then condition $(U2')$ cannot be satisfied, and Theorem 2 implies that δ is not a symbolic ultrametric. □

The latter result implies, that there is no hope for finding a map δ for a graph G, that assigns symbols or colors to edges, resp., non-edges such that for δ (and hence, for G) there is a symbolic representation (T, t), unless G is already a cograph. The (decision version of the) problem to edit a given graph G into a cograph G', and thus, to find the closest graph G' that has a symbolic representation, is NP-complete [18,19]. In this contribution, we are interested in the following problem: *What is the minimum number of cotrees that are needed to represent the topology of G in an unambiguous way?*

4 Cotree Representation and Cograph *k*-Decomposition

Recollect, a graph $G = (V, E)$ is represented by a set of cotrees $\mathbb{T} = \{T_1, \ldots, T_k\}$, if and only if for each edge $[x, y] \in E$ there is a tree $T_i \in \mathbb{T}$ with $t(\text{lca}_{T_i}(x, y)) = 1$. Note, by definition, each cotree T_i determines a subset $E_i = \{[x, y] \in E \mid t(\text{lca}_{T_i}(x, y)) = 1\}$ of E. Hence, the subgraph (V, E_i) must be a cograph. Therefore, in order to find the minimum number of cotrees representing a graph G, we can equivalently ask for a decomposition $\Pi = \{E_1, \ldots, E_k\}$ of E so that each subgraph (V, E_i) is a cograph, where k is the least integer among all cograph decompositions of G. Thus, we are dealing with the following two equivalent problems.

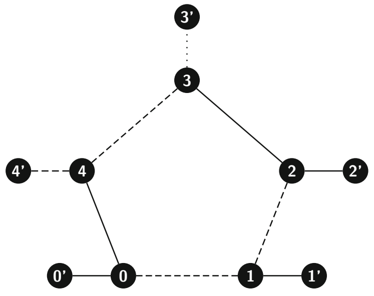


Fig. 1. Full enumeration of all possibilities (which we leaf to the reader), shows that the depicted graph has no cograph 2-decomposition. The existing cograph 3-decomposition is also a cograph 3-partition; highlighted by dashed-lined, dotted and bold edges.

Problem. COTREE k-REPRESENTATION
Input: Given a graph $G = (V, E)$ and an integer k .
Question: Can G be represented by k cotrees?

Problem. COGRAPH k-DECOMPOSITION
Input: Given a graph $G = (V, E)$ and an integer k.
Question: Is there a cograph k-decomposition of G?

Clearly, any cograph has an optimal 1-decomposition, while for cycles of length > 4 or paths P_4 there is always an optimal cograph 2-decomposition. However, there are examples of graphs that do not have a 2-decomposition, see Figure 1. To derive an upper bound for the integer k s.t. there is a cograph k-decomposition for arbitrary graphs, the next theorem is given.

Theorem 4. *For every graph G with maximum degree Δ there is a cograph k-decomposition with $1 \leq k \leq \Delta + 1$ that can be computed in $O(|V||E| + \Delta(|V| + |E|))$ time. Hence, any graph can be represented by at most $\Delta + 1$ cotrees.*

Proof. Consider a proper edge-colorings $\varphi : E \to \{1, \dots, k\}$ of G, i.e., an edge coloring such that no two incident edges obtain the same color. Any proper edge-coloring using k colors yields a cograph k-partition $\Pi = \{E_1, \dots, E_k\}$ where $E_i = \{e \in E \mid \varphi(e) = i\}$, because any connected component in $G_i = (V, E_i)$ is an edge and thus, no P_4's are contained in G_i. Vizing's Theorem [25] implies that for each graph there is a proper edge-coloring using k colors with $\Delta \leq k \leq \Delta+1$.

An proper edge-coloring using at most $\Delta + 1$ colors can be computed with the Misra-Gries-algorithm in $O(|V||E|)$ time [20]. Since the (at most $\Delta + 1$) respective cotrees can be constructed in linear-time $O(|V| + |E|)$ [9], we derive the runtime $O(|V||E| + \Delta(|V| + |E|))$. □

Obviously, any optimal k-decomposition must also be a coarsest k-decomposition, while the converse is in general not true, see Fig.2. The partition $\Pi = \{E_1, \dots, E_k\}$ obtained from a proper edge-coloring is usually not a coarsest one, as possibly (V, E_J) is a cograph, where $E_J = \cup_{i \in J} E_i$ and $J \subseteq \{1, \dots, l\}$.

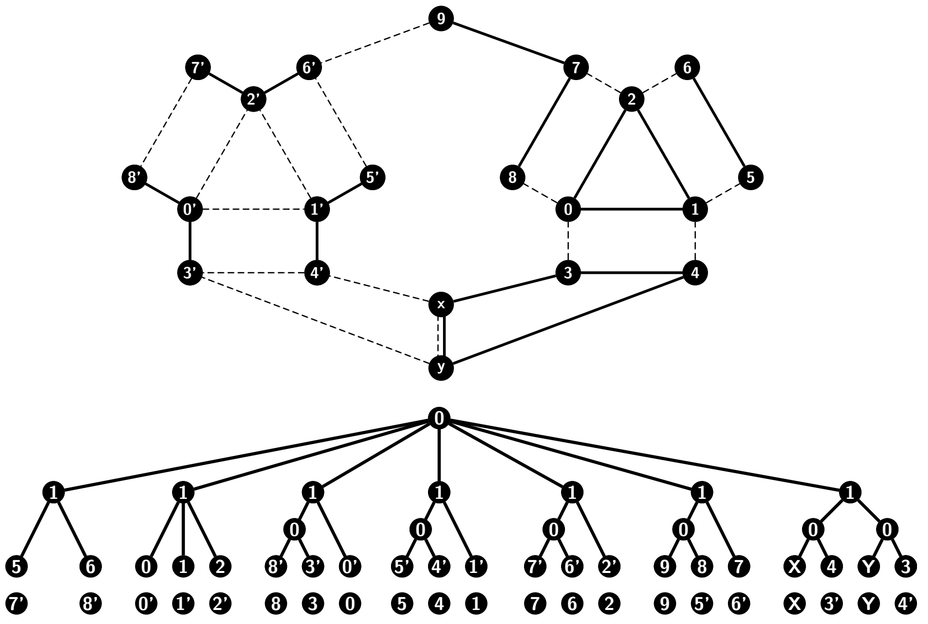

5 6 0 1 2 8' 3' 0' 5' 4' 1' 7' 6' 2' 9 8 7 X 4 Y 3
7' 8' 0' 1' 2' 8 3 0 5 4 1 7 6 2 9 5' 6' X 3' Y 4'

Fig. 2. The shown (non-co)graph G has a 2-decomposition $\Pi = \{E_1, E_2\}$. Edges in the different elements E_1 and E_2 are highlighted by dashed and solid edges, respectively. Thus, two cotrees, shown in the lower part of this picture, are sufficient to represent the structure of G. The two cotrees are isomorphic, and thus, differ only in the arrangement of their leaf sets. For this reason, we only depicted one cotree with two different leaf sets. Note, G has no 2-partition, but a coarsest 3-partition. The latter can easily be verified by application of the construction in Lemma 1.

A graph having an optimal cograph Δ-decomposition is shown in Fig. 1. Thus, the derived bound $\Delta + 1$ is almost sharp. Nevertheless, we assume that this bound can be sharpened:

Conjecture 1. For every graph G with maximum degree Δ there is a cograph Δ-decomposition.

However, there are examples of non-cographs containing many induced P_4's that have a cograph k-decomposition with $k \ll \Delta + 1$, which implies that any optimal k-decomposition of those graphs will have significantly less elements than $\Delta + 1$, see the following examples.

Example 1. Consider the graph $G = (V, E)$ with vertex set $V = \{1, \ldots, k\} \cup \{a, b\}$ and $E = \{[i, j] \mid i, j \in \{1, \ldots, k\}, i \neq j\} \cup \{[k, a], [a, b]\}$. The graph G is not a cograph, since there are induced P_4's of the form $i - k - a - b$, $i \in \{1, \ldots, k - 1\}$. On the other hand, the subgraph $H = (V, E \setminus \{[k, a]\})$ has two connected components, one is isomorphic to the complete graph K_k on k vertices and the other to the complete graph K_2. Hence, H is a cograph. Therefore, G

has a cograph 2-partition $\{E \setminus \{[k,a]\}, \{[k,a]\}\}$, independent from k and thus, independent from the maximum degree $\Delta = k$.

Example 2. Consider the 2n-dimensional hypercube $Q_{2n} = (V,E)$ with maximum degree $2n$. We will show that this hypercube has a coarsest cograph n-partition $\Pi = \{E_1,\ldots,E_n\}$, which implies that for any optimal cograph k-decomposition of Q_{2n} we have $k \leq \Delta/2$.

We construct now a cograph n-partition of Q_{2n}. Note, $Q_{2n} = \Box_{i=1}^{2n} K_2 = \Box_{i=1}^{n}(K_2 \Box K_2) = \Box_{i=1}^{n} Q_2$. In order to avoid ambiguity, we write $\Box_{i=1}^{n} Q_2$ as $\Box_{i=1}^{n} H_i$, $H_i \simeq Q_2$ and assume that Q_2 has edges $[0,1]$, $[1,2]$, $[2,3]$, $[3,0]$. The cograph n-partition of Q_{2n} is defined as $\Pi = \{E_1,\ldots,E_n\}$, where $E_i = \cup_{v\in V} E(H_i^v)$. In other words, the edge set of all H_i-layers in Q_{2n} constitute a single class E_i in the partition for each i. Therefore, the subgraph $G = (V, E_i)$ consists of n connected components, each component is isomorphic to the square Q_2. Hence, $G_i = (V, E_i)$ is a cograph.

Assume for contradiction that $\Pi = \{E_1,\ldots,E_n\}$ is not a coarsest partition. Then there are distinct classes E_i, $i \in I \subseteq \{1,\ldots,n\}$ such that $G_I = (V, \cup_{i\in I} E_i)$ is a cograph. W.l.o.g. assume that $1,2 \in I$ and let $v = (0,\ldots,0) \in V$. Then, the subgraph $H_1^v \cup H_2^v \subseteq Q_{2n}$ contains a path P_4 with edges $[x,v] \in E(H_1^v)$ and $[v,a],[a,b] \in E(H_2^v)$, where x=(1,0,...,0), a=(0,1,0...,0) and $b = (0,2,0\ldots,0)$. By definition of the Cartesian product, there are no edges connecting x with a or b or v with b in Q_{2n} and thus, this path P_4 is induced. As this holds for all subgraphs $H_i^v \cup H_j^v$ ($i,j \in I$ distinct) and thus, in particular for the graph G_I we can conclude that classes of Π cannot be combined. Hence Π is a coarsest cograph n-partition.

Because of the results of computer-aided search for $n-1$-partitions and decompositions of hypercubes Q_{2n} we are led to the following conjecture:

Conjecture 2. Let $k \in \mathbb{N}$ and $k > 1$. Then the $2k$-cube has no cograph $k-1$-decomposition, i.e., the proposed k-partition of the hypercube Q_{2k} in Example 2 is also optimal.

The proof of the latter hypothesis would immediately verify the next conjecture.

Conjecture 3. For every $k \in \mathbb{N}$ there is a graph that has an *optimal* cograph k-decomposition.

Proving the last conjecture appears to be difficult. We wish to point out that there is a close relationship to the problem of finding pattern avoiding words, see e.g. [2,3,5,7,22,23]: Consider a graph $G = (V,E)$ and an ordered list $(e_1,\ldots,e_m)$ of the edges $e_i \in E$. We can associate to this list $(e_1,\ldots,e_m)$ a word $w = (w_1,\ldots,w_m)$. By way of example, assume that we want to find a valid cograph 2-decomposition $\{E_1,E_2\}$ of G and that G contains an induced P_4 consisting of the edges e_i,e_j,e_k. Hence, one has to avoid assignments of the edges e_i,e_j,e_k to the single set E_1, resp., E_2. The latter is equivalent to find a binary word $(w_1,\ldots,w_m)$ such that $(w_i,w_j,w_k) \neq (X,X,X)$, $X \in \{0,1\}$ for each of those induced P_4's. The latter can easily be generalized to find pattern avoiding words over an alphabet $\{1, \ldots, k\}$ to get a valid k-decomposition. However, to the authors knowledge, results concerning the counting of k-ary words avoiding forbidden patterns and thus, verifying if there is any such word (or equivalently a k-decomposition) are basically known for scenarios like: If $(p_1, \ldots p_l) \in \{1, \ldots, k\}^l$ (often $l < 3$), then none of the words w that contain a subword $(w_{i_1}, \ldots, w_{i_l}) = (p_1, \ldots p_l)$ with $i_{j+1} = i_j + 1$ (consecutive letter positions) or $i_j < i_k$ whenever $j < k$ (order-isomorphic letter positions) is allowed. However, such findings are to restrictive to our problem, since we are looking for words, that have only on a few, but fixed positions of non-allowed patterns. Nevertheless, we assume that results concerning the recognition of pattern avoiding words might offer an avenue to solve the latter conjectures.

4.1 NP-completeness and NP-hardness Results

We are now in the position to prove the NP-completeness of Cotree 2-Representation and Cotree 2-Decomposition. These results allow to show that the problem of determining whether there is cograph 2-partition is NP-complete, as well.

We start with two lemmata concerning cograph 2-decompositions of the graphs shown in Fig. 3 and 4.

Lemma 1. *For the literal and extended literal graph in Figure 3 every cograph 2-decomposition is a uniquely determined cograph 2-partition.*

In particular, in every cograph 2-partition $\{E_1, E_2\}$ of the extended literal graph, the edges of the triangle $(0, 1, 2)$ must be entirely contained in one E_i and the pending edge $[6, 9]$ must be in the same edge set E_i as the edges of the of the triangle. Furthermore, the edges $[9, 10]$ and $[9, 11]$ must be contained in E_j, $i \neq j$.

Proof. see Appendix.

Lemma 2. *Given the clause gadget in Fig. 4.*

For any cograph 2-decomposition, all edges of exactly two of the triangles in the underlying three extended literal graphs must be contained in one E_i and not in E_j, while the edges of the triangle of one extended literal graph must be in E_j and not in E_i, $i \neq j$.

Furthermore, for each cograph 2-decomposition exactly two of the edges e, e' of the triangle (a, b, c) must be in one E_i while the other edge f is in E_j but not in E_i, $j \neq i$. The cograph 2-decomposition can be chosen so that in addition $e, e' \notin E_j$, resulting in a cograph 2-partition of the clause gadget.

Proof. see Appendix.

We are now in the position to prove NP-completeness of Cograph 2-Partition by reduction from the following problem.

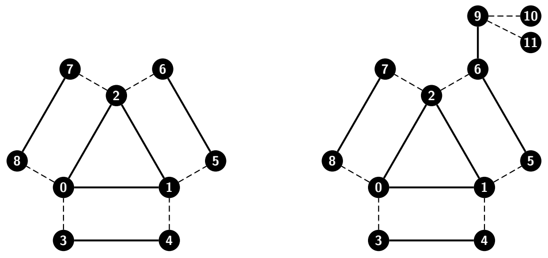


Fig. 3. Left the *literal graph* and right the *extended* literal graph with unique corresponding cograph 2-partition (indicated by dashed and bold-lined edges) is shown

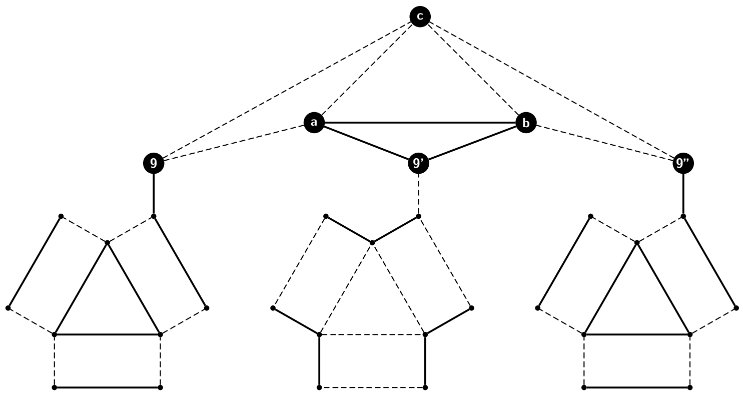


Fig. 4. Shown is a *clause gadget* which consists of a triangle (a, b, c) and three extended literal graphs (as shown in Fig. 3) with edges attached to (a, b, c). A corresponding cograph 2-partition is indicated by dashed and bold-line edges.

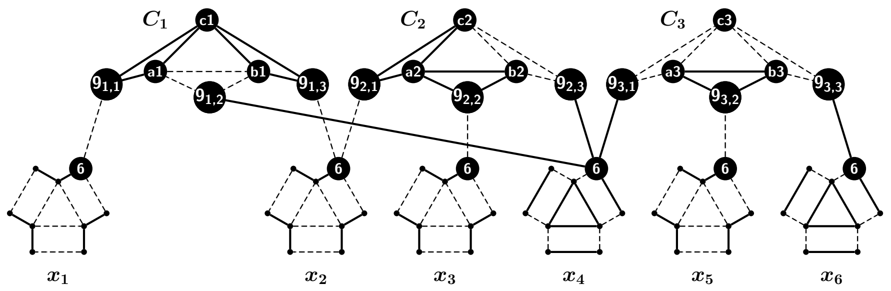


Fig. 5. Shown is the graph Ψ as constructed in the proof of Theorem 6. In particular, Ψ reflects the NAE 3-SAT formula $\psi = \{C_1, C_2, C_3\}$ with clauses $C_1 = (x_1, x_4, x_2), C_2 = (x_2, x_3, x_4)$ and $C_3 = (x_4, x_5, x_6)$. Different literals obtain the same truth assignment true or false, whenever the edges of the triangle in their corresponding literal gadget are contained in the same set E_i of the cograph 2-partition, highlighted by dashed and bold-lined edges.

Problem. MONOTONE NAE 3-SAT
Input: Given a set U of Boolean variables and a set of clauses $\psi = \{C_1, \ldots, C_m\}$ over U such that for all $i = 1, \ldots, m$ it holds that $|C_i| = 3$ and C_i contains no negated variables.
Question: Is there a truth assignment to ψ such that in each C_i not all three literals are set to true?

Theorem 5 ([21,24]). MONOTONE NAE 3-SAT *is NP-complete.*

Theorem 6. COGRAPH 2-DECOMPOSITION, *and thus,* COTREE 2-REPRESENTATION *is NP-complete.*

Proof. see Appendix.

As the proof of Theorem 6 allows us to use cograph 2-partitions in all proof steps, instead of cograph 2-decompositions, we can immediately infer the NP-completeness of the following problem for k=2, as well.

Problem. COGRAPH k-PARTITION
Input: Given a graph $G = (V, E)$ and an integer k.
Question: Is there a Cograph k-Partition of G?

Theorem 7. COGRAPH 2-PARTITION *is NP-complete.*

As a direct consequence of the latter results, we obtain the following theorem.

Theorem 8. *Let G be a given graph that is not a cograph. The following three optimization problems to find the least integer $k > 1$ so that there is a Cograph k-Partition, or a Cograph k-Decomposition, or a Cotree k-Representation for the graph G, are NP-hard.*

Acknowledgments. This work was funded by the German Research Foundation (DFG) (Proj. No. MI439/14-1).

References

1. Achlioptas, D.: The complexity of g-free colourability. Discrete Mathematics **165–166**, 21–30 (1997). Graphs and Combinatorics
2. Bernini, A., Ferrari, L., Pinzani, R.: Enumeration of some classes of words avoiding two generalized patterns of length three. arXiv preprint arXiv:0711.3387 (2007)
3. Bilotta, S., Grazzini, E., Pergola, E., Morgagni, V.G.B.: Counting binary words avoiding alternating patterns. Journal of Integer Sequences **16**(2), 3 (2013)
4. Böcker, S., Dress, A.W.M.: Recovering symbolically dated, rooted trees from symbolic ultrametrics. Adv. Math. **138**, 105–125 (1998)
5. Brändén, P., Mansour, T.: Finite automata and pattern avoidance in words. Journal of Combinatorial Theory, Series A **110**(1), 127–145 (2005)
6. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications. Soc. Ind. Appl. Math., Philadephia (1999)

7. Burstein, A., Mansour, T.: Words restricted by patterns with at most 2 distinct letters. Electron. J. Combin. Number Theory **9**(2), 1–16 (2002)
8. Corneil, D.G., Lerchs, H., Burlingham, L.K.S.: Complement reducible graphs. Discr. Appl. Math. **3**, 163–174 (1981)
9. Corneil, D.G., Perl, Y., Stewart, L.K.: A linear recognition algorithm for cographs. SIAM Journal on Computing **14**(4), 926–934 (1985)
10. Dorbec, P., Montassier, M., Ochem, P.: Vertex partitions of graphs into cographs and stars. Journal of Graph Theory **75**(1), 75–90 (2014)
11. Gimbel, J., Nesětřil, J.: Partitions of graphs into cographs. Electronic Notes in Discrete Mathematics **11**, 705–721 (2002). The Ninth Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms and Applications
12. Hammack, R., Imrich, W., Klavžar, S.: Handbook of Product graphs, 2nd edn. CRC Press, Boca Raton (2011)
13. Hellmuth, M., Hernandez-Rosales, M., Huber, K.T., Moulton, V., Stadler, P.F., Wieseke, N.: Orthology relations, symbolic ultrametrics, and cographs. Journal of Mathematical Biology **66**(1–2), 399–420 (2013)
14. Hellmuth, M., Wiesecke, N., Lenhof, H.P., Middendorf, M., Stadler, P.F.: Phylogenomics with paralogs. PNAS **112**(7), 2058–2063 (2015)
15. Lafond, M., El-Mabrouk, N.: Orthology and paralogy constraints: satisfiability and consistency. BMC Genomics **15**(Suppl. 6), S12 (2014)
16. Lerchs, H.: On cliques and kernels. Technical report, Dept. of Comput. Sci. University of Toronto (1971)
17. Lerchs, H.: On the clique-kernel structure of graphs. Technical report, Dept. of Comput. Sci. University of Toront (1971)
18. Liu, Y., Wang, J., Guo, J., Chen, J.: Cograph editing: complexity and parameterized algorithms. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 110–121. Springer, Heidelberg (2011)
19. Liu, Y., Wang, J., Guo, J., Chen, J.: Complexity and parameterized algorithms for cograph editing. Theoretical Computer Science **461**, 45–54 (2012)
20. Misra, J., Gries, D.: A constructive proof of vizing's theorem. Information Processing Letters **41**(3), 131–133 (1992)
21. Moret, B.M.: The Theory of Computation. Addison-Wesley (1997)
22. Pudwell, L.K.: Enumeration schemes for pattern-avoiding words and permutations. ProQuest (2008)
23. Pudwell, L.K.: Enumeration schemes for words avoiding patterns with repeated letters. Electron. J. Combin. Number Theory **8**(A40), 1–19 (2008)
24. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226. ACM, New York (1978)
25. Vizing, V.G.: On an estimate of the chromatic class of a p-graph. Journal of Mathematical Biology **3**, 23–30 (1964). (Russian)
26. Zhang, P.: A study on generalized solution concepts in constraint satisfaction and graph colouring. Master's thesis, University of British Columbia, Canada (2014)

Appendix

Proof of Lemma 1: It is easy to verify that the given cograph 2-partition $\{E_1, E_2\}$ in Fig. 3 fulfills the conditions and is correct, since $G = (V, E_1)$ and $G = (V, E_2)$ do not contain induced P_4's and are, thus, cographs. We have to show that it is also unique.

Assume that there is another cograph 2-decomposition $\{F_1, F_2\}$. Note, for any cograph 2-decomposition $\{F_1, F_2\}$ it must hold that two incident edges in the triangle $(0, 1, 2)$ are contained in one of the sets F_1 or F_2. W.l.o.g. assume that $[0, 1], [0, 2] \in F_1$.

Assume first that $[1, 2] \notin F_1$. In this case, because of the paths $P_4 = 6 - 2 - 0 - 1$ and $P_4 = 2 - 0 - 1 - 5$ it must hold that $[2, 6], [1, 5] \notin F_1$ and thus, $[2, 6], [1, 5] \in F_2$. However, in this case and due to the paths $P_4 = 6 - 2 - 1 - 4$ and $2 - 0 - 1 - 4$ the edge $[1, 4]$ can neither be contained in F_1 nor in F_2, a contradiction. Hence, $[1, 2] \in F_1$.

Note, the square S_{1256} induced by vertices $1, 2, 5, 6$ cannot have all edges in F_1, as otherwise the subgraph (V, F_1) would contain the induced $P_4 = 6{-}5{-}1{-}0$. Assume that $[1, 5] \in F_1$. As not all edges S_{1256} are contained in F_1, at least one of the edges $[5, 6]$ and $[2, 6]$ must be contained in F_2. If only one of the edges $[5, 6]$, resp., $[2, 6]$ is contained in F_2, we immediately obtain the induced $P_4 = 6{-}2{-}1{-}5$, resp., $6{-}5{-}1{-}2$ in (V, F_1) and therefore, both edges $[5, 6]$ and $[2, 6]$ must be contained in F_2. But then the edge $[2, 7]$ can neither be contained in F_1 (due to the induced $P_4 = 5 - 1 - 2 - 7$) nor in F_2 (due to the induced $P_4 = 5 - 6 - 2 - 7$), a contradiction. Hence, $[1, 5] \notin F_1$ and thus, $[1, 5] \in F_2$ for any 2-decomposition. By analogous arguments and due to symmetry, all edges $[0, 3]$, $[0, 8]$, $[1, 4]$, $[2, 6]$, $[2, 7]$ are contained in F_2, but not in F_1.

Moreover, due to the induced $P_4 = 7 - 2 - 6 - 5$ and since $[2, 6], [2, 7] \in F_2$, the edge $[5, 6]$ must be in F_1 and not in F_2. By analogous arguments and due to symmetry, it holds that $[3, 4], [7, 8] \in F_1$ and $[3, 4], [7, 8] \notin F_2$. Finally, none of the edges of the triangle $(0, 1, 2)$ can be contained in F_2, as otherwise, we obtain an induced P_4 in (V, F_2). Taken together, any 2-decomposition of the literal graph must be a partition and is unique.

Consider now the extended literal graph in Figure 3. As this graph contains the literal graph as induced subgraph, the unique 2-partition of the underlying literal graph is determined as by the preceding construction. Due to the path $P_4 = 7 - 2 - 6 - 9$ with $[2, 6], [2, 7] \in F_2$ we can conclude that $[6, 9] \notin F_2$ and thus $[6, 9] \in F_1$. Since there are induced paths $P_4 = 5 - 6 - 9 - y$, $y = 10, 11$ with $[5, 6], [6, 9] \in F_1$ we obtain that $[9, 10], [9, 11] \notin F_1$ and thus, $[9, 10], [9, 11] \in F_2$ for any 2-decomposition (which is in fact a 2-partition) of the extended literal graph, as claimed. □

Proof of Lemma 2: It is easy to verify that the given cograph 2-partition in Fig. 4 fulfills the conditions and is correct, as $G = (V, E_1)$ and $G = (V, E_2)$ are cographs.

As the clause gadget contains the literal graph as induced subgraph, the unique 2-partition of the underlying literal graph is determined as by the construction given in Lemma 1. Thus, each edge of the triangle in each underlying literal graph is contained in either one of the sets E_1 or E_2. Assume that edges of the triangles in the three literal gadgets are *all* contained in the same set, say E_1. Then, Lemma 1 implies that $[9, a], [9, c], [9', a], [9', b], [9'', b], [9'', c] \in E_1$ and none of them is contained in E_2. Since there are induced P_4's: $9 - a - b - 9''$, $9' - a - c - 9''$ and $9 - c - b - 9'$, the edges $[a, b], [a, c], [b, c]$ cannot be contained in E_1, and thus must be in E_2. However, this is not possible, since then we would have the induced paths $P_4 = 9 - a - 9' - b$ in the subgraph (V, E_1) a contradiction. Thus, the edges of the triangle of exactly one literal gadget must be contained in a different set E_i than the edges of the other triangles in the other two literal gadgets. W.l.o.g. assume that the 2-decomposition of the underlying literal gadgets is given as in Fig. 4. and identify bold-lined edges with E_1 and dashed edges with E_2.

It remains to show that this 2-decomposition of the underlying three literal gadgets determines which of the edges of triangle (a, b, c) are contained in which of the sets E_1 and E_2. Due to the induced path $9 - a - b - 9''$ and since $[9, a], [9'', b] \in E_2$, the edge $[a, b]$ cannot be contained in E_2 and thus, is contained in E_1. Moreover, if $[b, c] \notin E_2$, then then there is an induced path $P_4 = b - 9'' - c - 9$ in the subgraph (V, E_2), a contradiction. Hence, $[b, c] \in E_2$ and by analogous arguments, $[a, c] \in E_2$. If $[b, c] \notin E_1$ and $[a, c] \notin E_1$, then we obtain a cograph 2-partition. However, it can easily be verified that there is still a degree of freedom and $[a, c], [b, c] \in E_1$ is allowed for a valid cograph 2-decomposition. $\square$

Proof of Theorem 6: Given a graph $G = (V, E)$ and cograph 2-decomposition $\{E_1, E_2\}$, one can verify in linear time whether (V, E_i) is a cograph [9]. Hence, Cograph 2-Partition $\in$ NP.

We will show by reduction from Monotone NAE 3-SAT that Cograph 2-Decomposition is NP-hard. Let $\psi = (C_1, \ldots, C_m)$ be an arbitrary instance of Monotone NAE 3-SAT. Each clause C_i is identified with a triangle (a_i, b_i, c_i). Each variable x_j is identified with a literal graph as shown in Fig. 3 (left) and different variables are identified with different literal graphs. Let $C_i = (x_{i_1}, x_{i_2}, x_{i_3})$ and G_{i_1}, G_{i_2} and G_{i_3} the respective literal graphs. Then, we extend each literal graph G_{i_j} by adding an edge $[6, 9_{i,j}]$. Moreover, we add to G_{i_1} the edges $[9_{i,1}, a_i], [9_{i,1}, c_i]$, to G_{i_2} the edges $[9_{i,2}, a_i], [9_{i,2}, b_i]$, to G_{i_3} the edges $[9_{i,3}, c_i], [9_{i,3}, b_i]$. The latter construction connects each literal graph with the triangle (a_i, b_i, c_i) of the respective clause C_i in a unique way, see Fig. 4. We denote the clause gadgets by Ψ_i for each clause C_i. We repeat this construction for all clauses C_i of ψ resulting in the graph Ψ. An illustrative example is given in Fig. 5. Clearly, this reduction can be done in polynomial time in the number m of clauses.

We will show in the following that Ψ has a cograph 2-decomposition (resp., a cograph 2-partition) if and only if ψ has a truth assignment f.

Let $\psi = (C_1, \ldots, C_m)$ have a truth assignment. Then in each clause C_i at least one of the literals $x_{i_1}, x_{i_2}, x_{i_3}$ is set to true and one to false. We assign all edges e of the triangle in the corresponding literal graph G_{i_j} to E_1, if $f(x_{i_j}) =$ *true* and to E_2, otherwise. Hence, each edge of exactly two of the triangles (one in G_{i_j} and one in $G_{i_{j'}}$ contained in one E_r and not in E_s, while the edges of the other triangle in $G_{i_{j''}}$, $j'' \neq j, j'$ are contained in E_s and not in E_r, $r \neq s$, as needed for a possible valid cograph 2-decomposition (Lemma 2). We now apply the construction of a valid 2-decomposition (or 2-partition) for each Ψ_i as given in Lemma 2, starting with the just created assignment of edges contained in the triangles in G_{i_j}, $G_{i_{j'}}$ and $G_{i_{j''}}$ to E_1 or E_2. In this way, we obtain a valid 2-decomposition (or 2-partition) for each subgraph Ψ_i of Ψ. Thus, if there would be an induced P_4 in Ψ with all edges belonging to the same set E_r, then this P_4 can only have edges belonging to different clause gadgets Ψ_k, Ψ_l. By construction, such a P_4 can only exist along different clause gadgets Ψ_k and Ψ_l only if C_k and C_l have a literal $x_i = x_{k_m} = x_{l_n}$ in common. In this case, Lemma 2 implies that the edges $[6, 9_{k,m}]$ and $[6, 9_{l,n}]$ in Ψ_i must belong to the same set E_r. Again by Lemma 2, the edges $[9_{k,m}, y]$ and $[9_{k,m}, y']$, $y, y' \in \{a_k, b_k, c_k\}$ as well as the edges $[9_{l,n}, y]$ and $[9_{l,n}, y']$, $y, y' \in \{a_l, b_l, c_l\}$ must be in a different set E_s than $[6, 9_{k,m}]$ and $[6, 9_{l,n}]$. Moreover, respective edges $[5, 6]$ in Ψ_k, as well as in Ψ_l (Fig. 3) must then be in E_r, i.e., in the same set as $[6, 9_{k,m}]$ and $[6, 9_{l,n}]$. However, in none of the cases it is possible to find an induced P_4 with all edges in the same set E_r or E_s along different clause gadgets. Hence, we obtain a valid cograph 2-decomposition, resp., cograph 2-partition of Ψ.

Now assume that Ψ has a valid cograph 2-decomposition (or a 2-partition). Any variable x_{i_j} contained in some clause $C_i = (x_{i_1}, x_{i_2}, x_{i_3})$ is identified with a literal graph G_{i_j}. Each clause C_i is, by construction, identified with exactly three literal graphs $G_{i_1}, G_{i_2}, G_{i_3}$, resulting in the clause gadget Ψ_i. Each literal graph G_{i_j} contains exactly one triangle t_j. Since Ψ_i is an induced subgraph of Ψ, we can apply Lemma 2 and conclude that for any cograph 2-decomposition (resp., 2-partition) all edges of exactly two of three triangles t_1, t_2, t_3 are contained in one set E_r, but not in E_s, and all edges of the other triangle are contained in E_s, but not in E_r, $s \neq r$. Based on these triangles we define a truth assignment f to the corresponding literals: w.l.o.g. we set $f(x_i) =$*true* if the edge $e \in t_i$ is contained in E_1 and $f(x_i) =$*false* otherwise. By the latter arguments and Lemma 2, we can conclude that, given a valid cograph 2-partitioning, the so defined truth assignment f is a valid truth assignment of the Boolean formula ψ, since no three different literals in one clause obtain the same assignment and at least one of the variables is set to *true*. Thus, Cograph 2-Decomposition is NP-complete

Finally, because Cograph 2-Decomposition and Cotree 2-Representation are equivalent problems, the NP-completeness of Cotree 2-Representation follows. □

Bounds for the Super Extra Edge Connectivity of Graphs

Chia-Wen Cheng and Sun-Yuan Hsieh(✉)

Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan 701, Taiwan
{p78981037,hsiehsy}@mail.ncku.edu.tw

Abstract. Let G be a connected graph, S be a subset of edges in G, and k be a positive integer. If $G - S$ is disconnected and every component has at least k vertices, then S is a k-extra edge-cut of G. The k-extra edge-connectivity, denoted by $\lambda_k(G)$, is the minimum cardinality over all k-extra edge-cuts of G. If $\lambda_k(G)$ exists and at least one component of $G - S$ contains exactly k vertices for any minimum k-extra edge-cut S, then G is super-λ_k. Moreover, when G is super-λ_k, the persistence of G, denoted by $\rho_k(G)$, is the maximum integer m for which $G - F$ is still super-λ_k for any set $F \subseteq E(G)$ with $|F| \leq m$. It has been shown that the bounds of $\rho_k(G)$ when $k \in \{1, 2\}$. This study shows the bounds of $\rho_k(G)$ when $k \geq 3$.

Keywords: Extra edge-connectivity · Fault tolerance · Super extra edge connectivity

1 Introduction

The edge connectivity is an important measurement for reliability and fault tolerance of networks. Given two vertex subsets $S, T \subseteq V(G)$, notation $[S, T]$ is used to denote the set of edges having one endpoint in S and the other in T. An *edge-cut* is an edge set of the form $[S, \overline{S}]$, where S is a nonempty proper subset of $V(G)$ and $\overline{S}$ denotes $V(G) - S$. A classic measure for the fault tolerance and reliability of a communication network is the *edge connectivity*, denoted by $\lambda(G)$, which is the minimum cardinality of edge-cut, where G is underlying network. Obviously, the remaining graph will be connected when the number of edges deleted is less than $\lambda(G)$. Hence, the larger $\lambda(G)$ is, the more reliable the network is. Numerous studies [2,4,5,8,12,13,15,18] discussed the special edge-cut such that the remaining graph still satisfies some conditions when the edge-cut is removed. One issue discussed each component has enough vertices in the remaining graph. A *k-extra edge-cut*, where $k \geq 1$ is an integer, is an edge-cut such that each component of the remaining graph has at least k vertices when the edge-cut has removed. Fàbrega and Fiol [4,5] generalized the concept of the edge-connectivity to the k-extra edge-connectivity, denoted by $\lambda_k(G)$, is the minimum cardinality of a k-extra edge-cut for a graph. Note that the larger $\lambda_k(G)$ is, the more reliable the network is [9,10,14].

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 624–631, 2015.
DOI: 10.1007/978-3-319-21398-9_49

A k-extra edge-connected graph is further said to be *super k-extra edge-connected* (super-λ_k for short) if the remaining graph contains at least one component has exactly k vertices for each minimum k-extra edge-cut being removed. Recently, the sufficient conditions for graphs to be super-λ_k were discussed in [12,13,15].

Link (edge) faults may occur when a network is activated, so it is important to consider faulty networks. In this paper, we study the edge fault tolerance of graphs with respect to super-λ_k properties. In the other words, we aim at determining how many faulty edges can be tolerated such that the remaining graph is still super-λ_k. Hong et al. [6] showed the bounds when $k = 1$, and Hong and Xu [7] showed the bounds when $k = 2$. In this paper, we propose the bounds for $k \geq 3$.

The remainder of this paper is organized as follows. Section 2 provides some definitions and notations. Section 3 shows the edge fault tolerance of graphs with respect to super-λ_3 graph. Section 4 shows the edge fault tolerance of graphs with respect to super-λ_k graph when $k \geq 4$. Section 5 provides concluding remarks.

2 Preliminaries

A *graph* $G = (V, E)$ is a pair of the *vertex set* V and the *edge set* E, where V is a finite set and E is a subset of $\{(u, v)|\ (u, v)$ is an unordered pair of $V\}$. We also use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G, respectively. Let $n(G) = |V(G)|$ be the *order* of G. Two vertices u and v are *adjacent* if (u, v) is an edge in G. We also say that the edge (u, v) *incident to* u and v, and u and v are the *endpoints* of (u, v). For the endpoints of an edge, one is a *neighbor* of the other. For a vertex v in G, we use $N_G(v)$ to denote the neighbors of v. The *degree* of vertex v, denoted by $d_G(v)$, is the number of edges incident to it. Let $\delta(G) = min\{d_G(v)|\ v \in V(G)\}$. A *path* $\langle v_1, v_2, \ldots, v_k\rangle$ is a sequence of distinct vertices such that any two consecutive vertices are adjacent. Vertices v_1 and v_k are the *endpoints* of the path. A *complete graph* is a simple graph whose vertices are pairwise adjacent; the (unlabeled) complete graph with n vertices is denoted K_n.

An *isomorphism* from a simple graph G to a simple graph H is a one-to-one and onto function $\pi : V(G) \to V(H)$ such that $(u, v) \in E(G)$ if and only if $(\pi(u), \pi(v)) \in E(H)$. We say that "$G$ is *isomorphic to* H", written $G \cong H$, if there is an isomorphism from G to H.

Let $k \geq 1$ be an integer. A edge set $S \subseteq E(G)$ is a *k-extra edge-cut* if $G - S$ is disconnected and each component has at least k vertices. The *k-extra edge connectivity* of G, denoted by $\lambda_k(G)$, is defined as the minimum cardinality over all k-extra edge-cuts of G. If $\lambda_k(G)$ exists, then G is said to be λ_k-connected and $\lambda(G) = \lambda_1(G) \leq \lambda_2(G) \leq \lambda_3(G) \leq \cdots \leq \lambda_k(G)$ obviously. The following lemma shows that the necessary and sufficient condition for a graph to be λ_k-connected.

Lemma 1. [11] *Let $k \geq 1$ be an integer and G be a connected graph. If G has order at least $3k - 2$, then G is λ_k-connected graph if and only if G contains no such vertex u that every component of $G - \{u\}$ has order at most $k - 1$.*

Let $X \subseteq V(G)$ and $\overline{X} = V(G) - X$, use $[X, \overline{X}]$ to denote the set of edges between X and $\overline{X}$ in G and $\omega_G(X) = |[X, \overline{X}]|$. Notation $G[X]$ is used to denote the subgraph of G induced by X. Let $\xi_k(G) = \min\{\omega_G(X) : X \subseteq V(G), |X| = k,$ and $G[X]$ is connected$\}$. It has been shown that $\lambda_k(G) \leq \xi_k(G)$ holds for any λ_k-connected graph, where $k \geq 1$ [1,3,10].

Let K_t be a complete graph with t vertices, and let $G_1, G_2, \ldots, G_m$ be m copies of K_t, where $m \geq 1$ and $t \geq 1$. Let v be a new vertex such that v is adjacent to every vertex in $\cup_{i=1}^{m} V(G_i)$. Then, the resulting graph is denoted by $G^*_{m,t}$. The following lemma shows a sufficient condition for a graph G to be λ_k-connected and $\lambda_k(G) \leq \xi_k(G)$ for $1 \leq k \leq \delta(G) + 1$.

Lemma 2. [17] *Let G be a connected graph with order at least $2(\delta(G)+1)$. If G is not isomorphic to $G^*_{m,\delta(G)}$ for any positive integer m, then $\lambda_k(G)$ exists and $\lambda_k(G) \leq \xi_k(G)$ for any k with $1 \leq k \leq \delta(G) + 1$.*

A graph G is said to be λ_k-optimal if it satisfies $\lambda_k(G) = \xi_k(G)$. Some properties of λ_k-optimal graphs were investigated in [16]. Moreover, if $\lambda_k(G)$ exists and at least one component of $G - S$ contains exactly k vertices for any minimum k-extra edge-cut S, then G is said to be super-λ_k. Obviously, $\lambda_k(G) = \xi_k(G)$ if G is super-λ_k. The following lemma show that the necessary and sufficient condition for a λ_k-connected graph to be super-λ_k.

Lemma 3. *Let G be a λ_k-connected graph with $\lambda_k(G) \leq \xi_k(G)$ for some $k \geq 1$. Then G is super-λ_k if and only if G is not λ_{k+1}-connected or $\omega_G(X) > \xi_k(G)$ holds for any vertex set $X \subseteq V(G)$ with $k + 1 \leq |X| \leq \lfloor n(G)/2 \rfloor$ and $G[X]$, $G[\overline{X}]$ being connected.*

Proof. Because $\omega_G(X) > \xi_k(G)$ holds for any vertex set $X \subseteq V(G)$ with $k+1 \leq |X| \leq \lfloor |V(G)|/2 \rfloor$ and $G[X]$ and $G[\overline{X}]$ being connected, this implies $\lambda_{k+1}(G) > \xi_k(G)$. By Lemma 1.4 in [7], we know that G is super-λ_k if and only if G is not λ_{k+1}-connected or $\lambda_{k+1}(G) > \xi_k(G)$ for any $k \geq 1$. Therefore, the result holds. □

Let $\eta_k(G)$ denote the number of edge-disjoint connected subgraphs with order k such that each subgraph H of them satisfies $\omega_G(V(H)) = \xi_k(G)$. For example, consider a graph G shown in Fig. 1. Because vertices a and b are only two vertices such that each of them has minimum degree in this graph, $\eta_1(G) = 2$. Moreover, because $\omega_G(\{(a,b)\}) = \xi_2(G) = 4$ and each of other edges e does not satisfy the condition (i.e., $\omega_G(\{\text{the endpoints of } e\}) \neq \xi_2(G) = 4$), $\eta_2(G) = 1$. The equality $\eta_3(G) = 2$ holds because we can find two edge disjoint paths $\langle a, c, b\rangle$ and $\langle a, d, b\rangle$ such that $\omega_G(\{a, c, b\}) = \omega_G(\{a, d, b\}) = \xi_3(G) = 4$. Finally, the equality $\eta_4(G) = 2$ holds because we can find two edge disjoint paths $\langle a, c, d, b\rangle$ and $\langle d, a, b, c\rangle$ such that $\omega_G(\{a, c, d, b\}) = \xi_4(G) = 2$.

Definition 1. Let $k \geq 1$ be an integer. The *persistence* of the super-λ_k graph G, denoted by $\rho_k(G)$, is the maximum integer m for which $G - F$ is still super-λ_k for any set $F \subseteq E(G)$ with $|F| \leq m$.

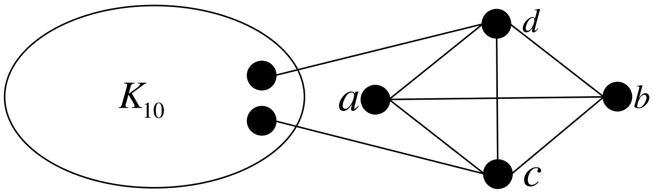


Fig. 1. Illustration of $\eta_k(G)$ for $k = 1, 2, 3$

Hong et al.[6] showed that $\min\{\lambda_2(G)-\delta(G)-1, \delta(G)-1\} \leq \rho_1(G) \leq \delta(G)-1$ for any super-λ_1 and λ_2-connected graph G. In addition, Hong and Xu [7] also showed that $\min\{\lambda_3(G) - \xi_2(G) - 1, \delta(G) - 1\} \leq \rho_2(G) \leq \delta(G) - 1$ for any super-λ_2 and λ_3-connected graph G with $\eta_2(G) \geq \delta(G)$. This study showed the following bounds of $\rho_k(G)$ for $k \geq 3$:

1. $\rho_k(G) = \lambda_1(G) - 1$ if graph G is super-λ_k, but not λ_{k+1}-connected,
2. $\min\{\lambda_4(G) - \xi_3(G) - 1, \eta_3(G) - 1, \lambda_1(G) - 1\} \leq \rho_3(G) \leq \lambda_1(G) - 1$ for any super-λ_3 and λ_4-connected graph G, and
3. $\min\{\lambda_{k+1}(G) - \xi_k(G) - 1, \eta_k(G) - 1, \delta(G) - k + 1, \lambda_1(G) - 1\} \leq \rho_k(G) \leq \lambda_1(G) - 1$ for any super-λ_k and λ_{k+1}-connected graph G, where $k \geq 4$.

3 Bounds on the Persistence of Super-$\boldsymbol{\lambda_3}$ Graphs

If a graph $G = (V, E)$ has a path with the endpoints u and v, then the *distance* between u and v, denoted by $d_G(u, v)$ or simply $d(u, v)$, is the least length of a path between u and v. If G has no such path, then $d(u, v) = \infty$. The *eccentricity* of a vertex u, denoted by $\epsilon(u)$, is $\max_{v \in V} d(u, v)$.

Let H_1 be a connected graph with order at least six, which satisfies: (a) H_1 contains no cycles of length greater than three and (b) there exists exactly one vertex $v_0 \in V(H_1)$ with degree greater than two, and v_0 has excentricity equal or less than two. Fig. 2 show that the graph H_1 and another simple graph H_2. The following lemma show that the necessary and sufficient condition for a graph to be λ_3-connected.

Lemma 4. [1] *A connected graph with order six is not λ_3-connected if and only if G is isomorphic to H_1 or H_2 (see Fig. 2). Furthermore, if G is λ_3-connected, then $\lambda_3(G) \leq \xi_3(G)$.*

According to Lemma 4, we can find the edge fault tolerance of graph with respect to λ_3-connected.

Lemma 5. *Let G be a λ_3-connected graph with $\delta(G) \geq 3$. Then $G - F$ is λ_3-connected graph and $\lambda_3(G - F) \leq \xi_3(G - F)$ for any $F \subseteq E(G)$ with $|F| \leq \lambda_1(G) - 1$.*

Proof. Assume that $G - F$ is not λ_3-connected. By Lemma 4, $G - F \cong H_1$ or $G - F \cong H_2$. There are the following scenarios.

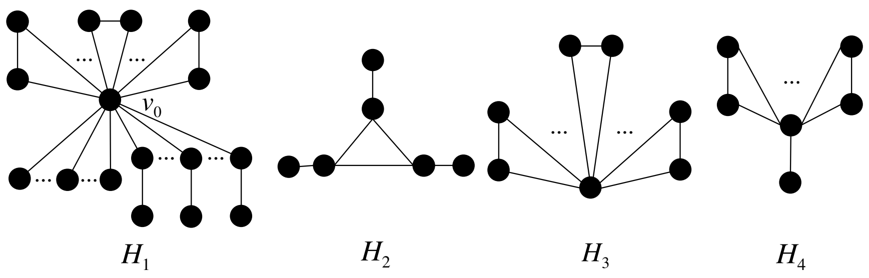


Fig. 2. Illustration of Lemma 4 and Lemma 5

Case 1: $\delta(G-F) \geq 2$. In this case, $G-F \cong H_3$ (Fig. 2) and there are at least six vertices with degree two in $G-F$. Hence,

$$|F| = |E(G)| - |E(G-F)| \geq \frac{(6\delta(G) - 6\cdot 2)}{2} = 3\delta(G) - 6 \geq \delta(G) \geq \lambda_1(G),$$

which leads to a contradiction.

Case 2: $\delta(G-F) = 1$. There is exactly one vertex in $G-F$ such that its degree is one, and all edges of F are incident to this vertex. Hence, $G-F$ is not isomorphic to H_2. By Lemma 4, $G-F \cong H_4$ (Fig. 2), there exists at least four vertices with degree two. Note that all edges of F are incident to the vertex with degree one, hence

$$|F| = |E(G)| - |E(G-F)| \geq 4\delta(G) - 4\cdot 2 \geq \delta(G) + 1 \geq \lambda_1(G),$$

which leads to a contradiction.

Combining the above cases completes the proof. □

Next, we show that the bounds of $\rho_3(G)$ for the super-λ_3 graph G in Theorem 1.

Theorem 1. *Let G be a super-λ_3 graph with $\delta(G) \geq 3$. Then the following statements hold: (a) If G is not λ_4-connected, then $\rho_3(G) = \lambda_1(G) - 1$. (b) If G is λ_4-connected, then* $\min\{\lambda_4(G) - \xi_3(G) - 1, \eta_3(G) - 1, \lambda_1(G) - 1\} \leq \rho_3(G) \leq \lambda_1(G) - 1$.

Proof. There exists one edge set F with size $\lambda_1(G)$ such that $G-F$ is disconnected. Then, $G-F$ is not super-λ_3, which implies that $\rho_3(G) \leq |F| - 1 = \lambda_1(G) - 1$. Next, we find the lower bound of $\rho_3(G)$ according to G is λ_4-connected or not.

(a) To prove that $\rho_3(G) \geq \lambda_1(G) - 1$, it suffices to show that for any $F \subseteq E(G)$ with $|F| \leq \lambda_1(G) - 1$, $G-F$ is super-λ_3. By Lemma 5, $G-F$ is λ_3-connected with $\lambda_3(G-F) \leq \xi_3(G-F)$. Because G is not λ_4-connected, $G-F$ is also not λ_4-connected. By Lemma 3, $G-F$ is super-λ_3 and (a) is proved.

(b) Let $m = \min\{\lambda_4(G) - \xi_3(G) - 1, \eta_3(G) - 1, \lambda_1(G) - 1\}$. To prove that $\rho_3(G) \geq m$, it suffices to show that for any $F \subseteq E(G)$ with $|F| \leq m$, $G' = G-F$

is super-λ_3. Note that $|F| \leq m \leq \lambda_1(G)-1$. By Lemma 5, $G-F$ is λ_3-connected with $\lambda_3(G-F) \leq \xi_3(G-F)$. If $G-F$ is not λ_4-connected, then by Lemma 3, $G-F$ is super-λ_3 and (b) is proved. If $G-F$ is λ_4-connected, then let X be any subset of $V(G')$ with $|X| \geq 4$ and $|\overline{X}| \geq 4$ such that $G'[X]$ and $G'[\overline{X}]$ are connected. Hence,

$$\omega_{G-F}(X) \geq \omega_G(X) - |F| \geq \lambda_4(G) - (\lambda_4(G) - \xi_3(G) - 1) = \xi_3(G) + 1. \quad (1)$$

Because $|F| \leq m \leq \eta_3(G) - 1$, there exists one subgraph A with order three such that $\omega_G(A) = \xi_3(G)$ and $A - F$ is still connected. In the other words,

$$\xi_3(G) = \omega_G(A) \geq \omega_{G-F}(A) \geq \xi_3(G-F). \quad (2)$$

According to equation (1) and (2), $\omega_{G-F}(X) > \xi_3(G-F)$. By Lemma 3, $G-F$ is super-λ_3 and (b) is proved. □

4 Bounds on the Persistence of Super-λ_k Graphs for $k \geq 4$

First, we find the edge fault tolerance of graph with respect to λ_k-connected for the positive integer $k \geq 2$.

Lemma 6. *Let G be a λ_k-connected graph with $2 \leq k \leq \delta(G)$, and $n(G) \geq 3\delta(G) - 2$. Then $G-F$ is λ_k-connected graph for any $F \subseteq E(G)$ with $|F| \leq \lambda_1(G) - 1$.*

Proof. Obviously, $G-F$ is connected because $|F| \leq \lambda_1(G) - 1$. If $G-F$ is not λ_k-connected, by Lemma 1, then there exists $n(G) - 1$ vertices with degree at most $k-1$ in $G-F$. Since $\delta(G) \geq k$, there are at least one edge of F incident to each of the above $n(G) - 1$ vertices. Hence,

$$|F| \geq \lceil (n(G) - 1)/2 \rceil \geq \lceil (3\delta(G) - 3)/2 \rceil > \delta(G) - 1 \geq \lambda_1(G) - 1,$$

which leads to a contradiction. Therefore, $G-F$ is λ_k-connected graph. □

Now, let G be a super-λ_k graph, but not λ_{k+1}-connected for $k \geq 4$. We show that the bounds of $\rho_k(G)$ for $k \geq 4$ in Theorem 2.

Theorem 2. *Let G be a super-λ_k graph with $4 \leq k \leq \delta(G)$, and $n(G) \geq 3\delta(G) - 2$. If G is not λ_{k+1}-connected, then $\rho_k(G) = \lambda_1(G) - 1$.*

Proof. There exists one edge set F with size $\lambda_1(G)$ such that $G-F$ is disconnected. Then $G-F$ is not super-λ_k. We have $\rho_k(G) \leq |F| - 1 = \lambda_1(G) - 1$.

To prove that $\rho_k(G) \geq \lambda_1(G) - 1$, it suffices to show that for any $F \subseteq E(G)$ with $|F| \leq \lambda_1(G) - 1$, $G-F$ is super-λ_k. By Lemma 6, $G-F$ is λ_k-connected. Since G is not λ_{k+1}-connected, $G-F$ is also not λ_{k+1}-connected. Hence, every minimum k-extra edge-cut of $G-F$ isolates at least one connected subgraph of order k. Therefore, $G-F$ is super-λ_k and this theorem is proved. □

Next, we try to find out the bounds of $\rho_k(G)$ for super-λ_k and λ_{k+1}-connected graph G.

Lemma 7. *Let G be a λ_k-connected graph with $\lambda_k(G) \leq \xi_k(G)$, $k \leq \delta(G)+1$, and $n(G) \geq 2\delta(G)+2$. Then $G-F$ is λ_k-connected graph and $\lambda_k(G-F) \leq \xi_k(G-F)$ for any $F \subseteq E(G)$ with $|F| \leq \min\{\delta(G)-k+1, \lambda_1(G)-1\}$.*

Proof. Since $|F| \leq \lambda_1(G)-1$, $G-F$ is connected. If $G-F$ is not isomorphic to $G^*_{m,k-1}$ for any positive integer m, by Lemma 2, then $G-F$ is λ_k-connected and $\lambda_k(G-F) \leq \xi_k(G-F)$ because $n(G) \geq 2\delta(G)+2 \geq 2\delta(G-F)+2$.

Hence, we only consider the case when $G-F$ is isomorphic to $G^*_{m,k-1}$ for some positive integer m. Note that $|F| \leq \delta(G)-k+1$, $k-1 = \delta(G-F) \leq \delta(G)$, and there exists $n(G)-1$ vertices with degree $k-1$ in $G-F$. There are the following scenarios.

Case 1: $\delta(G) \geq k$. In this case, $\delta(G)-k+1 \geq 1$. Hence, there are at least $\delta(G)-k+1$ edges of F incident to each of the $n(G)-1$ vertices with degree $k-1$ in $G-F$. Moreover, $|F| > \delta(G)-k+1$ because $n(G)-1 \geq 3$, which leads to a contradiction.

Case 2: $\delta(G) = k-1$. In this case, $|F| = \delta(G)-k+1 = 0$ and $G-F = G$. Obviously, $G-F$ is λ_k-connected graph and $\lambda_k(G-F) \leq \xi_k(G-F)$. □

Theorem 3. *Let G be a super-λ_k graph with $k \geq 4$ and $n(G) \geq 2\delta(G)+2$. If G is λ_{k+1}-connected, then* $\min\{\lambda_{k+1}(G)-\xi_k(G)-1, \eta_k(G)-1, \delta(G)-k+1, \lambda_1(G)-1\} \leq \rho_k(G) \leq \lambda_1(G)-1$.

Proof. Let $m = \min\{\lambda_{k+1}(G)-\xi_k(G)-1, \eta_k(G)-1, \delta(G)-k+1, \lambda_1(G)-1\}$ and $|F| \leq m$. Then, let X be any subset of $V(G')$ with $|X| \geq k+1$ and $|\overline{X}| \geq k+1$ such that $G'[X]$ and $G'[\overline{X}]$ are connected if $G-F$ is λ_{k+1}-connected.

According to the same analysis of Theorem 1, we know that

$$\omega_{G-F}(X) \geq \omega_G(X) - |F| \geq \lambda_{k+1}(G) - (\lambda_{k+1}(G)-\xi_k(G)-1) = \xi_k(G)+1 \quad (3)$$

and

$$\xi_k(G) = \omega_G(A) \geq \omega_{G-F}(A) \geq \xi_k(G-F). \quad (4)$$

Therefore, this result of Theorem 3 can be proved using a method similar to that used in Theorem 1 (Lemma 7 is used in this process).

5 Conclusion

Fault-tolerance is an important issue for retaining the system's reliability. This study investigates edge fault tolerance of graphs with respect to super-λ_k. Recently, the bounds of this result for $k \in \{1,2\}$ are already presented. In this paper, we show that (1) $\rho_k(G) = \lambda_1(G)-1$ if the graph G is super-λ_k, but not λ_{k+1}-connected for $k \geq 3$, (2) $\min\{\lambda_4(G)-\xi_3(G)-1, \eta_3(G)-1, \lambda_1(G)-1\} \leq \rho_3(G) \leq \lambda_1(G)-1$ for any super-λ_3 and λ_4-connected graph G, and (3) $\min\{\lambda_{k+1}(G)-\xi_k(G)-1, \eta_k(G)-1, \delta(G)-k+1, \lambda_1(G)-1\} \leq \rho_k(G) \leq \lambda_1(G)-1$ for any super-λ_k and λ_{k+1}-connected graph G, where $k \geq 4$. A future work should be to evaluate tighter bounds of $\rho_k(G)$ for the graphs which satisfy some conditions.

References

1. Bonsma, P., Ueffing, N., Volkmann, L.: Edge-cuts leaving components of order at least three. Discrete Mathematics **256**(1), 431–439 (2002)
2. Balbuena, C., Marcote, X.: The k-restricted edge-connectivity of a product of graphs. Discrete Applied Mathematics **161**(1), 52–59 (2013)
3. Esfahanian, A.H., Hakimi, S.L.: On computing a conditional edge-connectivity of a graph. Information Processing Letters **27**(4), 195–199 (1988)
4. Fàbrega, J., Fiol, M.A.: Extraconnectivity of graphs with large girth. Discrete Mathematics **127**(1), 163–170 (1994)
5. Fàbrega, J., Fiol, M.A.: On the extraconnectivity of graphs. Discrete Mathematics **155**(1), 49–57 (1996)
6. Hong, Y., Meng, J., Zhang, Z.: Edge fault tolerance of graphs with respect to super edge connectivity. Discrete Applied Mathematics **160**(4), 579–587 (2012)
7. Hong, Z.M., Xu, J.M.: Vulnerability of super edge-connected networks. Theoretical Computer Science **520**, 75–86 (2014)
8. L̈u, M., Chen, G.L., Xu, X.R.: On super edge-connectivity of product graphs. Applied Mathematics and Computation **207**(2), 300–306 (2009)
9. Meng, J.X.: Optimally super-edge-connected transitive graphs. Discrete Mathematics **260**(1), 239–248 (2003)
10. Meng, J.X., Ji, Y.H.: On a kind of restricted edge connectivity of graphs. Discrete applied mathematics **117**, 183–193 (2002)
11. Ou, J.: Edge cuts leaving components of order at least m. Discrete mathematics **305**(1), 365–371 (2005)
12. Shang, L., Zhang, H.: Sufficient conditions for graphs to be λ'-optimal and super-λ'. Networks **49**(3), 234–242 (2007)
13. Shang, L., Zhang, H.: Degree conditions for graphs to be λ_3-optimal and super-λ_3. Discrete Mathematics **309**(10), 3336–3345 (2009)
14. Wang, M., Li, Q.: Conditional edge connectivity properties, reliability comparison and transitivity of graphs. Discrete Mathematics **258**, 205–214 (2002)
15. Wang, S., Lin, S., Li, C.: Sufficient conditions for super k-restricted edge connectivity in graphs of diameter 2. Discrete Mathematics **309**(4), 908–919 (2009)
16. Xu, J.: Topological structure and analysis of interconnection networks. Springer Publishing Company, Incorporated (2010)
17. Zhang, Z., Yuan, J.: A proof of an inequality concerning k-restricted edge connectivity. Discrete mathematics **304**(1), 128–134 (2005)
18. Zhao, W., Ou, J.: On restricted edge-connectivity of lexicographic product graphs. International Journal of Computer Mathematics **91**(8), 1618–1626 (2014)

Encoding and Security

Quantifying Communication in Synchronized Languages

Zhe Dang[1,2], Thomas R. Fischer[2], William J. Hutton III[2](✉), Oscar H. Ibarra[3], and Qin Li[1]

[1] School of Computer Science and Technology, Anhui University of Technology, Ma'anshan, China
[2] School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA
wiiliam.hutton@gmail.com
[3] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA

Abstract. A mutual information rate is proposed to quantitatively evaluate inter-process synchronized communication. For finite-state processes with implicit communication that can be described by a counting language, it is shown that the mutual information rate is effectively computable. When the synchronization always happens between the same two symbols at the same time (or with a fixed delay), the mutual information rate is computable. In contrast, when the delay is not fixed, the rate is not computable. Finally, it is shown that some cases exist where the mutual information rate is not computable.

1 Introduction

Computer systems often run a set of processes communicating with a provided interface for processes to talk with each other. The system may implement highly complex functions; e.g., a concurrent system. Obviously, communications between two processes contribute significantly to the complexity of the whole system. Therefore, it is desirable to find a way to measure the *quantity* of the communications between two processes. Since such communications are often nondeterministic, the quantity would also be part of an indication of hardness of testability when an analyst tests the system.

However, defining such a quantity is not trivial. Static analysis [3,19,23] would not always work because the quantity we are looking for is a dynamic indicator on how "tightly" two communicating processes are working together. Another idea would be to use the classic theory of communication complexity [13,18,21]. However, this theory aims at studying the minimal communication bit-rate needed for a given communication task instead of analyzing the actual amount of communication involved between given processes [27].

In this paper, we provide a metric to quantify the amount of communication between two communicating processes where the communication mechanism is synchronous. Our ideas are as follows. When a process runs, it demonstrates a behavior, which is a sequence (or a word) of events. In this way, two processes, when run synchronously, demonstrate two parallel aligned sequences. In automata theory, the parallel sequences can be thought of a word of two tracks called a synchronized word. The set of all such synchronized words that are the actual runs of the two communication processes forms a synchronized language, written as L_{12}. The metric we study is defined as the information in bits shared between two tracks (each called a string, and both strings are of the same length) in a synchronized word of the language.

There has been already a definition of the amount of information contained in a word. This definition was proposed by Shannon [22] and later Chomsky and Miller [8], that we have evaluated through experiments [5,7,9,11]. For a number n, we use $S_n(L)$ to denote the number of words in a language L whose length is n. The *information rate* λ_L of L is defined as $\lambda_L = \lim \lambda_{n,L}$, where $\lambda_{n,L} = \frac{\log S_n(L)}{n}$. When the limit does not exist, we take the upper limit, which always exists for a finite alphabet. Throughout this paper, we use log_2.

The intuition behind Shannon's definition is as follows. $\lambda_{n,L}$ specifies the average number of bits needed per symbol, i.e. bit rate, if one losslessly compresses a word of length n in L, while the information rate λ_L is simply the asymptotic bit rate. In other words, λ_L is the average amount of information per symbol contained in a word in L.

Notice that communication passes information between two processes. Let L_1 and L_2 be the projections of the aforementioned synchronized language L_{12} to the first and second tracks, respectively. Let w_1 and w_2 be the aforementioned two strings that form a two-track word w_{12}, with length n in the synchronized language L_{12}. By definition, $\log S_n(L_2)$ is the number of bits needed (on average) to encode w_2. Those bits are divided into two parts: a) the bits or the information that w_1 "knows" about or shares with w_2 as the result of communication, and b) the bits or the information that w_1 does not share with w_2. There are possibly many occurrences of w_2 in L_2 to pair with w_1 to make a two-track word in L_{12}. Due to nondeterminism, the mapping from w_1 to w_2 is one-to-many. Hence, for part b), what w_1 does not know is which branch to take in the mapping to reach w_2. Notice that $\frac{S_n(L_{12})}{S_n(L_1)}$ is the average branching factor in the mapping. Therefore, $\log S_n(L_{12}) - \log S_n(L_1)$ is the number of bits needed to encode a branch, i.e. the bits in part b). In summary, a) is the amount of information shared between w_1 and w_2 on average, $\log S_n(L_2) - (\log S_n(L_{12}) - \log S_n(L_1))$ which equals $\log S_n(L_1) + \log S_n(L_2) - \log S_n(L_{12})$. Taking its asymptotic form, we now define the mutual information rate to quantify the communication in L_{12}:

$$\eta_{L_{12}} = \lambda_{L_1} + \lambda_{L_2} - \lambda_{L_{12}}. \tag{1}$$

In the paper, we show cases when computing the mutual information rate is effective. These cases assume that the two processes are finite-state but the implicit communication mechanism between the two makes the resulting synchronized language L_{12} rather complex; e.g., a counting language (a regular language constrained by a Presburger formula on counts of individual symbols in a word. Notice that a counting language may be nonregular). We show that when the synchronization always happens between two symbols that are the same and that are at the same time, the mutual information is computable. The proof is quite complex, which involves combinatorial analysis of the synchronized words in L_{12} and properties from reversal-bounded counter machines [14]. Later, we also show that this result can be further generalized to cases when the two symbols are not necessarily synchronized at the same time (with a fixed delay). However, the case of arbitrary delays is not computable. We also present some other uncomputable cases as well.

We note that computing the mutual information rate of L_{12} is not trivial at all. We have cases (see the comment right after Theorem 7) where the information rates of L_1 and L_2 are computable but the information rate of L_{12} (and hence the mutual information rate $\eta_{L_{12}}$) is not computable. We also have cases where the information rate of L_{12} is computable but the information rates of L_1 and L_2 are unknown to be computable or takes some nontrivial effort in proving their computability (as we will do in the proof of Theorem 4).

2 Quantifying Communication with Information Rate

Let Σ be an alphabet and consider two languages L_1 and L_2 on the alphabet Σ. For the purpose of this paper, a word on the alphabet represents an observable behavior of a process, which is a sequence of its observable events. Such an event can be, for instance, a state transition when the states are observable. Suppose that P_i $(i = 1, 2)$ is a process. One can think the two processes as two nodes on a network, two concurrent programs running on a processor, two persons monitored by a surveillance cameras in a building, or simply two people dancing on a stage. When the two processes are observed together, a joint behavior is obtained. To ease our presentation, we assume that, whenever an event (say a) is observed in one process, an event (say b) is also observed in another process. Hence, the two processes, intuitively, run at the same pace by observation. Actually, this assumption is made without loss of generality. This is because, one can always introduce a new "idle" event into Σ. At the time when a is observed in one process and, at this time, if no event is observed in another, we treat the "no event" as the idle event. The two processes still run at the same pace by observation.

With this formulation, a joint behavior is a *synchronized word* α on alphabet Σ^k. Without loss of generality, we take $k = 2$ (i.e. two processes). It is trivial to generalize all the results in the paper to an arbitrary k. In the two process case, a synchronized word, α, is of the form

$$(a_1^1, a_1^2) \cdots (a_n^1, a_n^2) \tag{2}$$

for some n and some words $w_1 = a_1^1 \cdots a_n^1$ and $w_2 = a_1^2 \cdots a_n^2$ in Σ^*. For notational convenience, we often write $\alpha = [w_1, w_2]$ while, implicitly, we assume that the two projections w_1 and w_2 share the same length. In the sequence, w_1 is called the first coordinate of α while w_2 is the second coordinate of α.

The synchronized word α can be thought of as being woven from its first coordinate w_1 and second coordinate w_2. When one thinks of a joint run of two synchronized processes as a synchronized word, a restriction may be placed on the possible α so that not every pair of w_1 and w_2 can be woven into an actual joint run. For instance, under the scenario that an event a in process P_1 must be synchronized with an event $\dot{a}$ in process P_2, the synchronized word $[aba, \dot{a}\dot{a}\dot{a}]$ cannot be a joint run. The exact form of the restriction can be an explicit definition of the communication mechanism used between the processes or an implicit constraint on the set L_{12} of all the synchronized words α observed as joint runs. We take the latter approach. That is, in other words, we are given a *synchronization predicate* $R(w_1, w_2)$ to define the L_{12}: for all w_1 and w_2, $[w_1, w_2] \in L_{12}$ iff $R(w_1, w_2)$. Implicitly, $R(w_1, w_2)$ assumes that the lengths of w_1 and w_2 are the same. We use L_1 to denote the projection of L_{12} onto the first coordinate; i.e. $L_1 = \{w_1 : [w_1, w_2] \in L_{12}\}$. L_2 is defined similarly.

The information rate $\lambda_{L_{12}}$, defined in Section 1, can be thought of the entropy of a pair of random variable (X, Y) with certain joint distribution $p(X, Y)$. Then, the information rate λ_{L_1} (resp. λ_{L_2}) is naturally the entropy of random variable X (resp. Y) with marginal distribution $p(X)$ (resp. $p(Y)$) induced from the joint distribution $p(X, Y)$. In this sense, the *mutual information rate* between L_1 and L_2 with respect to the synchronized language L_{12} is defined as in (1) which can be intuitively seen from the classic Venn diagram of Shannon entropy [4].

The mutual information rate $\eta_{L_{12}}$ can be used in quantifying the amount of communication in the synchronized language L_{12}. In other words, suppose that P_1 (resp. P_2) is a process whose observable event sequences are exactly those in L_1 (resp. L_2). When P_1 and P_2 run synchronously, a synchronization mechanism (which is a form of communication) may be placed in the processes so that the resulting joint run is guaranteed to be in L_{12}. Now, when there is no communication between the two, the two processes run independently and hence L_{12} is simply the "Cartesian product" $[L_1, L_2] = \{[w_1, w_2] : w_1 \in L_1, w_2 \in L_2\}$ of L_1 and L_2. In this case, one can verify that $\eta_{L_{12}} = 0$. On the other hand, when P_1 and P_2 must synchronize on every event; e.g., $L_{12} = \{[w_1, w_2] : w_1 = w_2, w_1 \in L_1, w_2 \in L_2\}$ and $L_1 = L_2$, one can verify that $\eta_{L_{12}} = \lambda_{L_1} = \lambda_{L_2}$. That is, the synchronization mechanism makes sure that P_1 and P_2 share the complete information. Intuitively, $\eta_{L_{12}}$ characterize the average number of bits transmitted between the two processes per event observed, asymptotically.

3 Computing Mutual Information Rate in Synchronized Languages

Recall that, in order to compute the mutual information rate in a synchronized language L_{12}, it suffices for us to calculate the information rates $\lambda_{L_1}, \lambda_{L_2}$ and $\lambda_{L_{12}}$, according to (1). That is, we need algorithms to compute the information rate of a language drawn from a class. The following is a fundamental result.

Theorem 1. *The information rate of a regular language is computable [8].*

The case when L is non-regular (e.g., L is the external behavior set of a software system containing (unbounded) integer variables like counters and clocks) is more interesting, considering the fact that a complex software system nowadays is almost always of infinite state and the notion of information rate has been used in software testing [25].

However, in such a case, computing the information rate is difficult (sometimes even not computable [16]) in general. Existing results (such as unambiguous context-free languages [17], Lukasiewicz-languages [24], and regular timed languages [1]) are limited and mostly rely on structure generating functions and the theory of complex/real functions, which are also difficult to generalize. A recent important result [6], using a convex optimization technique, will be used in the paper. First, some definitions are needed.

A counter is a non-negative integer variable that can be incremented by 1, decremented by 1, or remain unchanged. Additionally, a counter can be tested for equality with 0. Let k be a non-negative integer. A *nondeterministic k-counter machine (NCM)* is a one-way nondeterministic finite automaton, with input alphabet Σ, augmented by k counters. For a non-negative integer r, we use NCM(k,r) to denote the class of k-counter machines where each counter is *r-reversal-bounded*; i.e. it makes at most r alternations between non-decreasing and non-increasing modes in any computation; e.g., the following counter value sequence $0\ 0\ 1\ 2\ 2\ 3\ \underline{3\ 2}\ 1\ 0\ \underline{0\ 1}\ 1$ is of 2-reversal, where the reversals are underlined. For convenience, we sometimes refer to a machine M in the class as an NCM(k,r). In particular, when k and r are explicitly given, we call M a *reversal-bounded NCM*. When M is deterministic, we use 'D' in place of 'N' (e.g., DCM). As usual, $L(M)$ denotes the language that M accepts.

Reversal-bounded NCMs have been extensively studied since their introduction in 1978 [14]; many generalizations are identified; e.g., multiple tapes, two-way tapes, stacks, etc. In particular, reversal-bounded NCMs have found applications in areas like Alur and Dill's [2] time-automata [10,12], Paun's [20] membrane computing systems [15], and Diophantine equations [26].

In a recent paper [6], we have shown the following result.

Theorem 2. *The information rate of the language accepted by a reversal-bounded DCM is computable.*

Notice that, in the theorem, the case for a reversal-bounded NCM is open.

We now recall a number of definitions that will be useful later. Let N be the set of non-negative integers and k be a positive number. A subset S of N^k is a linear set if there are vectors $\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_t$, for some t, in N^k such that $S = \{\mathbf{v} | \mathbf{v} = \mathbf{v}_0 + b_1\mathbf{v}_1 + \cdots + b_t\mathbf{v}_t, b_i \in N\}$. S is a semi-linear set if it is a finite union of linear sets. Let $\Sigma = \{a_1, \cdots, a_k\}$ be an alphabet. For each word $\alpha \in \Sigma^*$, define the Parikh map of α to be the vector $\#(\alpha) = (\#_{a_1}(\alpha), \cdots, \#_{a_k}(\alpha))$, where each symbol count $\#_{a_i}(\alpha)$ denotes the number of symbol a_i's in α. For a language $L \subseteq \Sigma^*$, the Parikh map of L is the set $\#(L) = \{\#(\alpha) : \alpha \in L\}$. The language L is semi-linear if $\#(L)$ is a semi-linear set. The following theorem is a classic result needed in this paper:

Theorem 3. *Let M be a reversal-bounded NCM. Then $\#(L(M))$ is a semi-linear set effectively computable from M [14].*

Let Y be a finite set of integer variables. An atomic Presburger formula on Y is either a linear constraint $\sum_{y \in Y} a_y y < b$, or a mod constraint $x \equiv_d c$, where a_y, b, c and d are integers with $0 \leq c < d$. A Presburger formula can always be constructed from atomic Presburger formulas using $\neg$ and $\wedge$. Presburger formulas are closed under quantification. Let S be a set of k-tuples in N^k. S is Presburger definable if there is a Presburger formula $P(y_1, \cdots, y_k)$ such that the set of non-negative integer solutions is exactly S. It is well-known that S is a semi-linear set iff S is Presburger definable.

In the rest of the section, we focus on the case when the synchronization predicate R in defining the L_{12} is given, how we are going to compute the mutual information rate $\eta_{L_{12}}$? In practice, such a predicate may be unknown (e.g., a black box implementation of a concurrent system). We will investigate how to estimate the mutual information rate $\eta_{L_{12}}$ for this case in a forthcoming paper.

When R is regular (i.e. the synchronized language L_{12} defined by R is a regular language), both L_1 and L_2 are also clearly regular languages. Then from Theorem 1 and the definition in (1), we have,

Corollary 1. *The mutual information rate in a regular synchronized language is computable.*

The result implies that, for finite state processes that run synchronously, the amount of communication between the two can be computed effectively.

A simple example would be two processes using a common resource as a covert communication channel. Imagine one process, C_W, is a covert writer, and another process, C_R, is a covert reader. In order to implement error-checking to ensure 100% accuracy when transmitting a message, C_W will not begin writing symbols until C_R acknowledges that it is ready to receive a symbol by synchronizing with C_W. When C_R has received a symbol from C_W, C_R moves its tape head left and overwrites the previous symbol on the tape with symbol it just received from C_W. C_W is idle until the symbols match. This enables C_W to wait until C_R has acknowledged the sent symbol before sending the next symbol. C_W sends an end-of-file symbol at the end of the message.

When R is not regular, computing the mutual information rate of L_{12} is more difficult and, in many cases, is not even effective. This is because, in doing so, the projected languages L_1 and L_2 are likely accepted by nondeterministic automata and hence, as we have mentioned earlier, their information rates in (1) are known to be computable for only a few cases. We now investigate the case for a non-regular R where counts of symbols are involved. More precisely, we employ a vector of variables $\#_{(a,b)}$, $a, b \in \Sigma$ and consider a Presburger formula P over these variables. For a synchronized word α in (2), we use $\#(\alpha)$ to indicate the vector of all Parikh maps $\#_{(a,b)}(\alpha)$ (i.e. the number of pairs (a, b) appearing in α), $a, b \in \Sigma$. A synchronized word α in (2) satisfies P if $P(\#(\alpha))$ holds.

We say that R is a counting predicate if it is specified by $[Q_1, Q_2] \wedge P$ where Q_1 and Q_2 are regular predicates and P is a Presburger formula. In consequence, L_{12} that is defined by R is a counting language; i.e. L_{12} is the intersection of

- the Cartesian product of the regular languages $\hat{Q}_1 = \{w_1 : Q_1(w_1)\}$ and $\hat{Q}_2 = \{w_2 : Q_2(w_2)\}$, and
- the Presburger language $\hat{P} = \{\alpha : P(\#(\alpha))\}$.

In other words, L_{12} is defined as $[\hat{Q}_1, \hat{Q}_2] \cap \hat{P}$. By definition, a synchronized word $[w_1, w_2]$ is in L_{12} iff (a). $w_1 \in \hat{Q}_1$ and $w_2 \in \hat{Q}_2$, and (b). $[w_1, w_2] \in \hat{P}$. Observe that condition (a) above specifies a condition independently applied on w_1 and w_2. However, for condition (b), one can imagine an explicit communication mechanism between w_1 and w_2 is placed to enforce the condition P. We now further specify an explicit (regular) synchronization mechanism Syn between the two. As a result, the L_{12} is now a synchronized counting language and defined by

$$[\hat{Q}_1, \hat{Q}_2] \cap \hat{P} \cap \hat{Syn}. \tag{3}$$

We now investigate the cases when the mutual information rates for synchronized counting languages are computable.

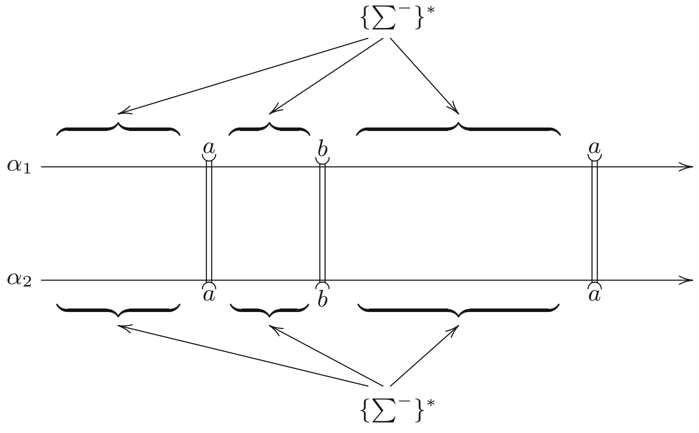


Fig. 1. Scenario of 0-delay Syn where $a, b \in \Pi$

Syn is used to specify an explicit synchronization pattern between two state sequences. A common pattern is to force certain state pairs to move together; e.g., dancer A jumps iff dancer B jumps. Formally, we are given a subset of synchronized symbols $\Pi \subseteq \Sigma$ and define $\Sigma^- = \Sigma - \Pi$ to be the set of independent symbols. The first pattern we consider is depicted as in Figure 1. In other words, $\hat{Syn}$ defines a regular language on alphabet $(\Sigma^- \times \Sigma^-) \cup \{(a, a) : a \in \Pi\}$. We call such a Syn 0-delay.

Consider the previous covert writer and reader example. We now assume that C_W and C_R agreed in advance on a subset of symbols $\Sigma' \subseteq \Sigma$ to be used for the leaked information. Clearly, appearances of such symbols should be sparse within the symbol stream transmitted. One can have various ways to specify the sparsity. Here, we use a simple one: the total number of appearances of symbols in Σ' when the end-of-file marker is reached can not exceed 1% of the total number of symbols transmitted. The synchronized language in consideration now is a pair of state-symbol sequences of the reader and the writer running synchronously and following the sparsity constraint. Clearly, the synchronized language is nonregular and in fact it is a synchronized counting language.

Theorem 4. *The mutual information rate in a synchronized counting language, specified in (3) with a 0-delay Syn, is computable.*

We shall point out that one can not simply obtain Theorem 4 from Theorem 1 since a counting language can be nonregular. Also, a difficulty of the proof of Theorem 4 is the fact that the projected L_1 and L_2 from a synchronized counting language L_{12} would be accepted by nondeterministic automata (as we have mentioned earlier, information rate of languages accepted by (infinite state) nondeterministic automata is rare to be known computable).

The second pattern of *Syn* is a slight generalization of the first, namely d-delay *Syn*. It says that any two corresponding synchronization symbols are separated away by exactly d symbols. Clearly, such a *Syn* is also regular.

It is straightforward to generalize the proof of Theorem 4 to d-delay *Syn*.

Theorem 5. *The mutual information rate in a synchronized counting language, specified in (3) with a d-delay Syn, is computable, for any given d.*

Recall that *Syn* is intended to describe a "regular" communication mechanism between two processes; i.e. the corresponding $\hat{Syn}$ defines a regular language (of synchronized words). However, we are not able to further generalize Theorem 4 to such *Syn*. For instance, in Theorem 4, $\hat{Syn}$ is a language on alphabet $(\Sigma^- \times \Sigma^-) \cup \{(a, a) : a \in \Pi\}$. That is, a symbol a must be synchronized to the same symbol a. When a symbol can be synchronized with multiple symbols, it is open whether the theorems obtained so far still hold.

Some other non-regular patterns *Syn* are also interesting. One such example is that the *Syn* allows arbitrary delay between two synchronized symbols. That is, $Syn = \{[w_1, w_2] : w_1 \downarrow_\Pi = w_2 \downarrow_\Pi, w_1, w_2 \in \Sigma^*\}$ where $w_1 \downarrow_\Pi = w_2 \downarrow_\Pi$ says that the projections of w_1 and w_2 on synchronization symbols Π (i.e. the results of dropping symbols not in Π) are the same. This third pattern is called an arbitrary-delay *Syn*. Notice that this pattern excludes "unaligned" sequences like $[eeaeeebee, ebeaeeeee]$, where $a, b \in \Pi$ and $e \notin \Pi$ (Here, noticing that the projections of the two words to Π are ab and ba respectively, which are not the same).

The covert communication processes explained earlier are an example of this pattern in practice when they are non-blocking and not time coupled. Although there is an arbitrary amount of delay between C_W sending a symbol and C_R

acknowledging the receipt of the sent symbol, the processes could still be considered synchronous because C_W will not send another symbol until C_R acknowledges the previous symbol.

However, for arbitrary-delay *Syn*, we have a (surprising) negative result. This result also reveals that the synchronization mechanism plays a key role in computing the mutual information rate.

Theorem 6. *The mutual information rate in a synchronized counting language, specified in (3) with an arbitrary-delay Syn is not computable.*

There are also other cases when the mutual information rate is not computable. For instance, a synchronization mechanism results in a L_{12} whose emptiness is undecidable (and hence its information rate is uncomputable). We will investigate some of these undecidable cases.

Below, we use a pair of automata working in parallel to specify the synchronization mechanism resulting the synchronized language L_{12} (i.e. the automata are not necessarily the actual concurrent processes; we will see an example right after Theorem 7 in a moment). Let M_1 and M_2 be two one-way deterministic/nondeterministic automata. They work on two tapes separately and independently. We use $[w_1, w_2]$ to denote the two inputs on the two tapes. Notice that the lengths of the two words w_1 and w_2 are equal (hence one may also think it is a single input tape with two tracks – M_1 and M_2 work on the two tracks respectively and they do not have to move in the same pace). However, these two automata actually do not move completely independent. When one reads a symbol $a \in \Pi$ from its tape storing w_1, the other must also read the same symbol a from its tape storing w_2, where $a \in \Pi$ ($\Pi \subseteq \Sigma$ is a given set of synchronization symbols). In other words, $w_1 \downarrow_{\Pi} = w_2 \downarrow_{\Pi}$. Again, by assumption the lengths of w_1 and w_2 are the same. We say that $[w_1, w_2]$ is accepted by $[M_1, M_2]$ if M_1 and M_2 accept w_1 and w_2 at the ends of the two inputs, respectively. Here, L_{12} is the synchronized language accepted by $[M_1, M_2]$.

We now define more precisely how the one-way machines operate. We assume that the input to a machine M has left and right end markers. At each step during the computation on a given input w (which has left and right end markers, so the input to M is actually $\#w\$$, where $\#$ and $\$$ are the left and right end markers), the machine's input head can remain on the same cell or move one cell to the right. (Note that the cell can contain $\#$, $\$$, or a symbol in Σ.) The input is accepted if the machine eventually falls off the right end marker in an accepting state. M is *real-time* if it does not have stationary moves, i.e. the input head always moves one cell to the right.

Let M_1 and M_2 be two nondeterministic, one-way machines over input alphabet Σ. Let $\Sigma = \Pi \cup \Sigma^-$, where Π and Σ^- are the (possibly empty) sets of synchronizing and non-synchronizing symbols, respectively.

We say that M_1 and M_2 have a synchronized accepting computation on a pair of inputs (w_1, w_2) (the end markers are not shown) if the following holds: (1): $|w_1| = |w_2|$, and (2): For $i = 1, 2$, M_i has an accepting computation on w_i such that if at time t, one machine is scanning a synchronizing symbol a, then the other machine must also be scanning a at time t. Moreover, if on this synchronizing symbol one machine makes a stationary move (i.e. remains on the same cell), then the other machine must also make a stationary move.

If there is a synchronizing accepting computations on input pair (w_1, w_2), we use the notation $[w_1, w_2]$. The set of all $[w_1, w_2]$'s is denoted by L_{12}.

Theorem 7. *It is undecidable, given two real-time 1-reversal deterministic pushdown automata (resp., real-time deterministic one-counter automata) M_1 and M_2, whether $L_{12} = \emptyset$.*

Notice that, in Theorem 7, the languages L_1 and L_2 (projected using L_{12}) are also accepted by real-time 1-reversal deterministic pushdown automata (resp., real-time deterministic one-counter automata). It is known that the information rate of such languages is computable [17]. However, according to the theorem, the information rate of L_{12} itself is not computable.

One can use the automata $[M_1, M_2]$ in Theorem 7 to specify a quite complex synchronization mechanism. For instance, consider two finite state processes P_1 and P_2 where they are required to synchronize only on one symbol a. We further require that, when such synchronization happens, for each odd number i, the number of (unsynchronized) symbols between the i-th a and the $(i+1)$-th a equals the number of (unsynchronized) symbols between the $(i+1)$-th a and the $(i+2)$-th a. For instance, $[a \underbrace{bbc}_{3} a \underbrace{ccc}_{3} a \underbrace{b}_{1} a \underbrace{c}_{1} a, a \underbrace{dgg}_{3} a \underbrace{gdd}_{3} a \underbrace{g}_{1} a \underbrace{d}_{1} a]$ satisfies the requirement (which would be useful in a covert timing channel – in this example, the covert writer leaks out two numbers 3 and 1. The second copies of the 3 and the 1 are used as error-correcting at the covert reader's side.).

But for reversal-bounded NCMs, we have:

Theorem 8. *It is decidable, given two reversal-bounded NCMs M_1 and M_2, whether $L_{12} = \emptyset$.*

Corollary 2. *We can effectively construct, given two reversal-bounded NCMs (resp., DCMs) M_1 and M_2, a 2-tape reversal-bounded NCM (resp., DCM) M that accepts L_{12}.*

For a set $L \subseteq \Sigma^* \times \Sigma^*$, let $R(L) = \{x_1 \# x_2 \mid (x_1, x_2) \in L$, and $S(L) = \{x_1 \# x_2^R \mid (x_1, x_2) \in L\}$.

Corollary 3. *The information rates of $R(L_{12})$ and $S(L_{12})$ for reversal-bounded DCMs M_1 and M_2 are computable.*

A reversal-bounded NPCM is a nondeterministic pushdown automaton augmented with reversal-bounded counters. It is known that the emptiness, infiniteness, and disjointness problems for reversal-bounded NPCMs are decidable [14].

Corollary 4. *It is decidable, given a reversal-bounded NPCM M_1 and a reversal-bounded NCM M_2, whether $L_{12} = \emptyset$.*

The next result shows that under some restriction, emptiness of L_{12} is decidable when both machines are reversal-bounded NPCMs.

Let $k \geq 0$. A machine M is k-sync if every input it accepts has at most k-occurrences of synchronizing symbols on accepted inputs.

Theorem 9. *Let $k \geq 0$. It is decidable, given two k-sync reversal-bounded NPCMs M_1 and M_2 on accepted inputs (there is no restriction in the occurrences of non-synchronizing symbols), whether $L_{12} = \emptyset$.*

We can refine the proof Theorem 9. As we noted before, given two machines M_1 and M_2, we can think of the strings in L_{12} as a two-track tape, where the first (resp., second) track is accepted by $M_1(resp., M_2)$.

Theorem 10. *Let $k \geq 0$. We can construct, given two k-sync reversal-bounded NCMs (resp., reversal-bounded DCMs) M_1 and M_2, a reversal-bounded NCM (resp., reversal-bounded DCM) M accepting L_{12}.*

Let M_1 and M_2 be k-sync reversal-bounded DCMs (i.e. the accepted strings have at most k synchronizing symbols). We know from Theorem 10 that L_{12} is accepted by a reversal-bounded DCM. We now show that the projection of L_{12} on the i-th coordinate ($i = 1, 2$) is accepted by a reversal-bounded DCM.

Lemma 1. *Let M_1 and M_2 be k-sync reversal-bounded DCMs. Then L_i = projection of L_{12} on the i-th coordinate is accepted by a reversal-bounded DCM for $i = 1, 2$.*

Corollary 5. *The mutual information rate of L_{12} for k-sync DCMs M_1 and M_2 is computable, for any k.*

4 Conclusions

In this paper, the mutual information rate is proposed as a quantitative method for analyzing inter-process synchronized communication. For finite-state processes with implicit communication that can be described by a counting language, the mutual information rate is effectively computable. When the synchronization always occurs between the same two symbols at the same time (or with a fixed delay), the mutual information rate is also computable. Finally, some cases are described where the mutual information rate is not computable.

References

1. Asarin, E., Degorre, A.: Volume and entropy of regular timed languages: discretization approach. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 69–83. Springer, Heidelberg (2009)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)
3. Chen, H., Malacaria, P.: Quantitative analysis of leakage for multi-threaded programs. In: PLAS 2007, pp. 31–40. ACM (2007)
4. Cover, T.M., Thomas, J.A.: Elements of information theory, 2nd edn. Wiley-Interscience (2006)

5. Cui, C. , Dang, Z., Fischer, T., Ibarra, O.: Execution information rate for some classes of automata. Information and Computation (2015) to appear
6. Cui, C., Dang, Z., Fischer, T.R., Ibarra, O.H.: Information rate of some classes of non-regular languages: an automata-theoretic approach. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 232–243. Springer, Heidelberg (2014)
7. Cui, C., Dang, Z., Fischer, T.R., Ibarra, O.H.: Execution information rate for some classes of automata. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 226–237. Springer, Heidelberg (2013)
8. Chomsky, N., Miller, G.A.: Finite state languages. Information and Control **1**, 91–112 (1958)
9. Cui, C., Dang, Z., Fischer, T.R., Ibarra, O.H.: Similarity in languages and programs. Theor. Comput. Sci. **498**, 58–75 (2013)
10. Dang, Z., Ibarra, O., Bultan, T., Kemmerer, R., Su, J.: Binary reachability analysis of discrete pushdown timed automata. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 69–84. Springer, Heidelberg (2000)
11. Dang, Z., Ibarra, O., Li, Q.: Sampling a two-way finite automaton. In: Automata, Universality, Computation, the series book: Emergence, Complexity and Computation. Springer (2014)
12. Dang, Z.: Pushdown timed automata: a binary reachability characterization and safety verification. Theoretical Computer Science **301**(13), 93–121 (2003)
13. Gurari, E.M., Ibarra, O.H.: The complexity of decision problems for finite-turn multicounter machines. Journal of Computer and System Sciences **22**(2), 220–229 (1981)
14. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. ACM **25**(1), 116–133 (1978)
15. Ibarra, O.H., Dang, Z., Egecioglu, O., Saxena, G.: Characterizations of catalytic membrane computing systems. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 480–489. Springer, Heidelberg (2003)
16. Kaminger, F.P.: The noncomputability of the channel capacity of context-sensitive languages. Inf. Comput. **17**(2), 175–182 (1970)
17. Kuich, W.: On the entropy of context-free languages. Information and Control **16**(2), 173–20 (1970)
18. Kushilevitz, E.: Communication complexity. Advances in Computers **44**, 331–360 (1997)
19. Muller, S., Chong, S.: Towards a practical secure concurrent language. SIGPLAN Not. **47**(10), 57–74 (2012)
20. Paun, G.: Membrane Computing, an Introduction. Springer (2000)
21. Papadimitriou, C.H., Sipser, M.: Communication complexity. Journal of Computer and System Sciences **28**(2), 260–269 (1984)
22. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press (1949)
23. Shaffer, A.B., Auguston, M., Irvine, C.E., Levin, T.E.: A security domain model to assess software for exploitable covert channels. In: PLAS 2008, pp. 45–56. ACM (2008)
24. Staiger, L.: The entropy of Lukasiewicz-languages. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 155–165. Springer, Heidelberg (2002)

25. Wang, E., Cui, C., Dang, Z., Fischer, T.R., Yang, L.: Zero-knowledge black box testing: where are the faults? International Journal of Foundations of Computer Science **25**(2), 196–218 (2014)
26. Xie, G., Dang, Z., Ibarra, O.: A solvable class of quadratic Diophantine equations with applications to verification of infinite-state systems. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 668–680. Springer, Heidelberg (2003)
27. Yao, A.C.: Some complexity questions related to distributive computing (preliminary report). In: STOC 1979, pp. 209–213. ACM (1979)

Simultaneous Encodings for Range and Next/Previous Larger/Smaller Value Queries

Seungbum Jo(✉) and Srinivasa Rao Satti

Seoul National University, Seoul, South Korea
sbcho@tcs.snu.ac.kr, ssrao@cse.snu.ac.kr

Abstract. Given an array of n elements from a total order, we propose encodings that support various range queries (range minimum, range maximum and their variants), and previous and next smaller/larger value queries. When query time is not of concern, we obtain a $4.088n + o(n)$-bit encoding that supports all these queries. For the case when we need to support all these queries in constant time, we give an encoding that takes $4.585n + o(n)$ bits, where n is the length of input array. We first extend the original DFUDS [Algorithmica, 2005] encoding of the colored 2d-Min (Max) heap that supports the queries in constant time. Then, we combine the extended DFUDS of $2d$-Min heap and $2d$-Max heap using the Min-Max encoding of Gawrychowski and Nicholson [arXiv, 2014] with some modifications. We also obtain encodings that take lesser space and support a subset of these queries.

1 Introduction

Given an array $A[1 \dots n]$ of n elements from a total order, for $1 \le i \le j \le n$, the *Range Minimum Query* ($RMinQ_A(i, j)$) returns the position p such that $A[p]$ is the smallest value in the subarray $A[i, \dots, j]$. If there are repeated elements in A, and there are some positions $i \le r_1 \dots r_m \le j$ in A which are the positions of minimum values between $A[i]$ and $A[j]$, $RMinQ_A(i, j)$ returns an arbitrary position among $r_1 \dots r_m$. In this case, we can define three additional queries: *Range Leftmost/Rightmost Minimum Queries* ($RLMinQ_A(i, j)$ and $RRMinQ_A(i, j)$), and for $1 \le k \le m$, *Range k-th Minimum Query* ($RkMinQ_A(i, j)$), which return r_1, r_m and r_k respectively. We can define the maximum queries ($RMaxQ_A(i, j)$, $RLMaxQ_A(i, j)$, $RRMaxQ_A(i, j)$, $RkMaxQ_A(i, j)$) analogous to the minimum queries defined above.

We also consider the queries *previous smaller/larger value* ($PSV_A(i)$ and $PLV_A(i)$) which return the position of the nearest smaller/larger value among $A[1] \dots A[i-1]$ in A. We can also define the *next smaller/larger value* queries ($NSV_A(i)$ and $NLV_A)(i)$) analogously.

We consider these problems in the *encoding model* in which we do not have access to the original input array A at query time. The minimum size of an encoding is also referred to as the *effective entropy* of the input data (with respect to the queries) [9].

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 648–660, 2015.
DOI: 10.1007/978-3-319-21398-9_51

Previous Work. The range minimum/maximum problem has been well-studied in the literature. It is well-known [2] that finding $RMinQ_A$ can be transformed to the problem of finding the LCA (Lowest Common Ancestor) between (the nodes corresponding to) the two query positions in the Cartesian tree constructed on A. Furthermore, since different topological structures of the Cartesian tree on A give rise to different set of answers for $RMinQ_A$ on A, one can obtain an information-theoretic lower bound of $2n - \Theta(\lg n)$[1] bits on the encoding of A that answers $RMinQ$ queries. Fischer and Heun [7] introduced the $2d$-Min heap, which is a variant of the Cartesian tree, and showed how to encode it using the Depth first unary degree sequence (DFUDS) [3] representation in $2n + o(n)$ bits which supports $RMinQ_A$ queries in constant time. Fischer and Heun [6] also defined the *approximate range median of minima query* problem which returns a position $RkMinQ_A$ for some $\frac{1}{16}m \leq k \leq \frac{15}{16}m$, and proposed an encoding that uses $2.54n + o(n)$ bits and supports the *approximate* $RMinQ$ queries in constant time, using a *Super Cartesian tree.*

For PSV_A and NSV_A, if all elements in A are distinct, then $2n + o(n)$ bits are enough to answer the queries in constant time, by using the $2d$-Min heap of Fischer and Heun [7]. For the general case, Fischer [5] proposed the *colored* $2d$*-Min heap*, that uses an optimal $2.54n + o(n)$ bits to answer PSV_A and NSV_A in constant time.

One can support both $RMinQ_A$ and $RMaxQ_A$ in constant time trivially using the encodings for $RMinQ_A$ and $RMaxQ_A$ queries, using a total of $4n + o(n)$ bits. Gawrychowski and Nicholson reduce this space to $3n + o(n)$ bits while maintaining constant time query time [8]. But their scheme works only when there are no consecutive equal elements in A.

Our Results. We first extend the original DFUDS [3] for colored 2d-Min(Max) heap that supports the queries in constant time. Then, we combine the extended DFUDS of $2d$-Min heap and $2d$-Max heap using Gawrychowski and Nicholson's Min-Max encoding [8] with some modifications. As a result, we obtain the following non-trivial encodings that support a wide range of queries.

Theorem 1. *An array* $A[1 \dots n]$ *containing* n *elements from a total order can be encoded using*

(a) at most $3.167n + o(n)$ *bits to support* $RMinQ_A$, $RMaxQ_A$, $RRMinQ_A$, $RRMaxQ_A$, PSV_A, *and* PLV_A *queries;*
(b) at most $3.322n + o(n)$ *bits to support the queries in (a) in constant time;*
(c) at most $4.088n + o(n)$ *bits to support* $RMinQ_A$, $RRMinQ_A$, $RLMinQ_A$, $RkMinQ_A$,
PSV_A, NSV_A, $RMaxQ_A$, $RRMaxQ_A$, $RLMaxQ_A$, $RkMaxQ_A$, PLV_A *and* NLV_A *queries; and*
(d) at most $4.585n + o(n)$ *bits to support the queries in (c) in constant time.*

If the array contains no two consecutive equal elements, then (a) and (b) take $3n + o(n)$ *bits, and (c) and (d) take* $4n + o(n)$ *bits.*

[1] We use $\lg n$ to denote $\log_2 n$.

We assume the standard word-RAM model [11] with word size $\Theta(\lg n)$.

2 Preliminaries

We first introduce some useful data structures that we use to encode various bit vectors and balanced parenthesis sequences.

Bit Strings (Parenthesis Sequences). Given a string $S[1 \dots n]$ over the alphabet $\Sigma = \{(,)\}$, $rank_S(x, i)$ returns the number of occurrence of the string $x \in \Sigma^*$ in $S[1 \dots i]$ and $select_S(x, i)$ returns the first position of i-th occurrence of $x \in \Sigma^*$ in S. Combining the results from [12] and [14], one can show the following.

Lemma 1 ([12], [14]). *Let S be a string of length n over the alphabet $\Sigma = \{'(', ')'\}$ containing m closing parentheses. One can encode S using $\lg \binom{n}{m} + o(n)$ bits to support both $rank_S(x, i)$ and $select_S(x, i)$ in constant time, for any $|x| \leq 1/2 \lg n$. Also we can decode any $\lg n$ consecutive bits in S in constant time.*

Balanced Parenthesis Sequences. Given a string $S[1 \dots n]$ over the alphabet $\Sigma = \{'(', ')'\}$, if S is balanced and $S[i]$ is an open (close) parenthesis, then we can define $findopen_S(i)$ ($findclose_S(i)$) which returns the position of the matching close (open) parenthesis to $S[i]$. Munro and Raman [13] showed that if we can access any $\lg n$-bit subsequence of S in constant time, then both $findopen_S$ and $findclose_S$ can be answered in constant time using an additional $o(n)$ bits.

We use the following lemma (proof omitted) to bound the space usage of the data structures described in Section 4.

Lemma 2. *Given two positive integers a and n and a nonnegative integer $k \leq n$, $\lg \binom{n}{k} + a(n-k) \leq n \lg (2^a + 1)$.*

2.1 2d-Min Heap

The 2d-Min heap [7] on A, denoted by $Min(A)$, is designed to encode the answers of $RMinQ_A(i, j)$ efficiently. We can also define the 2d-Max heap on A ($Max(A)$) analogously. $Min(A)$ is an ordered labeled tree with $n + 1$ nodes labeled with $0 \dots n$. Each node in $Min(A)$ is labeled by its preorder rank and each label corresponds to a position in A. We extend the array $A[1 \dots n]$ to $A[0 \dots n]$ with $A[0] = -\infty$. In the labeled tree, the node x denotes the node labeled x. For every vertex i, except for the root node, its parent node is (labeled with) $PSV_A(i)$.

Encoding of 2d-Min Heap. *Depth first unary degree sequence (DFUDS)* is one of the well-known methods for representing ordinal trees [3]. It consists of a balanced sequence of open and closed parentheses, which can be defined inductively as follows. If the tree consists of the single node, its DFUDS is '()'. Otherwise, if the ordinal tree T has k subtrees $T_1 \dots T_k$, then its DFUDS, D_T

is the sequence $(^{k+1}\ )d_{T_1}\dots d_{T_k}$ (i.e., $k+1$ open parentheses followed by a close parenthesis concatenated with the 'partial' DFUDS sequences $d_{T_1}\dots d_{T_k}$) where d_{T_i}, for $1 \leq i \leq k$, is the DFUDS of the subtree T_i (i.e., D_{T_i}) with the first open parenthesis removed. From the above construction, it is easy to prove by induction that if T has n nodes, then the size of D_T is $2n$ bits. The following lemma shows that DFUDS representation can be used to support various navigational operations on the tree efficiently.

Lemma 3 ([1], [3], [10]). *Given an ordinal tree T on n nodes with DFUDS sequence D_T, one can construct an auxiliary structure of size $o(n)$ bits to support the following operations in constant time: for any two nodes x and y in T,*
- *$parent_T(x)$: Label of the parent node of node x.*
- *$degree_T(x)$: Degree of node x.*
- *$depth_T(x)$: Depth of node x (The depth of the root node is 0).*
- *$subtree\_size_T(x)$: Size of the subtree of T which has the x as the root node.*
- *$next\_sibling_T(x)$: The label of the next sibling of the node x.*
- *$child_T(x, i)$: Label of the i-th child of the node x.*
- *$child\_rank_T(x)$: Number of siblings left to the node x.*
- *$la_T(x, i)$: Label of the level ancestor of node x at depth i.*
- *$lca_T(x, y)$: Label of the least common ancestor of node x and y.*
- *$pre\_rank_T(i)$: The preorder rank of the node in T corresponding to $D_T[i]$.*
- *$pre\_select_T(x)$: The first position of node with preorder rank x in D_T.*

Using the operations in Lemma 3, Fischer and Heun [7] showed that $RMinQ_A(i, j)$ can be answered in constant time using $D_{Min(A)}$. If the elements in A are not distinct, $RMinQ_A(i, j)$ returns the $RRMinQ_A(i, j)$.

Fischer and Heun [7] also proposed a linear-time stack-based algorithm to construct $D_{Min(A)}$. Their algorithm maintains a min-stack consisting of a decreasing sequence of elements from top to the bottom. The elements of A into the min-stack from right to left and before pushing the element $A[i]$, all the elements from the stack that are larger than $A[i]$ are popped. The algorithm construct a sequence S which is initialized to ϵ, the empty string, at the beginning of the algorithm. Whenever k elements are popped from the stack and then an element is pushed into the stack, $(^k)$ is prepended to S. Finally, after pushing $A[1]$ into the stack, if the stack contains m elements, then $(^{m+1})$ is prepended to S. One can show that this sequence S is the same as the DFUDS sequence $D_{Min(A)}$. Analogously, one can construct $D_{Max(A)}$ using a similar stack-based algorithm.

From the definition of $2d$-Min heap, it is easy show that $PSV_A(i)$, for $1 \leq i \leq n$, is the label corresponding to the parent of the node labeled i in $Min(A)$. Thus, using the encoding of Lemma 3 of $2n + o(n)$ bits, one can support the $PSV_A(i)$ queries in constant time. A straightforward way to support $NSV_A(i)$ is to construct the $2d$-Min heap structure for the reverse of the array A, and encode it using an additional $2n + o(n)$ bits. Therefore one can encode all answers of PSV_A and NSV_A using $4n+o(n)$ bits with constant query time. To reduce this size, Fischer proposed the *colored $2d$-Min heap* [5]. This has the same structure as normal $2d$-Min heap, and in addition, the vertices are colored either red or blue. Suppose there is a parent node x in the colored $2d$-Min heap with its children $x_1 \dots x_k$. Then for $1 < i \leq k$, node x_i is colored red if $A[x_i] < A[x_{i-1}]$, and all the other nodes are colored blue (see (A) in Figure 1).

If all the elements in A are distinct, then a $2n + o(n)$-bit encoding of $Min(A)$ is enough to support $RMinQ_A$, $RRMinQ_A$, PSV_A and NSV_A with constant query time. In the general case, Fischer proposed an optimal $2.54n + o(n)$-bit encoding of colored $2d$-Min heap on A using TC-encoding [4]. This encoding also supports two additional operations, namely *modified child*$_{CMin(A)}(x, i)$ and *child_rank*$_{CMin(A)}(x)$, which answer the i-th red child of node x and the number of red siblings to the left of node x, respectively, in constant time. Using these operations, one can also support $RLMinQ_A$ and $RkMinQ_A$ in constant time.

2.2 Encoding Range Min-max Queries

One can support both $RMinQ_A$ and $RMaxQ_A$ in constant time by encoding both $Min(A)$ and $Max(A)$ separately using $4n + o(n)$ bits. Gawrychowski and Nicholson [8] reduce this space to $3n + o(n)$ bits while maintaining $O(1)$ query time. There are two key observations in their structure: (i) if we can access any $\lg n$-bit substring of $D_{Min(A)}$ and $D_{Max(A)}$ on $O(1)$ time, we can still support both queries in $O(1)$ time, using an additional $o(n)$ bits; (ii) if we generate $D_{Min(A)}$ and $D_{Max(A)}$ using Fischer and Heun's stack-based algorithm with min and max-stack, then whenever we push $A[i], 1 \leq i < n$ in the reverse order, we need to pop elements from exactly one of the min or max-stack, unless $A[i] = A[i+1]$ (in which case, we do not pop elements from either stack).

Now we describe the overall encoding in [8] briefly. The structure consists of two bit strings T and U along with various auxiliary structures. For $1 \leq i < n$, if k elements are popped from the min (max)-stack when we push $A[i] (1 \leq i < n)$ into both the stacks (from right to left), we prepend $(^{k-1}\ )$ and 0(1) to the currently generated T and U respectively. Initially, when $i = n$, both min and max stacks push ')' so we do not prepend anything to both strings. But we can recover it easily because this is the last ')' in common. Finally, after pushing $A[1]$ into both the stacks, we pop the remaining elements from them, and store the number of these popped elements in min and max stack explicitly using $\lg n$ bits. One can show that the size of T is at most $2n$ bits, and that of U is $n - 1$ bits. Thus the total space usage is at most $3n$ bits.

To recover any $\lg n$-bit substring, $D_{Min(A)}[d_1 \dots d_{\lg n}]$, in constant time we first divide the $D_{Min(A)}$ into blocks of size $\lg n$, and for the starting position of each block, store its corresponding position in T. For this purpose, we construct a bit string B_{min} of length at most $2n$ such that $B_{min}[i] = 1$ if and only if $T[i]$ corresponds to the start position of the ith-block in $D_{Min(A)}$. We encode B_{min} using the representation of Lemma 1 which takes $o(n)$ bits since B_{min} has only $2n/\lg n$ 1's. Then if d_1 belongs to the i-th block, it is enough to recover the i-th and the $(i+1)$-st blocks in the worst case.

Now, to recover the i-th block of $D_{Min(A)}$, we first compute the distance between i-th and $(i+1)$-st 1's in B_{min}. If this distance is less than $c \lg n$ for some fixed constant $c > 9$, we call it a *min-good block*, otherwise, we call it a *min-bad* *block*. We can recover a min-good block in $D_{Min(A)}$ in $O(c)$ time using a $o(n)$-bit pre-computed table indexed by all possible strings of length $\lg n/4$ bits for T and U (we can find the position corresponding to the i-th block in U in constant time), which stores the appropriate $O(\lg n)$ bits of $D_{Min(A)}$ obtained from them (see [8] for details). For min-bad blocks, we store the answers explicitly. This takes $(2n/(c \lg n)) \cdot \lg n = 2n/c$ additional bits. To save this additional space, we store the min-bad blocks in compressed form using the property that any min-bad block in $D_{Min(A)}$ and $D_{Max(A)}$ cannot overlap more than $4 \lg n$ bits in T, (since any $2 \lg n$ consecutive bits in T consist of at least $\lg n$ bits from either $D_{Min(A)}$ or $D_{Max(A)}$). So, for $c > 9$ we can save more than $\lg n$ bits by compressing the remaining $(c-4)\lg n$ bits in T corresponding to each min-bad block in $D_{Min(A)}$. Thus, we can reconstruct any $\lg n$-bit substring of $D_{Min(A)}$ (and $D_{Max(A)}$) in constant time, using a total of $3n + o(n)$ bits.

We first observe that if there is a position $1 \le i < n$ such that $A[i] = A[i+1]$, we cannot decode the '$)'$ in T which corresponds to $A[i]$ only using T and U since we do not pop any elements from both min and max stacks when we push $A[i]$ into both stacks. Thus the encoding of [8] only works when there are no repeated elements in A. We describe how to handle if there are repeated elements in A.

Gawrychowski and Nicholson [8] also show that any encoding that supports both $RMinQ_A$ and $RMaxQ_A$ cannot use less than $3n - \Theta(\lg n)$ bits for sufficiently large n (even if all elements in A are distinct).

3 Extended DFUDS for Colored 2d-Min Heap

In this section, we describe an encoding of colored $2d$-Min heap on A ($CMin(A)$) using at most $3n+o(n)$ bits while supporting $RMinQ_A$, $RRMinQ_A$, $RLMinQ_A$, $RkMinQ_A$, PSV_A and NSV_A in constant time, by storing the color information of the nodes using a bit string of length at most n, in addition to the DFUDS representation of $CMin(A)$. (We can also encode the colored $2d$-Max heap in a similar way). In the worst case, this representation uses more space than the colored $2d$-Min heap encoding of Fischer [5], but the advantage is that it separates the tree encoding from the color information. We later describe how to combine the tree encodings of the $2d$-Min heap and $2d$-Max heap, and (separately) also combine the color information of the two trees, to save space.

Now we describe the main encoding of $CMin(A)$. The encoding consists of two parts: $D_{CMin(A)}$ and V_{min}. The sequence $D_{CMin(A)}$ is same as $D_{Min(A)}$, the DFUDS representation of $CMin(A)$, which takes $2n + o(n)$ bits and supports the operations in Lemma 3 in constant time.

The bit string V_{min} stores the color information of all nodes in $CMin(A)$, except the nodes which are the leftmost children of their parents (the color of these nodes is always blue), as follows. Suppose there are $1 \le p \le n$ nodes which are the leftmost children of their parents. Then, $V_{min}[i]$, stores 0 if the color of the node $node_{V_{min}}(i) = pre\_rank_{CMin(A)}(findclose_{D_{CMin(A)}}(select_{D_{CMin(A)}}(i+1, `(('))+1)$ in $CMin(A)$ is red, and 1 otherwise, for $1 \le i \le n-p$. This follows from the observation that if there is an $i, 1 \le i < 2n-1$ such that

$D_{CMin(A)}[i] =$ '$($' and $D_{CMin(A)}[i+1] =$ '$)$', then $D_{CMin(A)}[i+2]$ corresponds to the node which is the leftmost child of the node $pre\_rank_{CMin(A)}(D_{CMin(A)}[i])$, so we skip these nodes by counting the pattern '$(\ ($' instead of '$($' in $D_{CMin(A)}$. Also, we set $V_{min}[0] = 1$, which corresponds to the first open parenthesis in $D_{CMin(A)}$. Therefore, the total length of V_{min} is $n - p + 1$. (We define the bit string V_{max} in a similar way.)

The following theorem (proof omitted) shows that encoding $Min(A)$ and V_{min} separately, using $3n + o(n)$ bits, has the same functionality as $CMin(A)$ encoding of Fischer [5], which takes $2.54n + o(n)$ bits.

Theorem 2. *For an array $A[1 \dots n]$ of length n, there is an encoding for A which takes at most $3n + o(n)$ bits and supports $RMinQ_A$, $RRMinQ_A$, $RLMinQ_A$, $RkMinQ_A$, PSV_A and NSV_A in constant time.*

4 Encoding Colored 2d-Min and Max Heaps

In this section, we describe our encodings for supporting various subsets of operations, proving the results stated in Section 1. As mentioned in Section 2.1, the TC-encoding of the colored $2d$-Min heap of Fischer [5] can answer $RMinQ_A$, $RLMinQ_A$, $RkMinQ_A$, $RRMinQ_A$, PSV_A and NSV_A queries in $O(1)$ time, using $2.54n+o(n)$ bits. If we store a similar TC-encoding of colored $2d$-Max heap in addition, then we can support all the operations mentioned in Theorem 1(c) in $O(1)$ time, using at most $5.08n + o(n)$ bits.

We show that a combined encoding of $D_{CMin(A)}$ and $D_{CMax(A)}$, using at most $3.167n+o(n)$ bits, can be used to answer $RMinQ_A$, $RMaxQ_A$, $RRMinQ_A$, $RRMaxQ_A$, PSV_A, and PLV_A queries (Theorem 1(a)). To support the queries in constant time, we use a less space-efficient data structure that encodes the same structures, using at most $3.322n + o(n)$ bits (Theorem 1(b)). Similarly, a combined encoding of $D_{CMin(A)}$, $D_{CMax(A)}$, V_{min} and V_{max} using at most $4.088n + o(n)$ bits can be used to answer $RLMinQ_A$, $RkMinQ_A$, NSV_A, $RLMaxQ_A$, $RkMaxQ_A$, and NLV_A queries in addition (Theorem 1(c)). Again, to support the queries in constant time, we obtain a less space-efficient data structure using at most $4.58n + o(n)$ bits (Theorem 1(d)).

In the following, we first describe the data structure of Theorem 1(b) followed by the structure for Theorem 1(d). Next we describe the encodings of Theorem 1(a) and Theorem 1(c).

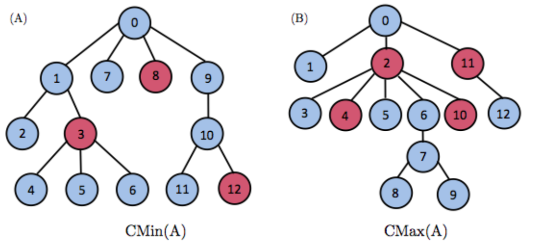
4.1 Combined Data Structure for $D_{CMin(A)}$ and $D_{CMax(A)}$

As mentioned in Section 2.2, the encoding of Gawrychowski and Nicholson [8] consists of two bit strings U and T of total length at most $3n$, along with the encodings of B_{min}, B_{max} and a few additional auxiliary structures of total size $o(n)$ bits. In this section, we denote this encoding by E. To encode the DFUDS sequences of $CMin(A)$ and $CMax(A)$ in a single structure, we use E with some modifications, which we denote by E'. As described in Section 2.2, encoding scheme of Gawrychowski and Nicholson does not work if there is a position i, for $1 \leq i < n$ such that $A[i] = A[i+1]$. To solve this problem, we define an additional bit string $C[1 \ldots n]$ such that $C[1] = 0$, and for $1 < i \leq n$, $C[i] = 1$ iff $A[i-1] = A[i]$.If the bit string C has k ones in it, then we can represent C using $\lg \binom{n}{k} + o(n)$ bits while supporting rank, select queries and decoding any $\lg n$ consecutive bits in C in constant time using Lemma 1. We also define a new array $A'[0 \ldots n-k]$ by setting $A'[0] = A[0]$ and for $0 < i \leq n-k$, $A'[i] = A[select_C(i, 0)]$. We also define another sequence $D'_{CMin(A)}$ of size $2n-k$ as follows. Suppose $D_{CMin(A')} = (^{\delta_1}\ ) \ldots (^{\delta_{n-k}}\ )$, for some $0 \leq \delta_1 \ldots \delta_n \leq n-k$, then we set $D'_{CMin(A)} = (^{\delta_1+\epsilon_1}\ ) \ldots (^{\delta_{n-k}+\epsilon_{n-k}}\ )$, where $\delta_i + \epsilon_i$ is the number of elements popped when $A[i]$ is pushed into the min stack of A, for $1 \leq i \leq n-k$. (Analogously, we define $D'_{CMax(A)}$.)

The encoding E' defined on A consists of two bit strings U' and T', along with B'_{min}, B'_{max} and additional auxiliary structures (as in E). Let U and T be the bit strings in E defined on A'. Then U' is same as U in E. To obtain T', we add some additional open parentheses to T as follows. Suppose $T = (^{\delta_1}\ ) \ldots (^{\delta_{n-k}}\ )$, for some $0 \leq \delta_1 \ldots \delta_n \leq n-k$. Then $T' = (^{\delta_1+\epsilon_1}\ ) \ldots (^{\delta_{n-k}+\epsilon_{n-k}}\ )$, where $\delta_i + \epsilon_i$ is the number of elements are popped when $A[i]$ is pushed into the min or max stack of A, for $1 \leq i \leq n-k$ (see Figure 1 for an example). The encodings of B'_{min} and B'_{max} are defined on $D'_{CMin(A)}$, $D'_{CMax(A')}$ and T', similar to B_{min} and B_{max} in E. Then the size of T' is at most $2n-k$ bits, size of U' is $n-k-1$ bits, and the size of the encodings of the modified B'_{min} and B'_{max} is $o(n)$ bits, and all the other auxiliary structures use $o(n)$ bits. Although we use E' instead of E, we can use the decoding algorithm in E without any modifications because all the properties used in the algorithm still hold even though T' has additional open parentheses compared to T. Therefore from E' we can reconstruct any $\lg n$ consecutive bits of $D'_{CMin(A)}$ or $D'_{CMax(A)}$ in constant time, and thus we can support rank and select on these strings in constant time with $o(n)$ additional structures by Lemma 1.

We use the following auxiliary structures to decode $D_{CMin(A)}$ from $D'_{CMin(A)}$ and C. For this, we first define a correspondence between $D_{CMin(A)}$ and $D'_{CMin(A)}$ as follows. Note that both $D_{CMin(A)}$ and $D'_{CMin(A)}$ have the same number of open parentheses, but $D'_{CMin(A)}$ has fewer close parentheses than $D_{CMin(A)}$. The ith open parenthesis in $D_{CMin(A)}$ corresponds to the ith open parenthesis in $D'_{CMin(A)}$. Suppose there are ℓ and ℓ' close parentheses between the ith and the $(i+1)$st open parentheses in $D_{CMin(A)}$ and $D'_{CMin(A)}$ respectively. Then the last ℓ' close parentheses in $D_{CMin(A)}$ correspond, in that order, to the ℓ' close parentheses in $D'_{CMin(A)}$; the remaining close parentheses in $D_{CMin(A)}$ do not have a corresponding position in $D'_{CMin(A)}$.

We construct three bit strings P_{min}, Q_{min} and R_{min} of lengths $2n-k$, $\lceil 2n/\lg n \rceil$ and $\lceil 2n/\lg n \rceil$, respectively, as follows. For $1 \leq i \leq \lceil 2n/\lg n \rceil$, if the position $i \lg n$ in $D_{CMin(A)}$ has its corresponding position j in $D'_{CMin(A)}$, then we set $P_{min}[j] = 1$, $Q_{min}[i] = 0$ and $R_{min}[i] = 0$. If position $i \lg n$ in $D_{CMin(A)}$ has no corresponding position in $D'_{CMin(A)}$ but for $k_i > 0$ suppose there is a



i	0	1	2	3	4	5	6	7	8	9	10	11	12
A[i]	$-\infty(\infty)$	3	8	5	6	6	6	3	2	2	7	10	9
$D'_{Min(A)}$	((((()	(()	)	((()	)			)	()		(()	)	)
$D'_{Max(A)}$	(((()	)	((((()	)	()			(()	)		)	()	)
T'		()	(((()	(()	)			()	)		()	)	
U'		0	1	0	1			1	0		0	1	
$C_{(uncompressed)}$	0	0	0	0	0	1	1	0	0	1	0	0	0
V_{min}	1101	0		11							0		
V_{max}	100		0110					1					
V			1 1 0 1 0 1 1 0				1 0 0 0 1 1 0 1						

Fig. 1. Data structure combining the colored $2d$-Min heap (A) and colored $2d$-Max heap (B) of A. C is represented in uncompressed form.

leftmost position $q = i \lg n + k_i < (i+1) \lg n$ which has its corresponding position j in $D'_{CMin(A)}$, then we set $P_{min}[j] = 1$, $Q_{min}[i] = 1$ and $R_{min}[i] = 0$. Finally, if all positions between $i \lg n$ and $(i+1) \lg n$ in $D_{CMin(A)}$ have no corresponding position in $D'_{CMin(A)}$ we set $Q_{min}[i] = 1$ and $R_{min}[i] = 1$. In remaining positions for P_{min}, Q_{min} and R_{min}, we set their values as 0. We also store the values, k_i, for $1 \leq i \leq \lceil 2n/\lg n \rceil$, whenever they are defined (as in the second case) explicitly. Since the length of $D_{CMin(A)}$ is $2n$ and $k_i < \lg n$, we can store k_i for all i explicitly using at most $2n \lg \lg n/ \lg n = o(n)$ bits.

Since these bit strings have at most $2n/ \lg n$ 1's, they can be represented using the structure of Lemma 1, taking $o(n)$ bits while supporting rank and select queries in constant time. We define P_{max}, Q_{max}, R_{max} in the same way.

In addition to these bit strings, we construct two pre-computed tables for decoding. To describe these tables, we first define two functions f and f' where $f(s, c)$ ($f'(s, c)$) returns the substring of $D_{CMin(A)}$ of length at most $|s + c|$ whose starting (end) position corresponds to the starting (end) position of s. Suppose s and c are substrings in $D'_{CMin(A)}$ and C respectively. Then we define two functions $f(s, c)$ and $f'(s, c)$ as follows:

$$\begin{cases} f(s,\epsilon) = s \\ f(\epsilon, c) = \epsilon \\ f(s, 1\cdot c_1) =)\cdot f(s, c_1) \\ f((^{\delta}\ )\cdot s_1, 0\cdot c_1) = (^{\delta}\cdot)\cdot f(s_1, c_1) \end{cases} \qquad \begin{cases} f'(s,\epsilon) = s \\ f'(\epsilon, c) = \epsilon \\ f'(s\cdot(^{\delta}\ ), c_1\cdot 1) = f'(s, c_1)\cdot) \\ f'(s_1\cdot(^{\delta}\ ), c_1\cdot 0) = f'(s_1, c_1)\cdot(^{\delta}\cdot) \end{cases}$$

We construct a pre-computed table T_f that, for each possible choice of bit strings s and c of length $1/4 \lg n$, stores the bit string $f(s, c)$. So the total space usage of T_f is $2^{1/4 \lg n} \cdot 2^{1/4 \lg n} \cdot (1/2 \lg n) = o(n)$ bits. We can also construct $T_{f'}$ defined analogous to T_f using $o(n)$ bits.

Now we describe how to decode $\lg n$ consecutive bits of $D_{CMin(A)}$ in constant time using above structures. (We can decode $\lg n$ consecutive bits of $D_{CMax(A)}$ in a similar way.) Suppose we divide $D_{CMin(A)}$ into blocks of size $\lg n$. As described in Section 2.2, it is enough to show that for $1 \le i \le \lceil 2n/\lg n \rceil$, we can decode i-th block of $D_{CMin(A)}$ in constant time. First, we check whether $R_{min}[i]$ is 0 or not. If this value is 0, then the i-th block in $D_{CMin(A)}$ consists of a sequence of $\lg n$ consecutive close parentheses. Otherwise, there are two cases depending on the value of $Q_{min}[i]$. Let $p = select_{P_{min}}(i - rank_{R_{min}}(i,1),1)$ be the position in $D'_{CMin(A)}$ and $c_p = rank_{D'_{CMin(A)}}(p,')')$ if $D'_{CMin(A)}[p] =')'$ and $rank_{D'_{CMin(A)}}(p,')') + 1$ otherwise.

Case 1) $\boldsymbol{Q_{min}[i] = 0}$. In this case, we first take the $\lg n$ consecutive bits of $D'_{CMin(A)}$ and C starting from the positions p and $select_C(c_p, 0)$ respectively. (If the corresponding position in C does not exist, we set the corresponding substring of C as ϵ.) Using these bit strings, we can decode the i-block in $D_{CMin(A)}$ by looking up T_f with these substrings (a constant number of times, until the pre-computed table generates the required $\lg n$ bits). Since the position p corresponds to the starting position of the i-th block in $D_{CMin(A)}$ in this case, we can decode the i-th block of $D_{CMin(A)}$ in constant time.

Case 2) $\boldsymbol{Q_{min}[i] = 1}$. First we decode $\lg n$ consecutive bits of $D_{CMin(A)}$ whose starting position corresponds to the position p using the same procedure as Case 1). Next, we take the $\lg n$ consecutive bits of $D'_{CMin(A)}$ and C ending with the positions p and $select_C(c_p, 0)$ respectively. (If the corresponding position in C does not exist, we set corresponding substring of C as the last $\lg n - 1$ bits of C with extra 0 in the rightmost position.) Then we can decode the $\lg n$ consecutive bits of $D_{CMin(A)}$ whose ending position corresponds to the p by looking up $T_{f'}$ (a constant number of times) with these substrings. By concatenating these two substrings of $D_{CMin(A)}$, we can decode a $2\lg n$-bit substring of $D_{CMin(A)}$ which contains the starting position of the i-th block of $D_{CMin(A)}$ (since the starting position of the i-th block in $D_{CMin(A)}$, and the position which corresponds to p differ by at most $\lg n$). Finally, we can obtain the i-th block in $D_{CMin(A)}$ by jumping the first $\lg n - k_i$ bits and taking $\lg n$ consecutive bits from there.

From the encoding described above, we can decode any $\lg n$ consecutive bits of $D_{CMin(A)}$ and $D_{CMax(A)}$ in constant time. Therefore by Theorem 2, we can answer all queries supported by $CMin(A)$ and $CMax(A)$ (without using the color information) in constant time. If there are k elements such that $A[i-1] = A[i]$ for $1 \le i \le n$, then the size of C is $\lg \binom{n}{k} + o(n)$ bits, and the size of E' on A is $3n - 2k + o(n)$ bits. All other auxiliary bit strings and tables take $o(n)$ bits. Therefore, by the Lemma 2, we can encode A using $3n - 2k + \lg \binom{n}{k} + o(n) \le ((1 + \lg 5)n + o(n) < 3.322n + o(n)$ bits. Also, this encoding supports the queries in Theorem 1(a) which do not need the color information in constant time. This proves Theorem 1(b).

Encoding V_{min} and V_{max}

We simply concatenate V_{max} and V_{min} on A and store it as bitstring V, and store the length of V_{min} using $\lg n$ bits (see V in Figure 1). If there are k elements such that $A[i-1] = A[i]$ for $1 \leq i \leq n$, Fischer and Heun's stack based algorithm [7] does not pop any elements from both stacks when these k elements and $A[n]$ are pushed into them. Before pushing any of the remaining elements in the max and min stacks, we pop some elements from exactly one of the stacks. Also after pushing $A[1]$ into both the stacks, we pop the remaining elements from the stacks in the final step. Supposing the min-stack pops p times during the stack based algorithm (i.e., if the n elements are popped during p runs of pop operations), then each run of pop operation generates exactly one open parenthesis such that $pre\_rank_{D_{CMin(A)}}$ of its matched closed parenthesis corresponds to leftmost child in $CMin(A)$. As described in Section 3, the size of V_{min} is $n-p+1$ bits, and the size of V_{max} is $p+k+1$ bits. Thus the size of V is $n+k+2$ bits.

Therefore, we can decode any $\lg n$-bit substring of V_{min} or V_{max} in constant time using V and the length of V_{min}. By combining these structures with the encoding of Theorem 1(b), we can support queries in Theorem 1(c) in constant time. By Lemma 2, the total space of these structures is $4n - k + \lg \binom{n}{k} + o(n) \leq ((3 + \lg 3)n + o(n) < 4.585n + o(n)$ bits. This proves Theorem 1(d).

4.2 Encoding Colored 2*d*-Min and Max Heap with Less Space

In this section, we give new encodings that prove Theorem 1(a) and Theorem 1(c), which use less space but take more query time than the previous encodings. To prove Theorem 1(a), we maintain the encoding E on A' (instead of the encoding E' on A, used in the previous section). ($f(s,c)$ is well-defined when s and c are substring in $D_{CMin(A')}$ and C respectively.) In addition, we construct the concatenated sequence of V_{min} and V_{max} on A' (not on A, as before). If there are k elements such that $A[i-1] = A[i]$ for $1 \leq i \leq n$, then the size of E on A' is at most $2(n-k) + (n-k) + o(n)$ bits, T_f and compressed C in Theorem 1(b) are also preserved. Therefore the total size of these structures is at most $3(n-k) + \lg \binom{n}{k} + o(n) \leq n \lg 9 + o(n) < 3.167n + o(n)$ bits. If we can reconstruct the sequences $D_{CMin(A)}$ and $D_{CMax(A)}$, by Theorem 2, we can support all the required queries. We now describe how to decode the entire $D_{CMin(A)}$ by this encoding. (Decoding $D_{CMax(A)}$ can be done in a similar way.)

Once we decode the sequence $D_{CMin(A')}$, we reconstruct the sequence $D_{CMin(A)}$ by scanning the sequences $D_{CMin(A')}$ and C from left to right, and using the lookup table T_f. Note that $f(D_{CMin(A')}, C)$ looses some open parentheses in $D_{CMin(A)}$ whose matched close parentheses are not in $D_{CMin(A')}$ but in $f(D_{CMin(A')}, C)$. So when we add m consecutive close parentheses from the r-th position in $D_{CMin(A')}$ in decoding with T_f, we add m more open parentheses before the position $pos = findopen_{D_Min(A')}(r-1)$. This proves Theorem 1(a).

To prove Theorem 1(c), we combine the concatenated sequence of V_{min} and V_{max} on A' whose total size is $n-k+2$ bits to the above encoding. Then we can reconstruct V_{min} on A by adding m 1's before the position

$rank_{D_M in(A')}(pos, `(('))$ in V_{min} on A' when m consecutive close parentheses are added from the r-th position in $D_{CMin(A')}$ in decoding with T_f. (Reconstructing V_{max} on A can be done in a similar way.) The space usage of this encoding is $4(n-k) + \lg \binom{n}{k} + o(n) \leq n \lg 17 + o(n) < 4.088n + o(n)$ bits. This proves Theorem 1(c).

5 Conclusion

We obtained space-efficient encodings that support a large set of range and previous/next smaller/larger value queries. The encodings that support the queries in constant time take more space than the ones that do not support the queries in constant time.

We conclude with the following open problems.

– Supporting queries in the Theorem 1(c) in constant time with $4.088n + o(n)$-bits data structure
– As described in the Section 2, Gawrychowski and Nicholson [8] show that any encoding that supports both $RMinQ_A$ and $RMaxQ_A$ cannot use less than $3n - \Theta(\lg n)$ bits. This lower bound holds even if we assume that there are no consecutive equal elements in A. In the general case, can we improve the lower bound (or, is this bound tight)?
– What is the lower bound of space for encoding that supports queries in the Theorem 1(c)?

References

1. Arroyuelo, D., Cánovas, R., Navarro, G., Sadakane, K.: Succinct trees in practice. In: ALENEX 2010, Austin, Texas, USA, January 16, 2010, pp. 84–97 (2010)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proceedings of the LATIN 2000, pp. 88–94 (2000)
3. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica **43**(4), 275–292 (2005)
4. Farzan, A., Munro, J.I.: A uniform paradigm to succinctly encode various families of trees. Algorithmica **68**(1), 16–40 (2014)
5. Fischer, J.: Combined data structure for previous- and next-smaller-values. Theor. Comput. Sci. **412**(22), 2451–2456 (2011)
6. Fischer, J., Heun, V.: Finding range minima in the middle: Approximations and applications. Mathematics in Computer Science **3**(1), 17–30 (2010)
7. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM Journal on Computing **40**(2), 465–492 (2011)
8. Gawrychowski, P., Nicholson, P.K.: Optimal encodings for range min-max and top-k (2014). CoRR abs/1411.6581, http://arxiv.org/abs/1411.6581

9. Golin, M., Iacono, J., Krizanc, D., Raman, R., Rao, S.S.: Encoding 2D range maximum queries. In: Asano, T., Nakano, S., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 180–189. Springer, Heidelberg (2011)
10. Jansson, J., Sadakane, K., Sung, W.K.: Ultra-succinct representation of ordered trees with applications. J. Comput. Syst. Sci. **78**(2), 619–631 (2012)
11. Miltersen, P.B.: Cell probe complexity - a survey. In: FSTTCS (1999)
12. Munro, J.I., Raman, V., Rao, S.S.: Space efficient suffix trees. J. Algorithms **39**(2), 205–222 (2001)
13. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM Journal on Computing **31**(3), 762–776 (2001)
14. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms **3**(4) (2007). Article 43

A New Non-Merkle-Damgård Structural Hash Function with Provable Security

Shenghui Su[1,2(✉)], Tao Xie[3], and Shuwang Lü[4]

[1] Laboratory of Trusted Computing, Beijing University of Technology, Beijing 100124, People's Republic of China
reesse@126.com
[2] College of Information Engineering, Yangzhou University, Yangzhou 225009, People's Republic of China
[3] School of Computers, National University of Defense Technology, Changsha 410073, People's Republic of China
[4] School of Computers, University of Chinese Academy of Sciences, Beijing 100039, People's Republic of China

Abstract. To check the integrity of an IP address efficiently and economically, the authors propose a new non-Merkle-Damgård structural hash function which is based on a multivariate permutation problem and an anomalous subset product problem to which no subexponential time solutions are found so far. It includes an initialization algorithm and a compression algorithm, and converts a short message of n bits treated as only a block into a digest of m bits, where $80 \le m \le 232$ and $80 \le m \le n \le 4096$. Analysis shows that the new hash is one-way, weakly collision-free, and strongly collision-free along with a proof, and its security against existent attacks such as birthday attack and meet-in-the-middle attack is to $O(2^m)$. In comparison with the Chaum-Heijst-Pfitzmann hash based on a discrete logarithm problem, the new hash is lightweight, and opens a door to convenience for utilization of lightweight digital signing schemes.

Keywords: Hash function · Compression algorithm · Merkle-Damgård structure · Provable security · Birthday attack · Meet-in-the-middle attack

1 Introduction

A message digest from a hash function may be used to check the integrity of IP addresses in a packet so as to prevent the addresses from being tampered.

Traditionally, a hash function consists of a compression function and the Merkle-Damgård (MD) structure [1][2]. Let $\hat{h}$ be a hash function, and generally, it should be computationally one-way, weakly collision-free, and strongly collision-free [3][4][5].

In this paper, the authors design a new non-MD structural hash function called JUNA which is based on a multivariate permutation problem (MPP) and an anomal-

S. Su—This work is supported by MOST with Project 2007CB311100 and 2009AA01Z441.

ous subset product problem (ASPP) [6], and includes two algorithms: an initialization algorithm and a compression algorithm, converts a short message or a message digest of n bits into an output string of m bits, where $80 \le m \le 232$ and $80 \le m \le n \le 4096$, and moreover ensures that the security of the output against collision attacks is to the $O(2^m)$ magnitude.

The new hash is efficient and economical in the check has two dominant novelties: ① designing the initialization algorithm based on a MPP which only has an exponential time solution currently, and makes the new hash function be able to resist a birthday attack; ② designing the compression algorithm based on an ASPP which also only has an exponential time solution currently, and makes the new hash function be able to resist other conventional attacks, especially a meet-in-the-middle attack. The significance of the paper lies in the thing that a new non-MD structural hash function with an m-bit output and the $O(2^m)$ magnitude security is first proposed by the authors while a classical iterative hash function with an m-bit output bears only the $O(2^{m/2})$ magnitude security.

Throughout the paper, unless otherwise specified, an even number $n \ge 80$ is the bit-lengthof a short message or the item-length of a sequence, the sign % denotes "modulo", $\bar{M}$ does "$M - 1$" with M prime, $\lg x$ means a logarithm of x to the base 2, $\neg b_i$ does NOT operation of a bit b_i, $\boldsymbol{\mathit{Þ}}$ does the maximal prime allowed in a coprime sequence, $\lfloor x \rfloor$ does the absolute value of a number x, $\|x\|$ does the order of x % M, $¦S¦$ does the size of a set S, and $\gcd(x, y)$ represents the greatest common divisor of two integers x and y. Without ambiguity, "% M" is usually omitted in expressions.

2 Several Definitions

2.1 A Coprime Sequence

Definition 1: If $A_1, \ldots, A_n$ are n pairwise distinct positive integers such that $\forall\ A_i$ and A_j $(i \ne j)$, either $\gcd(A_i, A_j) = 1$ or $\gcd(A_i, A_j) = F \ne 1$ with $(A_i / F) \nmid A_k$ and $(A_j / F) \nmid A_k$ $\forall\ k\ (\ne i, j) \in [1, n]$, these ordered integers are called a coprime sequence, denoted by $\{A_1, \ldots, A_n\}$, and shortly $\{A_i\}$.

Note that the elements of a coprime sequence are not necessarily pairwise coprime, but a sequence of which the elements are pairwise coprime is a coprime sequence.

For example, {21, 15, 29, 23, 11, 17, 19, 13} and {23, 7, 11, 3, 19, 13, 5, 17} are two coprime sequences separately.

Property 1: Let $\{A_1, \ldots, A_n\}$ be a coprime sequence. If randomly select $k \in [1, n]$ elements $A_{x_1}, \ldots, A_{x_k}$ from the sequence, then the mapping from a subset $\{A_{x_1}, \ldots, A_{x_k}\}$ to a subset product $G = \prod_{i=1}^{k} A_{x_i}$ is one-to-one, namely the mapping from $b_1 \ldots b_n$ to $G = \prod_{i=1}^{n} A_i^{b_i}$ is one-to-one, where $b_1 \ldots b_n$ is a bit string.

Refer to [6] for its proof.

2.2 A Bit Shadow and a Bit Long-Shadow

Definition 2: Let $b_1...b_n \neq 0$ be a bit string. Then $\underset{\cdot}{b}_i$ with $i \in [1, n]$ is called a bit shadow if it comes from such a rule: ① $\underset{\cdot}{b}_i = 0$ if $b_i = 0$; ② $\underset{\cdot}{b}_i = 1$ + the number of successive 0-bits before b_i if $b_i = 1$; or ③ $\underset{\cdot}{b}_i = 1$ + the number of successive 0-bits before b_i + the number of successive 0-bits after the rightmost 1-bit if b_i is the leftmost 1-bit.

Notice that (3) of this definition is slightly different from that in [6].

For example, let $b_1...b_8 = 01010110$, then $\underset{\cdot}{b}_1...\underset{\cdot}{b}_8 = 03020210$.

Fact 1: Let $\underset{\cdot}{b}_1...\underset{\cdot}{b}_n$ be the bit shadow string of $b_1...b_n \neq 0$. Then there is $\sum_{i=1}^{n} \underset{\cdot}{b}_i = n$.

Its proof is omitted due to limited pages.

Property 2: Let $\{A_1, ..., A_n\}$ be a coprime sequence, and $\underset{\cdot}{b}_1...\underset{\cdot}{b}_n$ be the bit shadow string of $b_1...b_n \neq 0$. Then the mapping from $b_1...b_n$ to $G = \prod_{i=1}^{n} A_i^{\underset{\cdot}{b}_i}$ is one-to-one [7].

Its proof is omitted due to limited pages.

Definition 3: Let $\underset{\cdot}{b}_1...\underset{\cdot}{b}_n$ be the bit shadow string of $b_1...b_n \neq 0$. Then $\bar{b}_i = \underset{\cdot}{b}_i 2^{\partial_i}$ with $i \in [1, n]$ is called a bit long-shadow, where $\partial_i = b_{i+(-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)} = 0$ or 1.

According to Definition 3, it is not difficult to understand that for every $\bar{b}_i$, there is $0 \le \bar{b}_i \le n$ when $b_1...b_n \neq 0$.

For example, let $b_1...b_8 = 01010110$, then $\bar{b}_1...\bar{b}_8 = 06020410$.

Fact 2: Let $\bar{b}_1...\bar{b}_n$ be the bit long-shadow string of $b_1...b_n \neq 0$. Then there is $n \le \sum_{i=1}^{n} \bar{b}_i \le 2n$.

Its proof is omitted due to limited pages.

Property 3: Let $\bar{b}_1...\bar{b}_n$ be the bit long-shadow string of $b_1...b_n \neq 0$. Then the mapping from $b_1...b_n$ to $\bar{b}_1...\bar{b}_n$ is one-to-one.

Its proof is omitted due to limited pages.

2.3 A Lever Function

The designing of the initialization algorithm of the new hash function is based on the hard problem $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ ($\%$ M) for $i = 1, ..., n$ first used for the REESSE1+ asymmetric cryptosystem, where the exponent $\ell(i)$ is called a lever function [6].

Definition 4: The secret parameter $\ell(i)$ in the transform of a non-MD structural hash function is called a lever function, if it has the following features:

① $\ell(.)$ is an injection from the domain $\{1,...,n\}$ to the codomain $\Omega \subset \{5,...,\bar{M}\}$ with $\bar{M}$ large;

② the mapping between i and $\ell(i)$ is established randomly without an analytical expression;

③ an attacker has to be faced with all the permutations of elements in Ω when inferring a related private parameter from a public parameter or an initial value;

④ the owner of the private parameter only need to consider the polynomial arithmetic of elements in Ω when decrypting a ciphertext or seeking a collision.

Feature ③ and ④ make it clear that if n is large enough, it is infeasible for the attacker to search all the permutations of elements in Ω exhaustively while the decryption or collision computation by the owner of the private parameter is feasible. Thus, the amount of calculation on $\ell(.)$ is large at "a public terminal", and is small at "a private terminal".

Property 4 **(Indeterminacy of $\ell(.)$):** Let $\delta = 1$ and $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $\ell(i) \in \Omega = \{5, \ldots, n + 4\}$ and $A_i \in \Lambda = \{2, \ldots,$ Ᵽ $\mid 863 \leq$ Ᵽ $\leq 1201\}$ for $i = 1, \ldots, n$. Then $\forall W$ ($\|W\| \neq \bar{M}$) $\in (1, \bar{M})$, and $\forall\, x, y, z$ ($x \neq y \neq z$) $\in [1, n]$,

① when $\ell(x) + \ell(y) = \ell(z)$, there is $\ell(x) + \|W\| + \ell(y) + \|W\| \neq \ell(z) + \|W\|$ (% $\bar{M}$);

② when $\ell(x) + \ell(y) \neq \ell(z)$, there always exist $C_x \equiv A'_x W'^{\ell'(x)}$ (% M), $C_y \equiv A'_y W'^{\ell'(y)}$ (% M), and $C_z \equiv A'_z W'^{\ell'(z)}$ (% M) such that $\ell'(x) + \ell'(y) \equiv \ell'(z)$ (% $\bar{M}$) with $A'_z \leq$ Ᵽ.

Its proof is omitted due to limited pages.

Notice that letting $\Omega = \{5, \ldots, n + 4\}$, namely every $\ell(i) \geq 5$ makes seeking W from $W^{\ell(i)} \equiv A_i^{-1} C_i$ (% M) face an unsolvable Galois group when the value of $A_i \leq$ Ᵽ is guessed [8], and moreover Property 4 still holds when Ω is any subset containing n elements from $\{1, \ldots, \bar{M}\}$.

Property 4 manifests that will continued fraction attack on $C_i \equiv A_i W^{\ell(i)}$ (% M) by Theorem 12.19 in Section 12.3 of [9] be utterly ineffectual only if elements in Ω are fitly selected [10].

3 Design of the New Non-MD Structural Hash Function

The Chaum-Heijst-Pfitzmann hash function, a non-MD structural one, is appreciable. It is based on a discrete logarithm problem, and proved to be strongly collision-free [11].

The new non-MD structural hash function is composed of two algorithms which contain two main parameters m and n, where m denotes the bit-length of a modulus used in the new hash, n denotes the bit-length of a short message or a message digest from a classical hash function, and there are $80 \leq m \leq 232$ with $80 \leq m \leq n \leq 4096$.

Additionally, Λ and Ω are two integral sets. Their lengths are selected as $2^{10} \leq |\Lambda| \leq 2^{32}$ and $n \leq |\Omega| = \tilde{n} \leq 2^{32}$, and moreover make $2n^5|\Omega||\Lambda|^5 \geq 2^m$ (see Section 4.1.1). Notice that $2^{10} \leq |\Lambda| \leq 2^{32}$ means $10 \leq \lceil \lg$ Ᵽ $\rceil \leq 32$.

For example, as $m = 80 \leq n$, there should be $|\Lambda| = 2^{10}$ and $|\Omega| = n$; as $m = 96 \leq n$, should $|\Lambda| = 2^{12}$ and $|\Omega| = n$; as $m = 112 \leq n$, should $|\Lambda| = 2^{14}$ and $|\Omega| = n$; as $m = 128 \leq n$, should $|\Lambda| = 2^{16}$ and $|\Omega| = 2^{12}$; as $m = 232 \leq n$, should $|\Lambda| = 2^{32}$ and $|\Omega| = 2^{32}$.

3.1 Initialization Algorithm

This algorithm is employed by an authoritative third party or the owner of a key pair, and only needs to be executed one time.

```
INPUT: the bit-length $m$ of a modulus with $80 \leq m \leq 232$;
       the item-length $n$ of a sequence with $80 \leq m \leq n \leq 4096$;
       the maximal prime Ᵽ with $10 \leq \lceil \lg Ᵽ \rceil \leq 32$;
       the size $\tilde{n}$ of the set $\Omega$ with $2\tilde{n}n^5 Ᵽ^5 \geq 2^m$ and $n \leq \tilde{n} \leq 2^{32}$.
S1: Produce $\Lambda \leftarrow \{2, 3, \ldots, Ᵽ\}$;
    produce a random coprime sequence $\{A_1, \ldots, A_n \mid A_i \in \Lambda\}$.
S2: Find a prime $M$ with $\lceil \lg M \rceil = m$ such that $\bar{M} / 2$ is a prime,
    or the least prime factor of $\bar{M} / 2 > 4n(2\tilde{n} + 3)$.
S3: Pick $W \in (1, \bar{M})$ making $\|W\| \geq 2^{m - \lceil \lg Ᵽ \rceil}$; pick $\delta \in (1, \bar{M})$ making $\gcd(\delta, \bar{M}) = 1$.
S4: Randomly yield $\Omega \leftarrow \{+/-5, +/-7, \ldots, +/-(2\tilde{n} + 3)\}$;
    randomly select pairwise distinct $\ell(i) \in \Omega$ for $i = 1, \ldots, n$.
```

```
S5: Compute $C_i \leftarrow (A_i W^{\ell(i)})^{\delta}$ % $M$ for $i = 1, \ldots, n$.
OUTPUT: an initial value $(\{C_i\}, M)$ which is public to the people.
```

A private parameter $(\{A_i\}, \{\ell(i)\}, W, \delta)$ may be discarded, but must not be divulged.

At S3, to seek W, let $W \equiv g^{\bar{M}/F}$ (% M), where g is a generator of $(\mathbb{Z}_M^*, \cdot)$ obtained through Algorithm 4.80 in Section 4.6 of [3], and $F < 2^{\lceil \lg \bar{P} \rceil}$ is a factor of $\bar{M}$.

At S4, $\Omega = \{+/-5, +/-7, \ldots, +/-(2\tilde{n} + 3)\}$ indicates that Ω is one of $2^{\tilde{n}}$ potential sets, indeterminate, and unknown to the public, where "+/–" means the selection of the "+" or "–" sign. Notice that in the arithmetic modulo $\bar{M}$, $-x$ represents $\bar{M} - x$.

Definition 5: Given $(\{C_i\}, M)$, seeking the original $(\{A_i\}, \{\ell(i)\}, W, \delta)$ from $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $A_i \in \{2, 3, \ldots, \bar{P} \mid 10 \le \lceil \lg \bar{P} \rceil \le 32\}$ and $\ell(i) \in \{+/-5, +/-7, \ldots, +/-(2\tilde{n} + 3) \mid n \le \tilde{n} \le 2^{32}\}$ for $i = 1, \ldots, n$ is referred to as a multivariate permutation problem, shortly MPP [6].

Property 5: The MPP $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $A_i \in \{2, 3, \ldots, \bar{P} \mid 10 \le \lceil \lg \bar{P} \rceil \le 32\}$ and $\ell(i) \in \{+/-5, +/-7, \ldots, +/-(2\tilde{n} + 3) \mid n \le \tilde{n} \le 2^{32}\}$ for $i = 1, \ldots, n$ is computationally at least equivalent to the DLP in the same prime field.

Refer to [6] for its proof.

3.2 Compression Algorithm

This algorithm is employed by one who wants to obtain a short message digest.

```
INPUT: an initial value $(\{C_1, \ldots, C_n\}, M)$, where $\lceil \lg M \rceil = m$ with $80 \le m \le n \le 4096$;
       A short message (or a digest from a classical hash function) $b_1 \ldots b_n \ne 0$.
  S1: Set $k \leftarrow 0$, $i \leftarrow 1$.
  S2: If $b_i = 0$ then
        S2.1: let $k \leftarrow k + 1$, $\underset{.}{b}_i \leftarrow 0$
      else
        S2.2: if $i = k + 1$ then let $\bar{s} \leftarrow i$;
        S2.3: let $\underset{.}{b}_i \leftarrow k + 1$, $k \leftarrow 0$.
  S3: Let $i \leftarrow i + 1$; if $i \le n$ then go to S2.
  S4: Compute $\underset{.}{b}_{\bar{s}} \leftarrow \underset{.}{b}_{\bar{s}} + k$.
  S5: Compute $\underline{d} \leftarrow \prod_{i=1}^{n} C_i^{\bar{b}_i}$ % $M$, where $\bar{b}_i = \underset{.}{b}_i 2^{\partial_i}$ with $\partial_i = b_{i + (-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$.
  OUTPUT: a digest $\underline{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ (% $M$) of which the bit-length is $m$.
```

It is easily known from Definition 3 that the max of $\{\bar{b}_1, \ldots, \bar{b}_n\}$ is less than or equal to n when $b_1 \ldots b_n \ne 0$.

Definition 6: Given $(\underline{d}, M)$, seeking the original $\bar{b}_1 \ldots \bar{b}_n$ from $\underline{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ (% M), where $\bar{b}_i = \underset{.}{b}_i 2^{\partial_i}$ with $\partial_i = b_{i + (-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$ and $\underset{.}{b}_i$ being a bit shadow is referred to as an anomalous subset product problem, shortly ASPP [6].

Property 6: The ASPP $\underline{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ (% M), where $\bar{b}_i = \underset{.}{b}_i 2^{\partial_i}$ with $\partial_i = b_{i+(-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$ and $\underset{.}{b}_i$ being a bit shadow is computationally at least equivalent to the DLP in the same prime field.

Refer to [6] for its proof.

4 Security Analysis of the New Hash Function

It is should be noted that $\lceil \lg M \rceil = m$, but not n, is the security dominant parameter of the new non-MD structural hash function.

4.1 Security of the Initialization Algorithm

Clearly, the security of the initialization algorithm depends on the security of the MPP $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $A_i \in \Lambda = \{2, 3, \ldots, Ƥ \mid 10 \le \lceil \lg Ƥ \rceil \le 32\}$ and $\ell(i) \in \Omega = \{+/-5, +/-7, \ldots, +/-(2\tilde{n} + 3) \mid n \le \tilde{n} \le 2^{32}\}$ for $i = 1, \ldots, n$.

In [6], we analyze the security of the MPP $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $A_i \in \{2, 3, \ldots, Ƥ \mid 863 \le Ƥ \le 1201\}$ and $\ell(i) \in \{5, 7, \ldots, (2n + 3)\}$ for $i = 1, \ldots, n$ from the three aspects, discover no subexponential time solution to it, and contrarily, find some evidence which inclines people to believe that the MPP is computationally harder than the DLP.

Considering that the set Ω is different from the old in [6], and the range of $Ƥ$ is larger than the old in [6], we will analyze the security of the MPP with different restrictions additionally.

4.1.1 Ineffectualness of Presupposing $\ell(x_1) + \ell(x_2) = \ell(y_1) + \ell(y_2)$

Because of $\Omega = \{+/-5, +/-7, \ldots, +/-(2\tilde{n} + 3)\}$, when the absolute values $|\ell(x_1)|$, $|\ell(x_2)|$, $|\ell(y_1)|$, $|\ell(y_2)|$ are determined, the value $\ell(x_1) + \ell(x_2) - (\ell(y_1) + \ell(y_2))$ has $2^4 = 16$ possible cases, which enhances the indeterminacy of the lever function, and increases the complexity of an attack task for cracking the MPP to some extent.

Adversaries may try to eliminate W through judging $\ell(x_1) + \ell(x_2) = \ell(y_1) + \ell(y_2)$.

$\forall\, x_1, x_2, y_1, y_2 \in [1, n]$, presuppose that $\ell(x_1) + \ell(x_2) = \ell(y_1) + \ell(y_2)$ holds.

Let $G_z \equiv C_{x_1} C_{x_2} (C_{y_1} C_{y_2})^{-1}$ (% M), namely $G_z \equiv (A_{x_1} A_{x_2} (A_{y_1} A_{y_2})^{-1})^{\delta}$ (% M).

If the adversaries divine the values of A_{x_1}, A_{x_2}, A_{y_1}, A_{y_2}, and compute u, v_{x_1}, v_{x_2}, v_{y_1}, v_{y_2} in at least $L_M[1/3, 1.923]$ time such that

$$G_z \equiv g^u,\ A_{x_1} \equiv g^{v_{x_1}},\ A_{x_2} \equiv g^{v_{x_2}},\ A_{y_1} \equiv g^{v_{y_1}},\ A_{y_2} \equiv g^{v_{y_2}}\ (\%\ M),$$

where g is a generator of $(\mathbb{Z}_M^*, \cdot)$, then $u \equiv (v_{x_1} + v_{x_2} - v_{y_1} - v_{y_2})\delta$ (% $\bar{M}$).

If $\gcd(v_{x_1} + v_{x_2} - v_{y_1} - v_{y_2}, \bar{M}) \mid u$, the congruence in δ has solutions. Because each of A_{x_1}, A_{x_2}, A_{y_1}, A_{y_2} may traverse the interval Λ, and the subscripts x_1, x_2, y_1, y_2 are unfixed, the number of potential values of δ is about $n^4 |\Lambda|^4$. Notice that the number of non-repeated values of δ will be less than 2^m.

In succession, need to seek W. Now, the most effectual approach to seeking W is that for every i, the adversaries fix a value of δ, divine A_i and $\ell(i)$, and find the set $\bar{V}_i$ according to $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M), where $\bar{V}_i$ is the set of possible values of W meeting $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) for $i = 1, \ldots, n$. If there exist $W_1 \in \bar{V}_1, \ldots, W_n \in \bar{V}_n$ being pairwise equal, the divination of δ, $\{A_i\}$, and $\{\ell(i)\}$ is thought right; else fix another value of δ, repeat the above process.

Notice that due to $\bar{M} / 2$ = a prime or the least prime factor of $\bar{M} / 2 > 4n(2\tilde{n} + 3)$, $W^{\ell(i)} \equiv C_i^{\delta^{-1}} A_i^{-1}$ (% M) can be solved in polynomial time, and besides letting $W = g^{\mu}$ % M is unnecessary.

It is not difficulty to understand that the size of every $\bar{V}_i$ is about $(2|\Omega|)|\Lambda|$.

In summary, the time complexity of the above attack task is

$$\bar{T} = (n + |\Lambda|)L_M[1 / 3, 1.923] + (n^4|\Lambda|^4) + (n^4|\Lambda|^4)(2|\Omega||\Lambda|)n \approx 2n^5|\Omega||\Lambda|^5.$$

Concretely speaking,

For $m = n = 80$ with $|\Lambda| = 2^{10}$ & $|\Omega| = 80$, $\bar{T} > 2(2^{6.3})^5(2^{6.3})(2^{10})^5 = 2^{88} > 2^m$.

For $m = n = 96$ with $|\Lambda| = 2^{12}$ & $|\Omega| = 96$, $\bar{T} > 2(2^{6.5})^5(2^{6.5})(2^{12})^5 = 2^{100} > 2^m$.

For $m = n = 112$ with $|\Lambda| = 2^{14}$ & $|\Omega| = 112$, $\bar{T} > 2(2^{6.8})^5(2^{6.8})(2^{14})^5 = 2^{112} = 2^m$.

For $m = n = 128$ with $|\Lambda| = 2^{16}$ & $|\Omega| = 2^{12}$, $\bar{T} > 2(2^7)^5(2^{12})(2^{16})^5 = 2^{128} = 2^m$.

For $m = n = 232$ with $|\Lambda| = 2^{32}$ & $|\Omega| = 2^{32}$, $\bar{T} > 2(2^{7.8})^5(2^{32})(2^{32})^5 = 2^{232} = 2^m$.

Thus, the time complexity of the attack by presupposing $\ell(x_1) + \ell(x_2) = \ell(y_1) + \ell(y_2)$ is not less than $O(2^m)$ when $|\Lambda|$ and $|\Omega|$ are chosen suitably.

4.1.2 Ineffectualness of Guessing $\|W\|$

Owing to $80 \le \lceil \lg M \rceil \le 232$, $\bar{M}$ can be factorized in tolerable subexponential time, and further a value of $\|W\|$ can be guessed.

Adversaries may try to eliminate W through $W^{\|W\|} \equiv 1$ (% M).

Raising either side of every equation $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) to the $\|W\|$-th power yields

$$C_i^{\|W\|} \equiv (A_i)^{\delta \|W\|} \% M.$$

Suppose that the value of every $A_i \in \Lambda = \{2, 3, \ldots, \bar{P} \mid 10 \le \lceil \lg \bar{P} \rceil \le 32\}$ is guessed, or the possible values of every A_i are traversed.

Let $C_i \equiv g^{u_i}$ (% M), and $A_i \equiv g^{v_i}$ (% M), where g is a generator of $(\mathbb{Z}_M^*, \cdot)$. Then

$$u_i\|W\| \equiv v_i\|W\|\delta \; (\% \; \bar{M}) \; (i = 1, \ldots, n).$$

Notice that $u_i \ne v_i \delta$ (% $\bar{M}$), and $\{v_1, \ldots, v_n\}$ is not a super increasing sequence.

The above congruence is seemingly the MH transform [12]. Actually, $\{v_1\|W\|, \ldots, v_n\|W\|\}$ is not a super increasing sequence, and moreover there is not necessarily $\lceil \lg(u_i \|W\|) \rceil = \lceil \lg \bar{M} \rceil$.

Because $v_i\|W\| \in [1, \bar{M}]$ is stochastic, the inverse δ^{-1} % $\bar{M}$ not need be close to the minimum $\bar{M} / (u_i\|W\|)$, $2\bar{M} / (u_i\|W\|)$, …, or $(u_i\|W\| - 1)\bar{M} / (u_i\|W\|)$. Namely δ^{-1} may lie at any integral position of the interval $[k\bar{M} / (u_i\|W\|), (k + 1)\bar{M} / (u_i\|W\|)]$, where $k = 0, 1, \ldots, u_i\|W\| - 1$, which illustrates that the accumulation points of minima do not exist. Further observing, in this case, when i traverses the interval $[2, n]$, the number of intersections of the intervals containing δ^{-1} is likely the max of $\{u_1\|W\|, \ldots, u_n\|W\|\}$ which is promisingly close to $\bar{M}$. Therefore, the Shamir attack by the accumulation point of minima is fully ineffectual [13].

Even if find out δ^{-1} through the Shamir attack method, because each of $\{v_1, \ldots, v_n\}$ has $\|W\|$ solutions, the number of potential sequences $\{g^{v_1}, \ldots, g^{v_n}\}$ is up to $\|W\|^n$. Because of needing to verify whether $\{g^{v_1}, \ldots, g^{v_n}\}$ is a coprime sequence for each different sequence $\{v_1, \ldots, v_n\}$, the number of possible coprime sequences is in proportion to $\|W\|^n$. Hence, the initial $\{A_1, \ldots, A_n\}$ cannot be determined in subexponential time. Further, the value of W cannot be computed, and the values of $\|W\|$ and δ^{-1} cannot be verified, which indicates that the MPP can also be resistant to the Shamir attack by the accumulation point of minima.

Additionally, the adversaries may divine the value of A_i in about $O(¦\Lambda¦)$ time with $i \in [1, n]$, and compute δ by $v_i\|W\| \equiv u_i\|W\|\delta$ (% $\bar{M}$). However, because of $\|W\| \mid \bar{M}$, the equation will have $\|W\|$ solutions. Therefore, the time complexity of finding the original δ is at least

$$Ŧ = (n + ¦\Lambda¦)L_M[1 / 3, 1.923] + ¦\Lambda¦\|W\| \geq (n + ¦\Lambda¦)L_M[1/3, 1.923] + 2^{\lceil \lg Ƥ \rceil} 2^{m - \lceil \lg Ƥ \rceil} > 2^m.$$

It is also not less than $O(2^m)$.

4.2 Security of the Compression Algorithm

The compression algorithm of which the input message is treated as only a block is the main body of the new non-MD structural hash function, and thus, through it the four natural properties of the new hash function are embodied dominantly.

Clearly, the security of the compression algorithm depends on the security of the ASPP $\underline{d} \equiv \prod_{i=1}^{n} C_i^{ƀ_i}$ (% M), where $ƀ_i = ḅ_i 2^{\partial_i}$ with $\partial_i = b_{i + (-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$ and $ḅ_i$ being a bit shadow.

In [6], we analyze the security of the ASPP $\bar{G} \equiv \prod_{i=1}^{n} C_i^{ḅ_i}$ (% M) from the three aspects, discover no subexponential time solution to it, and contrarily, find some evidence which inclines people to believe that $\bar{G} \equiv \prod_{i=1}^{n} C_i^{ḅ_i}$ (% M) is computationally harder than the DLP. Due to $ƀ_i = ḅ_i 2^{\partial_i} \geq ḅ_i$, the security conclusion about $\bar{G} \equiv \prod_{i=1}^{n} C_i^{ḅ_i}$ (% M) is also suitable for $\underline{d} \equiv \prod_{i=1}^{n} C_i^{ƀ_i}$ (% M) which is just another form of the ASPP. Hence $\underline{d} \equiv \prod_{i=1}^{n} C_i^{ƀ_i}$ (% M) has no subexponential time solution at present.

In what follows, we will analyze whether the compression formula $\underline{d} \equiv \prod_{i=1}^{n} C_i^{ƀ_i}$ (% M) satisfies the four natural properties of a hash function, and especially resists the three classical attacks or not.

In terms of Section 3.2, given the initial value $(\{C_i\}, M)$ and a short message $b_1 \ldots b_n$, it is transparently easy to calculate the digest $\underline{d} \equiv \prod_{i=1}^{n} C_i^{ƀ_i}$ (% M).

4.2.1 Compression Algorithm is Computationally One-way

Let $C_1 \equiv g^{u_1}$ (% M), …, $C_n \equiv g^{u_n}$ (% M), $\underline{d} \equiv g^{v}$ (% M), where g is a generator of the group $(\mathbb{Z}_M^*, \cdot)$, and easily found when $\lceil \lg M \rceil < 1024$.

Then, solving $\underline{d} \equiv \prod_{i=1}^{n} C_i^{ƀ_i}$ (% M) for $ƀ_1 \ldots ƀ_n$, namely $b_1 \ldots b_n$, is equivalent to solving

$$ƀ_1 u_1 + \ldots + ƀ_n u_n \equiv v \text{ (\% } \bar{M}\text{)},$$

which is called an anomalous subset sum problem, shortly ASSP [6], and computationally at least equivalent to a subset sum problem (SSP) due to $ƀ_i = ḅ_i 2^{\partial_i} \geq ḅ_i \geq b_i \in [0, 1]$.

The SSP has been proved to be NP-complete in its feasibility recognition form [14], and its computational version, especially the density-high or length-big one, is NP-hard [3][15]. Hence, solving ASSP is at least NP-hard.

Moreover in the non-MD structural hash function, there is $n \geq m = \lceil \lg M \rceil$ and $n \geq \bar{b}_i \geq b_i \in [0, 1]$. The knapsack density relevant to the ASSP $\bar{b}_1 u_1 + \ldots + \bar{b}_n u_n \equiv v$ ($\%$ $\bar{M}$) roughly equals

$$D = \textstyle\sum_{i=1}^{n} \lceil \lg n \rceil / \lceil \lg M \rceil = n \lceil \lg n \rceil / m > \lceil \lg n \rceil > 1,$$

which means that there exists many solutions to $\bar{b}_1 u_1 + \ldots + \bar{b}_n u_n \equiv v$ ($\%$ $\bar{M}$), namely the original solution cannot be determined, or will not occur in a reduced lattice base defined by LLL [16]. Notice that only such a $\langle \bar{b}_1, \ldots, \bar{b}_n \rangle$ from which a right bit string can be deduced will be a reasonable solution vector. Experiments show that when $D > 1$, the probability that the original solution or a reasonable solution is found through LLL lattice base reduction is almost zero [17].

Hence, LLL lattice base reduction attack on ASSP [16][18] is utterly ineffectual, which illustrates that even although a DLP with the modulus bit-length less than 1024 can be solved, the original or a reasonable $\bar{b}_1 \ldots \bar{b}_n$ cannot be found yet in DLP subexponential time, namely $\underline{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ ($\%$ M) is computationally one-way.

4.2.2 Compression Algorithm is Weakly Collision-Free

Assume that $b_1 \ldots b_n \neq 0$ is a short message or a message digest from a classical hash function. By Definition 3, we easily understand that $\bar{b}_i = \underset{.}{b}_i 2^{\partial_i} \leq n$ $\forall i \in [1, n]$.

Given a short message $b_1 \ldots b_n \neq 0$, and let $b'_1 \ldots b'_n \neq 0$ be another short message to need to be found.

Let $\bar{b}_1 \ldots \bar{b}_n$ be the bit long-shadow string of $b_1 \ldots b_n$, and $\bar{b}'_1 \ldots \bar{b}'_n$ be the bit long-shadow string of $b'_1 \ldots b'_n$.

Let $l\hat{h}$ be the compression algorithm of the new non-MD structural hash function described in Section 3.2. Hence, we have

$$\underline{d} = l\hat{h}(b_1 \ldots b_n) = \textstyle\prod_{i=1}^{n} C_i^{\bar{b}_i} \ \% \ M, \text{ and } \underline{d}' = l\hat{h}(b'_1 \ldots b'_n) = \textstyle\prod_{i=1}^{n} C_i^{\bar{b}'_i} \ \% \ M,$$

where $\bar{b}_i = \underset{.}{b}_i 2^{\partial_i}$ with $\partial_i = b_{i + (-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$, and $\bar{b}'_i = \underset{.}{b}'_i 2^{\partial'_i}$ with $\partial'_i = b'_{i + (-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$.

If $\underline{d} = \underline{d}'$, there is $\prod_{i=1}^{n} C_i^{\bar{b}_i} \equiv \prod_{i=1}^{n} C_i^{\bar{b}'_i}$ ($\%$ M).

Observe an extreme case.

Assume that $C_1 = \ldots = C_n = C$.

Owing to the max of $0 \leq \bar{b}_i \leq n$, we define logically $\prod_{i=1}^{n} C^{\bar{b}_i} \equiv \prod_{i=1}^{n} C^{(n+1)^{n-i} \bar{b}_i}$ ($\%$ M).

Under the circumstances, if $\underline{d} = \underline{d}'$, then there is

$\prod_{i=1}^{n} C^{(n+1)^{n-i} \bar{b}_i} \equiv \prod_{i=1}^{n} C^{(n+1)^{n-i} \bar{b}'_i}$ ($\%$ M), namely $C^{\sum_{i=1}^{n} (n+1)^{n-i} \bar{b}_i} \equiv C^{\sum_{i=1}^{n} (n+1)^{n-i} \bar{b}'_i}$ ($\%$ M).

Let $z \equiv \sum_{i=1}^{n} \bar{b}_i (n+1)^{n-i}$ ($\%$ $\bar{M}$), and $z' \equiv \sum_{i=1}^{n} \bar{b}'_i (n+1)^{n-i}$ ($\%$ $\bar{M}$).

Correspondingly, $C^{z} \equiv C^{z'}$ ($\%$ M).

We need to solve the above equation for z'.

If the order $\|C\|$ is known, let $z' = z + k\|C\|$, where $k \geq 1$ is an integer. Once a fit k is found, there will be $C^{z} \equiv C^{z'}$ ($\%$ M), and a bit string can be inferred from $\bar{b}'_1 \ldots \bar{b}'_n$. However, seeking $\|C\|$ is of the integer factorization problem (IFP) at present because the prime factors of $\bar{M}$ must be known.

In practice, $C_1, \ldots, C_n$ that are produced through the algorithm in Section 3.1 are pairwise unequal, which implies that for any given short message $b_1 \ldots b_n$, seeking another short message $b'_1 \ldots b'_n$ such that $\prod_{i=1}^{n} C_i^{\bar{b}_i} \equiv \prod_{i=1}^{n} C_i^{\bar{b}'_i}$ ($\%$ M) is harder than the

IFP in computational complexity, namely $b'_1...b'_n$ for $l\hat{h}(b_1...b_n) = l\hat{h}(b'_1...b'_n)$ cannot be found in IFP subexponential time.

Therefore, we say that the new non-MD structural hash function is weakly collision-free.

4.2.3 Compression Algorithm Is Resistant to Birthday Attack

First, observe an example of whether any two students in a class have the same birthday.

Suppose that the class has 23 students. If a teacher specifies a day (say February 12), then the chance that at least one student is born on that day is $(1 - (364 / 365)^{23}) \approx 6.11\ \%$. However, the probability that at least one student has the same birthday as any other student is around $(1 - (365\times...\times343 / 365^{23})) \approx 50.73\ \%$, which prompts birthday attack on hash functions.

Birthday attack is widely exploited for finding any two messages $\underline{m}$ and $\underline{m}'$ such that $\hat{h}(\underline{m}) = \hat{h}(\underline{m}')$, namely $(\underline{m}, \underline{m}')$ is a collision, where $\hat{h}$ is a hash function [19]. If the bit-length of a message digest is m, an adversary can find a collision $(\underline{m}, \underline{m}')$ such that $\hat{h}(\underline{m}) = \hat{h}(\underline{m}')$ with probability 50% in roughly $1.1774 \times 2^{m/2}$ time, namely with input of $1.1774 \times 2^{m/2}$ random messages [20].

However, to the new non-MD structural hash, a collision is transformed into a mapping.

Theorem 1: The new non-MD structural hash function is resistant to birthday attack on the assumption that the MPP and ASPP have only exponential time solutions.

Its proof is omitted due to limited pages.

4.2.4 Compression Algorithm is Resistant to Meet-in-the-Middle Attack

Meet-in-the-middle dichotomy used for attack on an intended expansion of a block cipher was first developed by Diffie and Hellman in 1977 [21]. Section 3.10 of [3] brings forth a meet-in-the-middle attack algorithm for solving a subset sum problem.

Let $b_1...b_n$ be a short message, and its digest be $\underline{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ (% M).

If $b_{n/2} = b_n = 1$ (thus, any bit *shadow* on the left of the middle point has no relation with bits on the right), an adversary may attempt to attack the ASPP $\underline{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ (% M) by the meet-in-the-middle method.

However, owing to $\bar{b}_i = \underset{\cdot}{b}_i 2^{\partial_i}$ with $\partial_i = b_{i+(-1)^{\lfloor 2(i-1)/n \rfloor}(n/2)}$ for every $i \in [1, n]$, when i is from 1 to $n / 2$, there exists $\bar{b}_1...\bar{b}_{n/2} = (\underset{\cdot}{b}_1 2^{b_{1+n/2}})...(\underset{\cdot}{b}_{n/2} 2^{b_n})$,

which involves all the bits of the short message, namely a reasonable middle point does not exist.

If a fork is selected in proportion to $(n / 3 : 2n / 3)$ or $(n / 4 : 3n / 4)$, the right of the fork substantially still involves all the bits $b_1, ..., b_n$.

For instance, let $n = 12$, a short message (a bit string) $= b_1...b_{12}$, and a fork be to $(4 : 8)$, then

$$\bar{b}_5...\bar{b}_{12} = (\underset{\cdot}{b}_5 2^{b_{11}})(\underset{\cdot}{b}_6 2^{b_{12}})(\underset{\cdot}{b}_7 2^{b_1})(\underset{\cdot}{b}_8 2^{b_2})(\underset{\cdot}{b}_9 2^{b_3})(\underset{\cdot}{b}_{10} 2^{b_4})(\underset{\cdot}{b}_{11} 2^{b_5})(\underset{\cdot}{b}_{12} 2^{b_6})$$

involves all the bits $b_1, ..., b_{12}$.

The above dissection manifests that the meet-in-the-middle attack is essentially ineffectual on the new non-MD structural hash function. Therefore, even if $n = m$, namely the input length = the output length of the function, the time complexity of the attack task is still $O(2^m)$ at present, but not $O(m2^{m/2})$.

Besides, unlike $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} b_i c_i + \sum_{i=1}^{n} \neg b_i c_i$ in the SSP, there is not

$$\prod_{i=1}^{n} C_i = \prod_{i=1}^{n} C_i^{\bar{b}_i} \prod_{i=1}^{n} C_i^{\neg \bar{b}_i} \ (\%\ M)$$

in the ASPP, where $\neg\bar{b}_i$ is the bit long-shadow of $\neg b_i$, which implies there does not exist an easy relation between the ASPP $\bar{d} \equiv \prod_{i=1}^{n} C_i^{\bar{b}_i}$ $(\%\ M)$ and the dichotomy.

4.2.5 Compression Algorithm is Strongly Collision-free

Theorem 2: If any arbitrary collision of the new non-MD structural hash function can be found in subexponential time, the ASPP $\prod_{i=1}^{n} C_i^{\bar{y}_i} \equiv 1$ $(\%\ M)$ can be solved efficiently, where $\bar{y}_i \in [-n, n]$ is the difference of two bit long-shadows at the same position.

Its proof is omitted due to limited pages.

5 Comparison with the Chaum-Heijst-Pfitzmann Hash

The Chaum-Heijst-Pfitzmann hash function is provably secure, and defined as follows [11]:

$$\hat{h}: w_1, w_2 \mapsto \hat{h}(w_1, w_2) = \alpha^{w_1} \beta^{w_2} \ \%\ p \quad (\{0, ..., q-1\}^2 \to \mathbb{Z}_p - \{0\}),$$

where w_1 and w_2 are the two complementary parts of a short message, p and q $(= (p - 1)/2)$ are two big primes, and α and β are two generators of the group $(\mathbb{Z}_p^*, \cdot)$. Hence, the Chaum-Heijst-Pfitzmann function based on the difficulty of the DLP $\beta = \alpha^x\ \%\ p$ compresses a short message of $2(\lceil \lg p \rceil - 1)$ bits into a digest of $\lceil \lg p \rceil$ bits.

Let $\lceil \lg p \rceil = 1024$, and then the time complexity of computing $\log_\alpha \beta\ \%\ p$ is 2^{80} according to the subexponential time $L_p[1/3, 1.923]$ [3], which means that the security of the Chaum-Heijst-Pfitzmann hash is the 2^{80} magnitude when $\lceil \lg p \rceil = 1024$.

Let $\lceil \lg M \rceil = 80$, and then the time complexity of solving the ASPP $\bar{d} = \prod_{i=1}^{n} C_i^{\bar{b}_i}\ \%$ M for $\bar{b}_1, \ldots, \bar{b}_n$ is also 2^{80} since the ASPP only has an exponential time solution at present [22], which means that the security of the new non-MD structural hash is also the 2^{80} magnitude when $\lceil \lg M \rceil = 80$. Besides, let the bit-length $n = 2046$ of a short message $(w_1, w_2) = (b_1 \ldots b_{1023}, b_{1024} \ldots b_{2046}) = b_1 \ldots b_n \neq 0$.

Under the same security, may draw a comparison between the new non-MD structural hash (the JUNA hash) and the Chaum-Heijst-Pfitzmann hash.

Table 1. Comparison between two non-MD structural hashes

	Chaum-Heijst-Pfitzmann hash	JUNA hash
Running time (bit operations)	$2(4\lceil \lg p \rceil^3) = 8589934592$	$4nm^2 = 52428800$ bit operations
Compression rate	$1024 / 2046 \approx 50.05\%$	$80 / 2046 \approx 3.91\%$
Resistant to birthday attack	No because the number of (w_1, w_2)'s needed during birthday attack is about $2^{\lceil \lg p \rceil / 2} = 2^{512}$, and larger than 2^{80} which is the security magnitude of the DLP.	Yes because the number of $b_1 \ldots b_n$'s needed during birthday attack is about $2^{\lceil \lg M \rceil / 2} = 2^{40}$, and smaller than 2^{80} which is the security magnitude of the ASPP.
Provably strongly collision-free	Yes on the assumption that a DLP has a subexponential time solution.	Yes on the assumption that an ASPP has an exponential time solution.

In summary, the JUNA hash has some advantages over the Chaum-Heijst-Pfitzmann one, and relatively the JUNA hash may be regarded lightweight.

6 Conclusion

In the paper, the authors propose a new non-MD structural hash function which contains the initialization algorithm and the compression algorithm, and converts a short message or a message digest of n bits into a string of m bits, where $80 \le m \le 232$ and $80 \le m \le n \le 4096$.

The authors analyze the security of the new non-MD structural hash function. The analysis shows that the new non-MD structural hash is computationally one-way, weakly collision-free, and strongly collision-free. Moreover, at present, any subexponential time algorithm for attacking the new non-MD structural hash is not found, and its security is to the $O(2^m)$ magnitude.

Especially, the analysis illustrates that the new non-MD structural hash function is resistant to birthday attack and meet-in-the-middle attack, and that the running time of its compression algorithm is $O(n\,m^2)$ bit operations.

The new non-MD structural hash function also opens a door to convenience for the utilization of a lightweight digital signing scheme of which the modulus length is not greater than 160 bits.

Acknowledgment. The authors would like to thank the Academicians Jiren Cai, Zhongyi Zhou, Jianhua Zheng, Changxiang Shen, Zhengyao Wei, Binxing Fang, Guangnan Ni, Andrew C. Yao, and Xicheng Lu for their important guidance, advice, and suggestions.

The authors also would like to thank the Professors Dingyi Pei, Jie Wang, Ronald L. Rivest, Moti Yung, Adi Shamir, Dingzhu Du, Mulan Liu, Huanguo Zhang, Dengguo Feng, Yixian Yang, Maozhi Xu, Hanliang Xu, Xuejia Lai, Yongfei Han, Yupu Hu, Dongdai Lin, Rongquan Feng, Ping Luo, Jianfeng Ma, Lusheng Chen, Chuankun Wu, Wenbao Han, Bogang Lin, Lequan Min, Qibin Zhai, Hong Zhu, Renji Tao, Zhiying Wang, Quanyuan Wu, and Zhichang Qi for their important counsel, suggestions, and corrections.

References

1. Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1990)
2. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
3. Menezes, A., Oorschot, P.V., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, London (1997)
4. Stallings, W.: Cryptography and Network Security: Principles and Practice, 2nd edn. Prentice-Hall, New Jersey (1999)
5. Su, S., Yang, Y., Yang, B.: etc: Design and Analysis of a Hash Ring-iterative Structure. Chinese Journal of Electronics **19**(2), 232–236 (2010)
6. Su, S., Lü, S.: A Public Key Cryptosystem Based on Three New Provable Problems. Theoretical Computer Science **426–427**, 91–117 (2012)
7. Yan, S.Y.: Number Theory for Computing, 2nd edn. Springer, New York (2002)

8. Hungerford, T.W.: Algebra. Springer, New York (1998)
9. Rosen, K.H.: Elementary Number Theory and Its Applications, 5th edn. Addison-Wesley, Boston (2005)
10. Wiener, M.J.: Cryptanalysis of Short RSA Secret Exponents. IEEE Transactions on Information Theory **36**(3), 553–558 (1990)
11. Chaum, D., van Heijst, E., Pfitzmann, B.: Cryptographically strong undeniable signatures, unconditionally secure for the signer. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 470–484. Springer, Heidelberg (1992)
12. Merkle, R.C., Hellman, M.E.: Hiding information and Signatures in Trapdoor Knapsacks. IEEE Transactions on Information Theory **24**(5), 525–530 (1978)
13. Shamir, A.: A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In: 23th IEEE Symposium on the Foundations of Computer Science, pp. 145–152. IEEE Press, New York (1982)
14. Du, D.Z., Ko, K., Hu, X.: Design and Analysis of Approximation Algorithms (in Chinese). Higher Education Press, Beijing (2011)
15. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press, Cambridge (2001)
16. Brickell, E.F.: Solving low density knapsacks. In: Advance in Cryptology: CRYPTO 1983, pp. 25–37. Plenum Press, New York (1984)
17. Li T., Su, S.: Analysis of success rate of attacking knapsacks from JUNA cryptosystem by LLL lattice basis reduction. In: 9th Int. Conf. on Comput. Intelligence and Security, pp. 454–458. IEEE Press, New York (2013)
18. Coster, M.J., Joux, A., LaMacchia, B.A., et al.: Improved Low-Density Subset Sum Algorithms. Computational Complexity **2**(2), 111–128 (1992)
19. Bellare, M., Kohno, T.: Hash function balance and its impact on birthday attacks. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 401–418. Springer, Heidelberg (2004)
20. Girault, M., Cohen, R., Campana, M.: A generalized birthday attack. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 129–156. Springer, Heidelberg (1988)
21. Diffie, W., Hellman, M.E.: Exhaustive Cryptanalysis of the NBS Data Encryption Standard. Computer **10**(6), 74–84 (1977)
22. Su, S., Lü, S.: REESSE1+. Reward. Proof by Experiment. A New Approach to Proof of P != NP. Cornell University Library (2009). http://arxiv.org/pdf/0908.0482 (revised 2014)

A Public Key Cryptoscheme Using Bit-Pairs with Provable Semantical Security

Shenghui Su[1,2(✉)], Shuwang Lü[3], and Maozhi Xu[4]

[1] Laboratory of Trusted Computing, Beijing University of Technology, Beijing 100124, People's Republic of China
reesse@126.com
[2] College of Information Engineering, Yangzhou University, Yangzhou 225009, People's Republic of China
[3] School of Computers, University of Chinese Academy of Sciences, Beijing 100039, People's Republic of China
[4] School of Mathematics, Peking University, Beijing 100871, People's Republic of China

Abstract. The authors give the definition and property of a bit-pair shadow, and design the algorithms of a public key cryptoscheme based on a multivariate permutation problem and an anomalous subset product problem to which no subexponential time solutions are found so far, and regards a bit-pair as an operation unit. Further, demonstrate that the decryption algorithm is correct, deduce the probability that a plaintext solution is nonunique is nearly zero, analyze the security of the new scheme against extracting a private key from a public key and recovering a plaintext from a ciphertext on the assumption that an integer factorization problem, a discrete logarithm problem, and a low-density subset sum problem can be solved efficiently, and prove that new scheme using random padding and permutation is semantically secure. The analysis shows that the bit-pair method increases the density D of a related knapsack to 1+, and decreases the modulus length $\lceil \lg M \rceil$ of the new scheme to 464, 544, or 640.

Keywords: Public key cryptoscheme · Semantical security · Bit-pair shadow · Random padding · Anomalous subset sum problem · Compact sequence

1 Introduction

In [1], we propose a prototypal public key cryptosystem called REESSE1+ which is based on the three new provable problems, contains the five algorithms, and is used for data encryption and digital signing. In REESSE1+, a ciphertext is defined as $\bar{G} \equiv \prod_{i=1}^{n} C_i^{\underset{\cdot}{b}_i}$ (% M), an anomalous subset product problem (ASPP), where $\underset{\cdot}{b}_i$ is the bit shadow of a bit b_i, $\{C_i\}$ is a public key, and n is the bit-length of a plaintext block [1].

S. Su—This work is supported by MOST with Project 2007CB311100 and 2009AA01Z441.

Let $C_1 \equiv g^{u_1}$ (% M), …, $C_n \equiv g^{u_n}$ (% M), and $\bar{G} \equiv g^{v}$ (% M), where g is a generator of $(\mathbb{Z}_M^*, \cdot)$ which can be found when the modulus $M < 2^{1024}$ is factorized in tolerable subexponential time [2]. Then solving $\bar{G} \equiv \prod_{i=1}^{n} C_i^{\underset{.}{b}_i}$ (% M) for $\underset{.}{b}_1 \ldots \underset{.}{b}_n$ is equivalent to solving

$$\underset{.}{b}_1 u_1 + \ldots + \underset{.}{b}_n u_n \equiv v \ (\%\ \bar{M}). \tag{1}$$

where v may be substituted with $v + k\bar{M}$ along with $k \in [0, n-1]$ [3].

Equation (1) is called an anomalous subset sum problem (ASSP) due to every $\underset{.}{b}_i \in [0, n]$ [1].

Likewise, due to every $\underset{.}{b}_i \in [0, n]$, $\{u_1, \ldots, u_n\}$ is called a compact sequence [4].

May convert an ASSP into a subset sum problem (SSP) through converting u_i to a binary number, and thus the density of an ASSP knapsack is defined as

$$D = \textstyle\sum_{i=1}^{n} \lceil \lg n \rceil / \lceil \lg M \rceil = n \lceil \lg n \rceil / \lceil \lg M \rceil. \tag{2}$$

Evidently, the parameters $\lceil \lg M \rceil$ and n have an important influence on the value of D.

In REESSE1+, there is $D < 1$, which means that the original solution to an ASSP may possibly be found through the LLL lattice basis reduction algorithm [5][6]. The LLL reduction algorithm is famous for it has a fatal threat to the classical MH knapsack cryptosystem [7] which produces a ciphertext in the form of a subset sum problem.

To avoid the low density of a knapsack from an ASPP and to decrease the modulus length of a system, on the basis of REESSE1+, we propose a new cryptoscheme called JUNA which treats a bit-pair as an operation unit, fetches randomicity in when a bit string is encrypted, and it is proved to be semantically secure.

Throughout this paper, unless otherwise specified, $n \geq 80$ is the bit-length of a plaintext block, $\tilde{n} \geq 144$ is the item-length of a public key sequence, the sign % denotes "modulo", $\bar{M}$ does "$M - 1$" with M prime, $\lg x$ means the logarithm of x to the base 2, $\neg$ does the opposite value of a bit, Ᵽ does the maximal prime allowed in coprime sequences, $|x|$ does the absolute value of a number x, $\|x\|$ does the order of an element x % M, $¦S¦$ does the size of a set S, gcd(a, b) represents the greatest common divisor of two integers, and "bos" indicates the number of bit operations. Without ambiguity, "% M" is usually omitted in expressions.

2 Several Definitions

2.1 A Coprime Sequence

Definition 1. If $A_1, \ldots, A_n$ are n pairwise distinct positive integers such that $\forall\, A_i, A_j$ $(i \neq j)$, either $\gcd(A_i, A_j) = 1$ or $\gcd(A_i, A_j) = F \neq 1$ with $(A_i / F) \nmid A_k$ and $(A_j / F) \nmid A_k$ $\forall\, k \neq i, j \in [1, n]$, these integers are called a coprime sequence, denoted by $\{A_1, \ldots, A_n\}$, shortly $\{A_i\}$.

Note that the elements of a coprime sequence are not necessarily pairwise coprime, but a sequence of which the elements are pairwise coprime is a coprime sequence.

Property 1. Let $\{A_1, \ldots, A_n\}$ be a coprime sequence. If randomly select $k \in [1, n]$ elements $A_{x_1}, \ldots, A_{x_k}$ from the sequence, then the mapping from a subset $\{A_{x_1}, \ldots, A_{x_k}\}$

to a subset product $G = \prod_{i=1}^{k} A_{x_i}$ is one-to-one, namely the mapping from $b_1 \ldots b_n$ to $G = \prod_{i=1}^{n} A_i^{b_i}$ is one-to-one, where $b_1 \ldots b_n$ is a bit string.

Refer to [1] for its proof.

2.2 A Bit Shadow

Definition 2. Let $b_1 \ldots b_n \neq 0$ be a bit string. Then $\underset{.}{b}_i$ with $i \in [1, n]$ is called a bit shadow if it comes from such a rule: ① $\underset{.}{b}_i = 0$ if $b_i = 0$, ② $\underset{.}{b}_i = 1 +$ the number of successive 0-bits before b_i if $b_i = 1$, or ③ $\underset{.}{b}_i = 1 +$ the number of successive 0-bits before b_i + the number of successive 0-bits after the rightmost 1-bit if b_i is the leftmost 1-bit.

Fact 1: Let $\underset{.}{b}_1 \ldots \underset{.}{b}_n$ be the bit shadow string of $b_1 \ldots b_n \neq 0$. Then there is $\sum_{i=1}^{n} \underset{.}{b}_i = n$.

Property 2. Let $\{A_1, \ldots, A_n\}$ be a coprime sequence, and $\underset{.}{b}_1 \ldots \underset{.}{b}_n$ be the bit shadow string of $b_1 \ldots b_n \neq 0$. Then the mapping from $b_1 \ldots b_n$ to $G = \prod_{i=1}^{n} A_i^{\underset{.}{b}_i}$ is one-to-one [8].

The proofs of Fact1 and Property 2 are omitted due to limited pages.

2.3 A Bit-Pair Shadow

To make the modulus M of the new cryptoscheme comparatively small, we will utilize the idea of a bit-pair string with 2 bits to 3 items.

In this wise, the length of a coprime sequence is changed to $3n/2$, namely $\{A_1, \ldots, A_n\}$ is substituted with $\{A_1, A_2, A_3, \ldots, A_{3n/2-2}, A_{3n/2-1}, A_{3n/2}\}$ that may be logically orderly partitioned into $n/2$ triples of which each comprises 3 elements: A_{3j-2}, A_{3j-1}, A_{3j} with $j \in [1, n/2]$. Likewise, a non-coprime sequence $\{C_1, \ldots, C_n\}$ is substituted with $\{C_1, C_2, C_3, \ldots, C_{3n/2-2}, C_{3n/2-1}, C_{3n/2}\}$, where $(C_{3j-2}, C_{3j-1}, C_{3j})$ with $j \in [1, n/2]$ is acquired from $(A_{3j-2}, A_{3j-1}, A_{3j})$ and other private parameters.

Definition 3. Let $\{A_{3j-2}, A_{3j-1}, A_{3j} \mid j = 1, \ldots, n/2\}$ be a coprime sequence. Orderly partition a bit string $b_1 \ldots b_n$ into $n/2$ pairs $B_1, \ldots, B_{n/2}$, where B_j with $j \in [1, n/2]$ has four states: 00, 01, 10, and 11 which correspond to 1, A_{3j-2}, A_{3j-1}, and A_{3j} respectively. Then $B_1, \ldots, B_{n/2}$ is called a bit-pair string, shortly $B_1 \ldots B_{n/2}$.

Property 3. Let $\{A_{3j-2}, A_{3j-1}, A_{3j} \mid j=1, \ldots, n/2\}$ be a coprime sequence, and $B_1 \ldots B_{n/2}$ be a nonzero bit-pair string. Then the mapping from $B_1 \ldots B_{n/2}$ to $G' = \prod_{i=1}^{n/2} (A_{3(i-1)+B_i})^{\lceil B_i/3 \rceil}$ with $A_0 = 1$ is one-to-one, where $\lceil B_i/3 \rceil = 0$ or 1, and G' is called a coprime subsequence product.

Its proof is parallel to that of Property 1 in [1].

Definition 4. Let $B_1 \ldots B_{n/2}$ be a nonzero bit-pair string. Then $\underset{.}{B}_i$ with $i \in [1, n/2]$ is called a bit-pair shadow if it comes from such a rule: ① $\underset{.}{B}_i = 0$ if $B_i = 00$, ② $\underset{.}{B}_i = 1 +$ the number of successive 00-pairs before B_i if $B_i \neq 00$, or ③ $\underset{.}{B}_i = 1 +$ the number of successive 00-pairs before B_i + the number of successive 00-pairs after the rightmost non-00-pair if B_i is the leftmost non-00-pair.

Fact 2. Let $\underset{.}{B}_1 \ldots \underset{.}{B}_{n/2}$ be the bit-pair shadow string of $B_1 \ldots B_{n/2} \neq 0$. Then there is $\sum_{i=1}^{n/2} \underset{.}{B}_i = n/2$.

Its proof is omitted due to limited pages.

Property 4. Let $\{A_{3j-2}, A_{3j-1}, A_{3j} \mid j = 1, \ldots, n/2\}$ be a coprime sequence, and $\underset{\cdot}{B}_1\ldots\underset{\cdot}{B}_{n/2}$ be the bit-pair shadow string of $B_1\ldots B_{n/2} \neq 0$. Then the mapping from $B_1\ldots B_{n/2}$ to $G = \prod_{i=1}^{n/2} (A_{3(i-1)+B_i})^{\underset{\cdot}{B}_i}$ with $A_0 = 1$ is one-to-one, where G is called an anomalous coprime subsequence product.

Its proof is parallel to that of Property 2 in Section 2.2.

2.4 A Lever Function

Definition 5. The secret parameter $\ell(i)$ in the key transform of a public key cryptoscheme is called a lever function, if it has the following features:

- $\ell(.)$ is an injection from the domain $\{1,\ldots,n\}$ to the codomain $\Omega \subset \{5,\ldots,\bar{M}\}$ with $\bar{M}$ large;
- the mapping between i and $\ell(i)$ is established randomly without an analytical expression;
- an attacker has to be faced with all the arrangements of n elements in Ω when extracting a related private key from a public key;
- the owner of a private key only needs to consider the accumulative sum of n elements in Ω when recovering a related plaintext from a ciphertext.

The latter two points manifest that if n is large enough, it is infeasible for the attacker to search all the permutations of elements in Ω exhaustively while the decryption of a normal ciphertext is feasible in polynomial time in n. Thus, there are the large amount of calculation on $\ell(.)$ at "a public terminal", and the small amount of calculation on $\ell(.)$ at "a private terminal".

Notice that ① in arithmetic modulo $\bar{M}$, $-x$ represents $\bar{M} - x$; ② considering the speed of decryption, the absolute values of all the elements should be comparatively small; ③ the lower limit 5 will make seeking the root W from $W^{\ell(i)} \equiv A_i^{-1} C_i$ ($\%$ M) face an unsolvable Galois group when the value of $A_i \leq 1201$ is guessed [9].

Property 5 **(Indeterminacy of $\ell(.)$).** Let $\delta = 1$ and $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ ($\%$ M) with $\ell(i) \in \Omega = \{5, 6, \ldots, n + 4\}$ and $A_i \in \Lambda = \{2, 3, \ldots, \mathcal{P} \mid \mathcal{P} \leq 1201\}$ for $i = 1, \ldots, n$. Then $\forall W$ ($\|W\| \neq \bar{M}$) $\in (1, \bar{M})$ and $\forall\, x, y, z\ (x \neq y \neq z) \in [1, n]$,

① when $\ell(x) + \ell(y) = \ell(z)$, there is $\ell(x) + \|W\| + \ell(y) + \|W\| \neq \ell(z) + \|W\|$ ($\%$ $\bar{M}$);

② when $\ell(x) + \ell(y) \neq \ell(z)$, there always exist $C_x \equiv A'_x W'^{\ell'(x)}$ ($\%$ M), $C_y \equiv A'_y W'^{\ell'(y)}$ ($\%$ M), and $C_z \equiv A'_z W'^{\ell'(z)}$ ($\%$ M) such that $\ell'(x) + \ell'(y) \equiv \ell'(z)$ ($\%$ $\bar{M}$) with $A'_z \leq \mathcal{P}$.

Refer to [1] for its proof.

Notice that according to the proof of Property 5 in [1], it is not difficult to understand that when $\Omega = \{5, 6, \ldots, n + 4\}$ is substituted with $\Omega = \{+/-5, +/-7, \ldots, +/-(2n + 3)\}$, where "+/–" means the selection of the "+" or "–" sign, Property 5 still holds.

3 Design of the New Cryptoscheme

Due to $L_p[1/3, 1.923] = 2^{80}$ with p prime and $\lceil \lg p \rceil \approx 1024$ [10], the shortest bit-length of a plaintext block should be 80. In the new scheme, to acquire provable semantical security, random 16 bits are appended the terminal of a plaintext block of n bits when it is encrypted.

Let $\tilde{n} = n + 16$ with n = 80, 96, or 112. Additionally, two adjacent bits are orderly treated as a unit, namely a bit-pair string $B_1 \ldots B_{\tilde{n}/2}$ is used to represent a plaintext block $b_1 \ldots b_{\tilde{n}} \neq 0$.

3.1 Key Generation Algorithm

Considering decryption speed, the absolute values of elements of Ω should be as small as possible, and every three successive elements of Ω are treated as a triple according to 2 bits to 3 items.

Let $\Lambda = \{2, \ldots, Ꝑ\}$, where $Ꝑ$ = 937, 991, or 1201 corresponding to $\tilde{n}$ = 96, 112, or 128 separately.

Let $\tilde{\imath} = \lceil \lg M \rceil$ = 464, 544, or 640 corresponding to $\tilde{n}$ = 96, 112, or 128 separately.

Assume that $\bar{A}_j$ is the maximum in $(A_{3j-2}, A_{3j-1}, A_{3j})$ $\forall\, j \in [1, \tilde{n}/2]$.

The following algorithm is generally employed by the owner of a key pair.

```
INPUT: the integer $n$; the integer $\tilde{\imath}$; the prime $Ꝑ$.
S1: Let $\tilde{n} \leftarrow n + 16$, $\Lambda \leftarrow \{2, \ldots, Ꝑ\}$;
    yield the first $\tilde{n}$ primes $\acute{p}_1, \ldots, \acute{p}_{\tilde{n}}$ in the natural number set;
    yield $\Omega \leftarrow \{(+/-(6j-1), +/-(6j+1), +/-(6j+3))_{Æ} \mid j=1, \ldots, \tilde{n}/2\}$.
S2: Produce an odd coprime $\{A_1, \ldots, A_{3\tilde{n}/2} \mid A_i \in \Lambda\} = \{A_{3j-2}, A_{3j-1}, A_{3j} \mid j=1, \ldots, \tilde{n}/2\}$;
    arrange $\bar{A}_1, \ldots, \bar{A}_{\tilde{n}/2}$ to $\bar{A}x_1, \ldots, \bar{A}x_{\tilde{n}/2}$ in descending order.
S3: Find a prime $M > \bar{A}x_1^{\tilde{n}/4+1} \prod_{i=2}^{\tilde{n}/4} \bar{A}x_i$ making $\lceil \lg M \rceil = \tilde{\imath}$ and
    $\prod_{i=1}^{k} \acute{p}_i^{e_i} \mid \bar{M}$, where $k$ meets $\prod_{i=1}^{k} e_i \geq 2^{10}$ and $\acute{p}_k < \tilde{n}$.
S4: Produce pairwise distinct $(\ell(3j-2), \ell(3j-1), \ell(3j)) \in \Omega$ for $j = 1, \ldots, \tilde{n}/2$.
S5: Stochastically pick $W, \delta \in (1, \bar{M})$ making $\|W\| \geq 2^{n-20}$ and $\gcd(\delta, \bar{M}) = 1$.
S6: Compute $C_i \leftarrow (A_i W^{\ell(i)})^{\delta}$ % $M$ for $i = 1, \ldots, 3\tilde{n}/2$.
OUTPUT: a public key $(\{C_1, \ldots, C_{3\tilde{n}/2}\}, M)$; a private key $(\{A_1, \ldots, A_{3\tilde{n}/2}\}, W, \delta, M)$.
```

The lever function $\{\ell(1), \ldots, \ell(\tilde{n}/2)\}$ is discarded but must not be divulged. Notice that

① at S1, $\Omega = \{(+/-(6j-1), +/-(6j+1), +/-(6j+3))_{Æ} \mid j = 1, \ldots, \tilde{n}/2\}$ indicates that Ω is one of $(3!)^{\tilde{n}/2} 2^{3\tilde{n}/2}$ potential sets consisting of 3-tuple elements, where "+/–" means the selection of the "+" or "–" sign, and the subscript Æ means that $(+/-(6j-1), +/-(6j+1), +/-(6j+3))_{Æ}$ is a permutation of $(+/-(6j-1), +/-(6j+1), +/-(6j+3))$;

② at S2, $\gcd(A_{3i-2}, A_{3i-1}, A_{3i}) \neq 1$ $(i \in [1, \tilde{n}/2])$ is allowed — $(3^3, 3^2, 3)$ for example since only one of three elements will occur in the product G;

③ at S3, the inequation $M > \bar{A}x_1^{\tilde{n}/4+1} \prod_{i=2}^{\tilde{n}/4} \bar{A}x_i$ assures that a ciphertext can be decrypted correctly;

④ at S5, let $W \equiv g^{\bar{M}/F}$ (% M), then $\|W\| = \bar{M} / \gcd(\bar{M}, \bar{M}/F)$ [9], where $F \geq 2^{n-20}$ is a factor of $\bar{M}$, and g is a generator by Algorithm 4.80 in Section 4.6 of [10].

Definition 6. Given $(\{C_1, \ldots, C_{3\tilde{n}/2}\}, M)$, seeking the original $(\{A_1, \ldots, A_{3\tilde{n}/2}\}, \{\ell(1), \ldots, \ell(3\tilde{n}/2)\}, W, \delta)$ from $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $A_i \in \Lambda = \{2, \ldots, Ꝑ \mid Ꝑ \leq 1201\}$ and $\ell(i)$ from $\Omega = \{(+/-(6j-1), +/-(6j+1), +/-(6j+3))_P \mid j = 1, \ldots, \tilde{n}/2\}$ for $i = 1, \ldots, 3\tilde{n}/2$ is referred to as a multivariate permutation problem (MPP).

Property 6: The MPP $C_i \equiv (A_i W^{\ell(i)})^{\delta}$ (% M) with $A_i \in \Lambda = \{2, \ldots, Ꝑ \mid Ꝑ \leq 1201\}$ and $\ell(i)$ from $\Omega = \{(+/-(6j-1), +/-(6j+1), +/-(6j+3))_P \mid j=1, \ldots, \tilde{n}/2\}$ for $i = 1, \ldots, 3\tilde{n}/2$ is computationally at least equivalent to the DLP in the same prime field.

Refer to Section 4.1 of [1] for its proof.

3.2 Encryption Algorithm

This algorithm is employed by a person who wants to encrypt plaintexts.

INPUT: a public key $(\{C_1, \ldots, C_{3\tilde{n}/2}\}, M)$;

the bit-pair string $B_1 \ldots B_{n/2}$ of a plaintext block $b_1 \ldots b_n \neq 0$.

Notice that if the number of 00-pairs in $B_1 \ldots B_{n/2}$ is larger than $n/4$, let $b_1 \ldots b_n = \neg b_1 \ldots \neg b_n$ in order that a related ciphertext can be decrypted conforming to the constraint on M.

```
S1: Yield a random bit string $b_{n+1} \ldots b_{\tilde{n}}$ appended to $b_1 \ldots b_n$;
    form $B_1 \ldots B_{\tilde{n}/2}$ until the number of 00-pairs $\leq \tilde{n}/4$.
S2: Set $C_0 \leftarrow 1$, $k \leftarrow 0$, $i \leftarrow 1$, $\bar{s} \leftarrow 0$.
S3: If $B_i = 00$ then let $k \leftarrow k + 1$, $\underset{\cdot}{B}_i \leftarrow 0$
    else {let $\underset{\cdot}{B}_i \leftarrow k + 1$, $k \leftarrow 0$; if $\bar{s} = 0$ then $\bar{s} \leftarrow i$ else null.}
S4: Let $i \leftarrow i + 1$; if $i \leq \tilde{n}/2$ then goto S3.
S5: If $k \neq 0$ then let $\underset{\cdot}{B}_{\bar{s}} \leftarrow \underset{\cdot}{B}_{\bar{s}} + k$.
S6: Stochastically produce $r_1 \ldots r_{\tilde{n}/2} \in \{0, 1\}^{\tilde{n}/2}$; set $r_{\bar{s}} \leftarrow 1$.
S7: Compute $\bar{G} \leftarrow \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i) + \neg r_i(3(i-\underset{\cdot}{B}_i)+B_i)})^{\underset{\cdot}{B}_i} \,\%\, M$.
```

OUTPUT: a ciphertext $\bar{G}$.

Obviously, a different ciphertext will be outputted every time an identical plaintext is inputted repeatedly. The identical plaintext may correspond to at most $2^{\tilde{n}/4} 2^{\tilde{n}-n}$ different ciphertexts because $\tilde{n}/2$ bit-pairs may be interlaced by a 00-pair and a non-00-pair, and $b_{n+1} \ldots b_{\tilde{n}}$ is produced randomly. It will take the running time of $O(\tilde{n} 2^{\tilde{n}/2} 2^{\tilde{n}-n} \lg^2 M)$ bit operations exhaustively to search all the possible ciphertexts of a plaintext.

Note a JUNA ciphertext $\prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i) + \neg r_i(3(i-\underset{\cdot}{B}_i)+B_i)})^{\underset{\cdot}{B}_i} (\% M)$ is different from a Naccache-Stern ciphertext $c \equiv \prod_{i=1}^{n} v_i^{b_i} (\% M)$ [11], where $v_i \equiv p_i^{1/\delta}$ $(\% M)$ with p_i prime is a public key.

Definition 7. Given $(\{C_1, \ldots, C_{3\tilde{n}/2}\}, M)$ and $\bar{G}$, seeking $B_1 \ldots B_{\tilde{n}/2}$ from $\bar{G}' \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+B_i})^{\lceil B_i/3 \rceil}$ $(\% M)$ with $C_0 = 1$ is referred to as a subset product problem (SPP), where $B_1 \ldots B_{\tilde{n}/2}$ is the bit-pair string of $b_1 \ldots b_{\tilde{n}} \neq 0$.

Property 7. The SPP $\bar{G}' \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+B_i})^{\lceil B_i/3 \rceil}$ $(\% M)$ with $C_0 = 1$ is computationally at least equivalent to the DLP in the same prime field, where $B_1 \ldots B_{\tilde{n}/2} \neq 0$ is a bit-pair string.

Definition 8: Given $(\{C_1, \ldots, C_{3\tilde{n}/2}\}, M)$ and $\bar{G}$, seeking $B_1 \ldots B_{\tilde{n}/2}$ from $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+B_i})^{\underset{\cdot}{B}_i}$ $(\% M)$ or $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i) + \neg r_i(3(i-\underset{\cdot}{B}_i)+B_i)})^{\underset{\cdot}{B}_i}$ $(\% M)$ with $C_0 = 1$ and $r_1 \ldots r_{\tilde{n}/2}$ a random bit string is referred to as an anomalous subset product problem (ASPP), where $\underset{\cdot}{B}_1 \ldots \underset{\cdot}{B}_{\tilde{n}/2}$ is the bit-pair shadow string of $B_1 \ldots B_{\tilde{n}/2}$ corresponding to $b_1 \ldots b_{\tilde{n}} \neq 0$.

Property 8: The ASPP $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+B_i})^{\underset{\cdot}{B}_i}$ $(\% M)$ with $C_0 = 1$ is computationally at least equivalent to the DLP in the same prime field, where $\underset{\cdot}{B}_1 \ldots \underset{\cdot}{B}_{\tilde{n}/2}$ is the bit-pair shadow string of $B_1 \ldots B_{\tilde{n}/2} \neq 0$.

Property 9: The ASPP $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i) + \neg r_i(3(i-\underset{\cdot}{B}_i)+B_i)})^{\underset{\cdot}{B}_i}$ $(\% M)$ with $C_0 = 1$ and $r_1 \ldots r_{\tilde{n}/2}$ a random bit string is computationally at least equivalent to the DLP in the same prime field, where $\underset{\cdot}{B}_1 \ldots \underset{\cdot}{B}_{\tilde{n}/2}$ is the bit-pair shadow string of $B_1 \ldots B_{\tilde{n}/2} \neq 0$.

The proofs of Property 7, 8, 9 are omitted due to limited pages.

3.3 Decryption Algorithm

This algorithm is employed by a person who wants to decrypt ciphertexts.

INPUT: a private key ($\{A_1, \ldots, A_{3\tilde{n}/2}\}$, W, δ, M); a ciphertext $\bar{G}$.

It should be noted that due to $2 \mid \sum_{i=1}^{\tilde{n}/2} \underset{\cdot}{B}_i$ and $2 \nmid \ell(r_i(3(i-1)+B_i)+\neg r_i(3(i-\underset{\cdot}{B}_i)+B_i))$ for $i \in [1, \tilde{n}/2]$ with $\ell(0) = 0$, $\underline{k} = \sum_{i=1}^{\tilde{n}/2} \underset{\cdot}{B}_i \ell(r_i(3(i-1)+B_i)+\neg r_i(3(i-\underset{\cdot}{B}_i)+B_i))$ must be even.

```
S1: Compute $Z_0 \leftarrow \bar{G}^{\delta^{-1}}$ % $M$; set $Z_1 \leftarrow Z_0$, $\underset{\cdot}{s} \leftarrow 0$.
S2: If $2 \mid Z_{\underset{\cdot}{s}}$ then {do $Z_{\underset{\cdot}{s}} \leftarrow Z_{\underset{\cdot}{s}} W^{2(-1)^{\underset{\cdot}{s}}}$ % $M$; goto S2.}
S3: Set $B_1 \ldots B_{n/2} \leftarrow 0$, $j \leftarrow 0$, $k \leftarrow 0$, $\tilde{v} \leftarrow 0$, $i \leftarrow 1$, $G \leftarrow Z_{\underset{\cdot}{s}}$.
S4: If $(A_{3i-j})^{\tilde{v}+1} \mid G$ then {let $\tilde{v} \leftarrow \tilde{v} + 1$; goto S4.}
S5: Let $j \leftarrow j + 1$; if $\tilde{v} = 0$ and $j \le 2$ then goto S4.
S6: If $\tilde{v} = 0$ then
        S6.1: let $k \leftarrow k + 1$, $i \leftarrow i + 1$
    else
        S6.2: compute $G \leftarrow G / (A_{3i-j})^{\tilde{v}}$;
        S6.3: if $k > 0$ or $\tilde{v} \ge i$ then let $B_i \leftarrow 3 - j$, $i \leftarrow i + 1$ else let $B_{i+\tilde{v}-1} \leftarrow 3 - j$, $i \leftarrow i + \tilde{v}$;
        S6.4: set $\tilde{v} \leftarrow 0$, $k \leftarrow 0$.
S7: If $i \le \tilde{n}/2$ and $G \ne 1$ then set $j \leftarrow 0$, goto S4.
S8: If $G \ne 1$ then {set $\underset{\cdot}{s} \leftarrow \neg \underset{\cdot}{s}$; do $Z_{\underset{\cdot}{s}} \leftarrow Z_{\underset{\cdot}{s}} W^{2(-1)^{\underset{\cdot}{s}}}$ % $M$; goto S2.}
S9: Extract $B_1 \ldots B_{n/2}$ from $B_1 \ldots B_{\tilde{n}/2}$.
```

OUTPUT: a related plaintext $B_1 \ldots B_{n/2}$, namely $b_1 \ldots b_n$.

Notice that only if $\bar{G}$ is a true ciphertext, can the algorithm always terminate normally, and $b_1 \ldots b_n$ will be original although $r_1 \ldots r_{\tilde{n}/2}$ is brought into an encryption process.

4 Correctness and Uniqueness

4.1 Correctness of the Decryption Algorithm

Because ($\mathbb{Z}_M^*$, $\cdot$) is an Abelian group, namely a commutative group, $\forall \underline{k} \in [1, \bar{M}]$, there is $W^{\underline{k}} (W^{-1})^{\underline{k}} \equiv W^{\underline{k}} (W^{\underline{k}})^{-1} \equiv 1$ (% M), where $W \in [1, \bar{M}]$ is any arbitrary integer.

Fact 3: Let $\underline{k} = \sum_{i=1}^{\tilde{n}/2} \underset{\cdot}{B}_i \ell(r_i(3(i-1)+B_i)+\neg r_i(3(i-\underset{\cdot}{B}_i)+B_i))$ % $\bar{M}$ with $\ell(0) = 0$, where $\underset{\cdot}{B}_1 \ldots \underset{\cdot}{B}_{\tilde{n}/2}$ is the bit-pair shadow string of $B_1 \ldots B_{\tilde{n}/2}$ corresponding to $b_1 \ldots b_{\tilde{n}} \ne 0$, and $r_1 \ldots r_{\tilde{n}/2}$ is a random bit string. Then $\bar{G}^{\delta^{-1}} (W^{-1})^{\underline{k}} \equiv \prod_{i=1}^{\tilde{n}/2} (A_{r_i(3(i-1)+B_i) + \neg r_i(3(i - \underset{\cdot}{B}_i) + B_i)})^{\underset{\cdot}{B}_i}$ (% M).

Its proof is omitted due to limited pages.

Notice that in practice, $\underline{k}$ is unknowable in advance.

However, because $|\underline{k}| < \tilde{n}(2(3\tilde{n}/2)+3)/2 = 3\tilde{n}(\tilde{n} + 1)/2$ is comparatively small, we may search $\underline{k}$ heuristically by multiplying W^{-2} or W^2 % M and judging whether $G = 1$ after it is divided exactly by some $(A_{3i-j})^{\tilde{v}}$. It is known from the decryption algorithm that the original $B_1 \ldots B_{\tilde{n}/2}$ will be acquired at the same time the condition $G = 1$ is satisfied.

4.2 Uniqueness of a Plaintext Solution

Because the public key $\{C_1, \ldots, C_{3\tilde{n}/2}\}$ is a non-coprime sequence, the mapping from $B_1 \ldots B_{\tilde{n}/2}$ to $\bar{G} = \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i)+\neg r_i(3(i-\underset{\cdot}{B}_i)+B_i)})^{\underset{\cdot}{B}_i}$ % M is theoretically many-to-one.

It might possibly result in the nonuniqueness of a plaintext solution $B_1 \ldots B_{\bar{n}/2}$ when $\bar{G}$ is being unveiled.

Fact 4. The probability that a plaintext solution for $\bar{G} = \prod_{i=1}^{\bar{n}/2} (C_{r_i(3(i-1)+B_i)+\neg r_i(3(i-B_i)+B_i)})^{B_i}$ % M is nonunique is nearly zero.

Its proof is omitted due to limited pages.

5 Security Analysis of a Private Key

In the below cryptanalysis, we suppose that the integer factorization problem (IFP) $N = pq$ with $\lceil \lg N \rceil < 1024$ [2], the discrete logarithm problem (DLP) $y \equiv g^x$ (% p) with $\lceil \lg p \rceil < 1024$ [12][13], and the subset sum problem (SSP) of low density $s \equiv \sum_{i=1}^{n} c_i b_i$ (% M) with $D \approx n / \lceil \lg M \rceil < 1$ and $n < \lceil \lg M \rceil < 1024$ [7] can be solved in tolerable subexponential time or in polynomial time [14].

Notice that the structure of the set Ω consisting of triples has no change in essence compared with the Ω in [1].

Hence, the security analysis of a private key is similar to Section 4 of [1].

6 Security Analysis of a Plaintext

The security of a plaintext depends on the ASPP $\bar{G} \equiv \prod_{i=1}^{\bar{n}/2} (C_{r_i(3(i-1)+B_i)+\neg r_i(3(i-B_i)+B_i)})^{B_i}$ (% M) with $C_0 = 1$ and $r_1 \ldots r_{\bar{n}/2}$ a random bit string.

Definition 9. Let A and B be two computational problems. A is said to reduce to B in polynomial time, written as $A \le_T^P B$, if there is an algorithm for solving A which calls, as a subroutine, a hypothetical algorithm for solving B, and runs in polynomial time, excluding the time of the algorithm for B [10][15].

Definition 10. Let A and B be two computational problems. If $A \le_T^P B$ and $B \le_T^P A$, then A and B are said to be computationally equivalent, written as $A =_T^P B$ [10][15].

Definition 9 and 10 suggest a reductive proof method called polynomial time Turing reduction (PTR) [15].

Naturally, we will enquire whether $A <_T^P B$ exists or not. The definition of $A <_T^P B$ may possibly be given theoretically, but the proof of $A <_T^P B$ is not easy in practice.

Let $\hat{H}(y = f(x))$ represent the complexity or hardness of solving the problem $y = f(x)$ for x [14].

6.1 Resisting LLL Lattice Basis Reduction

We know that after a lattice basis is reduced through the LLL algorithm, the final reduced base will contain the shortest or approximately shortest vectors, but among them does not necessarily exist the original solution to a subset sum problem because only if

① the solution vector for the SSP is the shortest,

② the shortest vector is unique in the lattice,

will the original solution vector appear in the reduced base with large probability.

In the new cryptoscheme, there are $\tilde{n} = 96$, 112, or 128 and $\lceil \lg M \rceil = 464$, 544, or 640. Under the circumstances, the DLP and IFP can be solved in tolerable subexponential time, namely the DLP and IFP cannot resist the attack of adversaries.

We first consider the ASPP $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+\underset{.}{B}_i})^{\underset{.}{B}_i}$ (% M).

For convenience, extend $\underset{.}{B}_1 \ldots \underset{.}{B}_{\tilde{n}/2}$ to $\underset{.}{b}'_1 \ldots \underset{.}{b}'_{3\tilde{n}/2}$ by the following rule for $i = 1, \ldots, \tilde{n}/2$:

① when $\underset{.}{B}_i = 0$, let $\underset{.}{b}'_{3(i-1)+1} = \underset{.}{b}'_{3(i-1)+2} = \underset{.}{b}'_{3(i-1)+3} = 0$;

② when $\underset{.}{B}_i \neq 0$, let $\underset{.}{b}'_{3(i-1)+1} = \underset{.}{b}'_{3(i-1)+2} = \underset{.}{b}'_{3(i-1)+3} = 0$, and $\underset{.}{b}'_{3(i-1)+\underset{.}{B}_i} = \underset{.}{B}_i$.

In this way, the ASPP $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+\underset{.}{B}_i})^{\underset{.}{B}_i}$ (% M) is converted into $\bar{G} \equiv \prod_{i=1}^{3\tilde{n}/2} C_i^{\underset{.}{b}'_i}$ (% M).

Let g be a generator of the group $(\mathbb{Z}_M^*, \cdot)$.

Let $C_1 \equiv g^{u_1}$ (% M), …, $C_{3\tilde{n}/2} \equiv g^{u_{3\tilde{n}/2}}$ (% M), and $\bar{G} \equiv g^v$ (% M).

Then, through a conversion in subexponential time, seeking $B_1 \ldots B_{\tilde{n}/2}$ from $\bar{G}$ is equivalent to seeking $\underset{.}{b}'_1 \ldots \underset{.}{b}'_{3\tilde{n}/2}$ from the congruence

$$u_1 \underset{.}{b}'_1 + \ldots + u_{3\tilde{n}/2} \underset{.}{b}'_{3\tilde{n}/2} \equiv v \ (\%\ \bar{M}), \tag{3}$$

where v may be substituted with $v + k\bar{M}$ along with $k \in [0, 3\tilde{n}/2]$ [3].

Similar to Section 1, $\{u_1, \ldots, u_{3\tilde{n}/2}\}$ is called a compact sequence due to every $\underset{.}{b}'_i \in [0, \tilde{n}/4 + 1]$ [4], and solving Equation (3) for $\underset{.}{b}'_1 \ldots \underset{.}{b}'_{3\tilde{n}/2}$ is called an ASSP [1].

May also convert this ASSP into a SSP through splitting u_i into bits, and thus according to $\underset{.}{b}'_i \in [0, \tilde{n}/4 + 1]$, the density of the related ASSP knapsack is defined as $D = \sum_{i=1}^{3\tilde{n}/2} \lceil \lg(\tilde{n}/4 + 1) \rceil / \lceil \lg M \rceil = (3\tilde{n}/2) \lceil \lg(\tilde{n}/4 + 1) \rceil / \lceil \lg M \rceil$. Namely,

$$D = 3\tilde{n} \lceil \lg(\tilde{n}/4 + 1) \rceil / (2 \lceil \lg M \rceil). \tag{4}$$

which is slightly different from Formula (2).

Concretely speaking, in the new cryptoscheme, there are

$D = 144 \times 5 / 464 \approx 1.5517 > 1$ for $\tilde{n} = 96$ and $\lceil \lg M \rceil = 464$;

$D = 168 \times 5 / 544 \approx 1.5441 > 1$ for $\tilde{n} = 112$ and $\lceil \lg M \rceil = 544$;

$D = 192 \times 6 / 640 \approx 1.8000 > 1$ for $\tilde{n} = 128$ and $\lceil \lg M \rceil = 640$.

Therefore, Equation (3) does represent an ASSP of high density, which indicates that many different subsets will have the same sum, and probability that the original solution vector will occur in the final reduced lattice basis is nearly zeroth. Meanwhile, our experiment demonstrates that the original solution vector does not occur in the final reduced base [16].

Because $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{3(i-1)+\underset{.}{B}_i})^{\underset{.}{B}_i}$ (% M) is only a special case of $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+\underset{.}{B}_i) + \neg r_i(3(i-\underset{.}{B}_i)+\underset{.}{B}_i)})^{\underset{.}{B}_i}$ (% M), the latter is also able to resist the LLL lattice basis reduction.

6.2 Avoiding Meet-in-the-middle Attack

Meet-in-the-middle dichotomy was first developed in 1977 [17]. Section 3.10 of [10] puts forward a meet-in-the-middle attack on a subset sum problem. It is not difficult to understand that the time complexity of the above algorithm is $O(n2^{n/2})$.

Likewise, currently the versatile meet-in-the-middle dichotomy may be used to assault the ASSP $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i)} + \neg r_i(3(i-\bar{B}_i)+B_i))^{\bar{B}_i}$ (% M) with entries

$$(\prod_{i=1}^{\tilde{n}/4} (C_{r_i(3(i-1)+B_i)} + \neg r_i(3(i-\bar{B}_i)+B_i))^{\bar{B}_i}, (r_1, \ldots, r_{\tilde{n}/4}), (B_1, \ldots, B_{\tilde{n}/4}))$$

for $(r_1, \ldots, r_{\tilde{n}/4}) \in \{0, 1\}^{\tilde{n}/4}$ and $(B_1, \ldots, B_{\tilde{n}/4}) \in \{00, 01, 10, 11\}^{\tilde{n}/4}$ when $B_{\tilde{n}/4} \neq 00$ and $B_{\tilde{n}/2} \neq 00$ which occurs with probability 9 / 16 = 0.5625. Obviously, the random bit string $r_1 \ldots r_{\tilde{n}/4}$ extends the scope of exhaustive search. Further, It is easy to see that the running time of this attack task is $O(\tilde{n}2^{\tilde{n}/2}2^{\tilde{n}/4}\lg^2 M) = O(\tilde{n}2^{3\tilde{n}/4}\lg^2 M)$ bit operations.

Concretely speaking,
when $\tilde{n} = 96$ namely $n = 80$ with $\lceil \lg M \rceil = 464$, $\mathcal{T}_m = 2^7 2^{3\times 96/4}(2^9)^2 = 2^{97} > 2^{80}$ bos;
when $\tilde{n} = 112$ namely $n = 96$ with $\lceil \lg M \rceil = 544$, $\mathcal{T}_m = 2^7 2^{3\times 112/4}(2^{10})^2 = 2^{111} > 2^{96}$ bos;
when $\tilde{n} = 128$ namely $n = 112$ with $\lceil \lg M \rceil = 640$, $\mathcal{T}_m = 2^8 2^{3\times 128/4}(2^{10})^2 = 2^{124} > 2^{112}$ bos.
Therefore, the new cryptoscheme can resist the meet-in-the-middle attack.

6.3 Avoiding Adaptive-chosen-Ciphertext Attack

Most of public key cryptoschemes may probably be faced with adaptive-chosen-ciphertext attack [18]. However, It is lucky the Cramer-Shoup asymmetric encryption scheme is very indistinguishable and nonmalleable [19], and proven to be secure against the adaptive-chosen-ciphertext attack under the cryptographic assumptions [20]. So is the OAEP+ scheme [21].

6.3.1 Indistinguishability of Ciphertexts

In the encryption process of a JUNA plaintext, ①a random padding string of $\tilde{n} - n$ bits is appended to the terminal of the JUNA plaintext, which changes the original plaintext to an extended plaintext, and ② a random permutation string of $\tilde{n}/2$ bits is introduced into the arrangement of bit-pairs of the extended plaintext, which is equivalent to the thing that the order of triple items of a public key is varied along with every encryption.

Due to the interlacement of 00-pairs and non-00-pairs and the randomicity of bit string generation, the padding string and the permutation string make one identical original plaintext be able to correspond to at most $2^{\tilde{n}/4}2^{\tilde{n}-n}$ (exponential in n) different ciphertexts. It will take the running time of $O(\tilde{n}2^{\tilde{n}/2}2^{\tilde{n}-n}\lg^2 M)$ bit operations exhaustively to search all the possible ciphertexts of an original plaintext. Therefore, the correspondence between any arbitrary ciphertext and a related original plaintext are indistinguishable in subexponential time.

Concretely speaking, the running time of searching all the ciphertexts of an original plaintext is

$\mathcal{T}_s = (96)2^{96/2}2^{96-80}(464)^2 \approx 2^{88} > 2^{80}$ for $n = 80$, $\tilde{n} = 96$, and $\lceil \lg M \rceil = 464$;
$\mathcal{T}_s = (112)2^{112/2}2^{112-96}(544)^2 \approx 2^{98} > 2^{96}$ for $n = 96$, $\tilde{n} = 112$, and $\lceil \lg M \rceil = 544$;
$\mathcal{T}_s = (128)2^{128/2}2^{128-112}(644)^2 \approx 2^{108} \approx 2^{112}$ for $n = 112$, $\tilde{n} = 128$, and $\lceil \lg M \rceil = 640$.

6.3.2 Nonmalleability of Ciphertexts

An encryption scheme is said to be malleable if it is possible for an adversary to transform a ciphertext into another ciphertext revertible to a related plaintext. That is, given an cipher-text of a plaintext $\underline{m}$, it is possible to generate another ciphertext which can decrypt to the plaintext $f(\underline{m})$ without necessarily knowing or learning $\underline{m}$, where f is a known function [19].

By way of examples, let a RSA ciphertext $\bar{C} = \underline{m}^e \% N$, then $z^e \bar{C} = (z\underline{m})^e \% N$ is a malleation of $\bar{C}$ which decrypts to $f(\underline{m}) = z\underline{m} \% N$. Again let an ElGamal ciphertext $\bar{C} = (g^r, \underline{m}y^r \% p)$, then $(g^r, z\underline{m}y^r \% p)$ is a malleation of $(g^r, \underline{m}y^r \% p)$ which decrypts to $f(\underline{m}) = z\underline{m} \% p$. Thus, if $\underline{m}' = z\underline{m} \% p$ is known, then $\underline{m} = \underline{m}'z^{-1} \% p$ is found.

In the new cryptoscheme, there is the ciphertext $\bar{G} = E(\dot{B}) = \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i)+\neg r_i(3(i-B_i)+B_i)})^{B_i} \% M$ which takes a bit-pair as an operation unit, where $\dot{B} = B_1 \ldots B_{\tilde{n}/2}$ is a related extended plaintext, and thus, evidently the plaintext function $f(\dot{B}) = z\dot{B} \% M$ is not suitable for $\bar{G}$. Again considering that B_j occurs in the subscript of multiplied C_i, and moreover is relevant to the random bit string $r_1 \ldots r_{\tilde{n}/2}$ that can be guessed only in exponential time, it is impossi-ble to exist other plaintext function $f(\dot{B})$ which corresponds to the malleation of $E(\dot{B}) = \bar{G}$.

6.3.3 Proof of the Semantical Security

If the security requirement of a cryptoscheme can be stated formally in an antagonistic model, as opposed to heuristically, with clear assumptions that certain computational problems are intractable, and an adversary has access to he algorithms of the cryptoscheme as well as enough computational resources, the cryptoscheme possesses provable security [22][23].

Definition 11. A cryptoscheme is said to be semantically secure if an adversary who knows the encryption algorithm of the cryptoscheme and is in possession of a ciphertext is unable to determine any information about the related plaintext [22].

It is subsequently demonstrated that semantic security is equivalent to another definition of security called ciphertext indistinguishability [24]. If a cryptoscheme has the property of indistinguishability, then an adversary will be unable to distinguish a pair of ciphertexts based on the two plaintexts encrypted by a challenger.

A chosen plaintext attack (CPA) is an attack model for cryptanalysis which presumes that the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts that are expected to decrease the security of an encryption scheme [23].

Definition 12. A cryptoscheme is said to be IND-CPA (indistinguishable under chosen plaintext attack), namely semantically secure against chosen plaintext attack, if the adversary cannot determine which of the two plaintexts was chosen by a challenger, with probability significantly greater than 1/2, where 1/2 means the success rate of random guessing [23][25].

For a probabilistic asymmetric cryptoscheme based on computational security, indistinguishability under chosen plaintext attack is illuminated by a game between an adversary and a challenger, where the adversary is regarded as a probabilistic polynomial time Turing machine, which means that it must complete the game and output a guess within a polynomial number of operation steps.

Notice that for the JUNA cryptoscheme, the adversary may be also regarded as a probabilistic subexponential time Turing machine since no subexponential time solution to the MPP or ASPP is found so far.

Theorem 1. The JUNA cryptoscheme is semantically secure against chosen plaintext attack on the assumption that the MPP and ASPP cannot be solved in subexponential time.

Its proof is omitted due to limited pages.

7 Conclusion

The new cryptoscheme builds its security firmly on two intractabilities: the MPP $C_i = (A_i W^{\ell(i)})^{\delta} \% M$ with $A_i \in \Lambda$ and $\ell(i)$ from Ω and the ASPP $\bar{G} \equiv \prod_{i=1}^{\tilde{n}/2} (C_{r_i(3(i-1)+B_i)+\neg r_i(3(i-\underset{\cdot}{B}_i)+B_i)})^{B_i}$ $(\% M)$ to which no subexponential time solutions are found, and there exist only exponential time solutions so far [26], utilizes a bit-pair string to decrease the bit-length of the modulus M, exploits a bit-pair shadow string to guard against the LLL lattice basis reduction attack, and adopts the approaches of introducing a random bit string into an encryption and appending a random bit string to a plaintext to avoiding the adaptive-chosen-ciphertext attack and the meet-in-the-middle dichotomy.

As $\tilde{n} = 96$, 112, or 128, there exists $\lceil \lg M \rceil = 464$, 544, or 640, which assures that when a JUNA ciphertext $\bar{G}$ with $r_1 \ldots r_{\tilde{n}/2} = 1 \ldots 1$ is converted into an ASSP through a discrete logarithm, the density of a related ASSP knapsack is pretty high, and larger than 1.

There always exists contradiction between time and security, so does between space and security, and so does between time and space. We attempt to find a balance which is none other than a delicate thing among time, space, and security.

Acknowledgment. The authors would like to thank the Academicians Jiren Cai, Zhongyi Zhou, Jianhua Zheng, Changxiang Shen, Zhengyao Wei, Binxing Fang, Guangnan Ni, Andrew C. Yao, and Xicheng Lu for their important guidance, advice, and suggestions.

The authors also would like to thank the Professors Dingyi Pei, Jie Wang, Ronald L. Rivest, Moti Yung, Adi Shamir, Dingzhu Du, Mulan Liu, Huanguo Zhang, Dengguo Feng, Yixian Yang, Hanliang Xu, Xuejia Lai, Yongfei Han, Yupu Hu, Dongdai Lin, Rongquan Feng, Ping Luo, Jianfeng Ma, Lusheng Chen, Chuankun Wu, Tao Xie, Wenbao Han, Bogang Lin, Lequan Min, Qibin Zhai, Hong Zhu, Renji Tao, Zhiying Wang, Quanyuan Wu, and Zhichang Qi for their important counsel, suggestions, and corrections.

References

1. Su, S., Lü, S.: A Public Key Cryptosystem Based on Three New Provable Problems. Theoretical Computer Science **426–427**, 91–117 (2012)
2. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
3. Niemi, V.: A new trapdoor in knapsacks. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 405–411. Springer, Heidelberg (1991)

4. Orton, G.A.: A multiple-iterated trapdoor for dense compact knapsacks. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 112–130. Springer, Heidelberg (1995)
5. Brickell, E.F.: Solving low density knapsacks. In: Advance in Cryptology: CRYPTO 1983, pp. 25–37. Plenum Press, New York (1984)
6. Coster, M.J., Joux, A., LaMacchia, B.A., et al.: Improved Low-Density Subset Sum Algorithms. Computational Complexity **2**(2), 111–128 (1992)
7. Merkle, R.C., Hellman, M.E.: Hiding information and Signatures in Trapdoor Knapsacks. IEEE Transactions on Information Theory **24**(5), 525–530 (1978)
8. Yan, S.Y.: Number Theory for Computing, 2nd edn. Springer, Berlin (2002)
9. Hungerford, T.W.: Algebra. Springer, New York (1998)
10. Menezes, A.J., Oorschot, P.V., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, London (2001)
11. Naccache, D., Stern, J.: A new public-key cryptosystem. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 27–36. Springer, Heidelberg (1997)
12. ElGamal, T.: A Public-key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985)
13. Blake, I.F., Seroussi, G., Smart, N.P.: Elliptic Curves in Cryptography. Cambridge Univ. Press, Cambridge (1999)
14. Davis, M.: The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions. Dover Publications, Mineola (2004)
15. Du, D.Z., Ko, K.: Theory of Computational Complexity. John Wiley & Sons, New York (2000)
16. Li, T., Su, S.: Analysis of success rate of attacking knapsacks from JUNA cryptosystem by LLL lattice basis reduction. In: 9th Int. Conf. On Computational Intelligence and Security, pp. 454–458. IEEE Press, New York (2013)
17. Diffie, W., Hellman, M.E.: Exhaustive Cryptanalysis of the NBS Data Encryption Standard. Computer **10**(6), 74–84 (1977)
18. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998)
19. Dolev, D., Dwork, C., Naor, M.: Nonmalleable Cryptography. SIAM Journal on Computing **30**(2), 391–437 (2000)
20. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
21. Shoup, V.: OAEP reconsidered. In: Advance in Cryptology: Crypto 2001, pp. 239–259. Springer, New York (2001)
22. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: 14th Annual ACM Symposium on Theory of Computing, pp. 365–377. ACM, New York (1982)
23. Bellare, M.: Practice-oriented provable security. In: Okamoto, E. (ed.) ISW 1997. LNCS, vol. 1396. Springer, Heidelberg (1998)
24. Goldwasser, S., Micali, S.: Probabilistic Encryption. Journal of Computer and System Sciences **28**, 270–299 (1984)
25. Katz, J., Lindell, Y.: Introduction to Modern Cryptography: Principles and Protocols. Chapman & Hall / CRC, Boca Raton (2007)
26. Su, S., Lü, S.: REESSE1+. Reward. Proof by Experiment. A New Approach to Proof of P != NP. Cornell University Library (2009). http://arxiv.org/pdf/0908.0482 (revised 2014)

Network and Algorithms

An Algorithmic Framework for Labeling Network Maps

Jan-Henrik Haunert[1] and Benjamin Niedermann[2](✉)

[1] University of Osnabrück, Osnabrück, Germany
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
niedermann@kit.edu

Abstract. Drawing network maps automatically comprises two challenging steps, namely laying out the map and placing non-overlapping labels. In this paper we tackle the problem of labeling an already existing network map considering the application of metro maps. We present a flexible and versatile labeling model. Despite its simplicity, we prove that it is NP-complete to label a single line of the network. For a restricted variant of that model, we introduce an efficient algorithm that optimally labels a single line. Based on that algorithm, we present a general and sophisticated workflow for multiple metro lines, which is experimentally evaluated on real-world metro maps.

1 Introduction

Label placement and geographic network visualization are classical problems in cartography, which independently of each other have received the attention of computer scientists. Label placement usually deals with annotating point, line or area features of interest in a map with text labels such that the associations between the features and the labels are clear and the map is kept legible [8]. Geographic network visualization, on the other hand, often aims at a geometrically distorted representation of reality that allows information about connectivity, travel times, and required navigation actions to be retrieved easily. Computing a good network visualization is thus related to finding a layout of a graph with certain favorable properties [14]. For example, to avoid visual clutter in metro maps, an *octilinear* graph layout is often chosen, in which the orientation of each edge is a multiple of 45° [10,11,13]. Alternatively, one may choose a *curvilinear* graph layout, that is, to display the metro lines as curves [4,12].

Computing a graph layout for a metro map and labeling the stops have been considered as two different problems that can be solved in succession [13], but also integrated solutions have been suggested [10,11]. Nevertheless, in practice, metro maps are often drawn manually by cartographers or designers, as the existing algorithms do not achieve results of sufficient quality in adequate time.

This work was started at the seminar "Drawing Graphs and Maps with Curves" organized by S. Fabrikant, S. G. Kobourov, M. Nöllenburg and M. Teillaud; Schloss Dagstuhl, Germany, April 2013.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 689–700, 2015.
DOI: 10.1007/978-3-319-21398-9_54

For example, Nöllenburg and Wolff [10] report that their method needed 10 hours and 31 minutes to compute a labeled metro map of Sydney that they present in their article, while an unlabeled map for the same instance was obtained after 23 minutes—both results were obtained without proof of optimality but with similar optimality gaps. On the other hand Wang and Chi [13] present an algorithm that creates the graph layout and labeling within one second, but they cannot guarantee that labels do not overlap each other or the metro lines.

An integrated approach to computing a graph layout and labeling the stops allows consideration to be given to all quality criteria of the final visualization. On the other hand, treating both problems separately will probably reduce computation time. Moreover, we consider the labeling of a metro map as an interesting problem on its own, since, in some situations, the layout of the network is given as part of the input and must not be changed. In a semi-automatic workflow, for example, a cartographer may want to draw or alter a graph layout manually before using an automatic method to place labels, probably to test multiple different labeling styles with the drawing. Hence, a labeling algorithm is needed that is rather flexible in dealing with different labeling styles.

In this paper, we are given the layout of a metro map consisting of several metro lines on which stops (also called stations) are located. For each stop we are further given its name, which should be placed close to its position. We first introduce a versatile and general model for labeling metro maps; see Section 2. Like many labeling algorithms for point sets [1,2,5], our algorithm uses a discrete set of candidate labels for each point. Often, each label is represented by a rectangle wrapping the text. Since we also want to use curved labels, however, we represent a label by a simple polygon approximated from a *fat curve*, that is, a curve of certain width reflecting the text height. We then prove that even in that simple model labeling a single metro line is NP-complete considering different labeling styles. Hence, we restrict the set of candidates satisfying certain properties, which allows us to solve the problem on one metro line C in $O(n^2)$ time, where n is the number of stops of C; see Section 3. This algorithm optimizes the labeling with respect to a weighting function that is based on Imhof's [8] classical criteria of cartographic quality. Utilizing that algorithm, we present an efficient heuristic for labeling a metro map consisting of multiple metro lines; see Section 4. Finally, we evaluate our approach presenting experiments conducted on realistic metro maps; see Section 5. Note that "stops" on "metro lines" can refer more generally to points of interest on the lines of any kind of a network map. We address labeling styles for octilinear graph layouts and curvilinear graph layouts that use Bézier curves. The more general model behind our method, however, subsumes but is not limited to these particular styles.

We sincerely thank Herman Haverkort, Arlind Nocaj, Aidan Slingsby and Jo Wood for helpful and interesting discussions.

2 Labeling Model

We assume that the metro lines are given by directed, non-self-intersecting curves in the plane described by polylines, which for example have been derived by approximating Bézier curves. We denote that set of metro lines by $\mathcal{M}$. Further, the stops of each metro line $C \in \mathcal{M}$ are given by an ordered set $\mathcal{S}_C$ of points on C going from the beginning to the end of C. For two stops $s, s' \in \mathcal{S}_C$ we write $s < s'$ if s lies before s'. We denote the union of the stops among all metro lines by $\mathcal{S}$ and call the tuple $(\mathcal{M}, \mathcal{S})$ a *metro map*.

For each stop $s \in \mathcal{S}$ we are further given a name that should be placed close to it. In contrast to previous work, we do not follow traditional map labeling abstracting from the given text by bounding boxes. Instead we model a *label* ℓ of a stop $s \in \mathcal{S}$ as a simple polygon. For example, a label could have been derived by approximating a fat curve prescribing the name of the stop; see Fig. 1. For each stop s we are given a set $\mathcal{K}_s$ of labels, which we also call *candidates* of s. The set $\bigcup_{s \in \mathcal{S}} \mathcal{K}_s$ is denoted by $\mathcal{K}$.

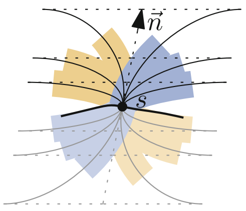


Fig. 1. Fan of candidates created by fat Bézier curves

Since "names should disturb other map content as little as possible"[8], we strictly forbid overlaps between labels and lines as well as label-label overlaps. Further, each stop must be labeled. Hence, a set $\mathcal{L} \subseteq \mathcal{K}$ is called a *labeling* if (1) no two labels of $\mathcal{L}$ intersect each other, (2) no label $\ell \in \mathcal{L}$ intersects any metro line $C \in \mathcal{M}$, and (3) for each stop $s \in \mathcal{S}$ there is exactly one label $\ell \in \mathcal{L} \cap \mathcal{K}_s$.

Definition 1 (MetroMapLabeling)
Given: *Metro map* $(\mathcal{M}, \mathcal{S})$*, candidates* $\mathcal{K}$ *and weighting function* $w \colon 2^{\mathcal{K}} \to \mathbb{R}^+$.
Find, if it exists: *Optimal labeling* $\mathcal{L}$ *of* $(\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$*, i.e.,* $w(\mathcal{L}) \leq w(\mathcal{L}')$ *for any labeling* $\mathcal{L}' \subseteq \mathcal{K}$.

The model allows us to create arbitrarily shaped label candidates for a metro map. In our evaluation we have considered two different *labeling styles*. The first style, OctilinStyle, creates for each stop a set of octilinear rectangles as label candidates; see Fig. 2. We use that style for octilinear maps. The second style, CurvedStyle, creates for each stop a set of fat Bézier curves as label candidates, which are then approximated by simple polygons; see Fig. 1. We use that style for curvilinear metro maps, in order to adapt the curvilinear style of

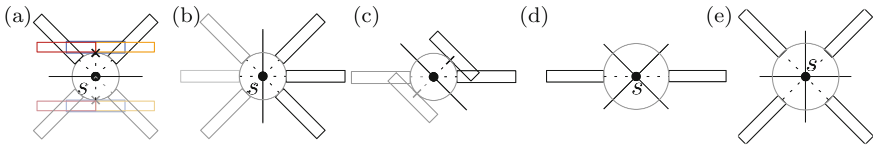


Fig. 2. Construction of octilinear candidates for a stop s of an ocitlinear metro map. (a) s lies on a horizontal segment. (b) s lies on a vertical segment. (c) s lies on a diagonal segment. (d) s lies on a crossing of two diagonal segments. (e) s lies on a crossing of a vertical and horizontal segment.

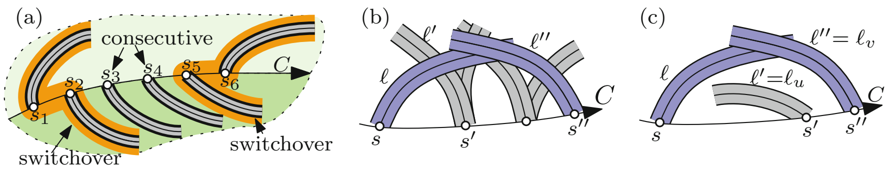


Fig. 3. (a) Consecutive stops and switchovers. (b) The candidates satisfy the transitivity property. (c) The candidates do not satisfy the transitivity property.

the metro map. The basic idea is that a label perpendicularly emanates from a stop with respect to its metro line and then becomes horizontal to sustain legibility. In the full version [7] we motivate our choice of candidates based on cartographic criteria and give detailed descriptions for both labeling styles.

Theorem 1. *It is NP-complete to decide whether a feasible labeling exists, even if the labels are based on* OctilinStyle *or* CurvedStyle *and the map has only one metro line.*

The proof uses a reduction from the NP-complete problem *monotone planar 3SAT*[9] and can be easily adapted to other labeling styles; see full version [7]. Note that the complexity of labeling points using a finite set of axis-aligned rectangular label candidates is a well-studied NP-complete problem, e.g., see [5, 6]. However, since we do not necessarily use axis-aligned rectangles as labels and since we place the labels along metro lines, it is not obvious how to reduce a point-feature labeling instance on an instance of MetroMapLabeling.

3 Labeling Algorithm for a Single Metro Line

We now study the case that the given instance $I = (\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$ consists only of one metro line C. Based on cartographic criteria we introduce three additional assumptions on I, which allows us to efficiently solve MetroMapLabeling.

For each stop $s \in S$, we assume that each candidate $\ell \in \mathcal{K}_s$ is assigned to one side of C; either ℓ is a *left candidate* assigned to the left side of C, or ℓ is a *right candidate* assigned to the right side of C. For appropriately defined candidate sets those assignments correspond with the geometric positions of the candidates, i.e., left (right) candidates lie on the left (right) hand side of C.

Assumption 1 (Separated Labels). *Candidates that are assigned to different sides of C do not intersect.*

This assumption is normally not a real restriction, because for appropriately defined candidate sets the line C separates both types of candidates geometrically. We further require what we call the *transitivity property.*

Assumption 2 (Transitivity Property). *For any three stops $s, s', s'' \in \mathcal{S}$ with $s < s' < s''$ and any three candidates $\ell \in \mathcal{K}_s$, $\ell' \in \mathcal{K}_{s'}$ and $\ell'' \in \mathcal{K}_{s''}$ assigned to the same side of C, it holds that if neither ℓ and ℓ' intersect nor ℓ' and ℓ'' intersect then also ℓ and ℓ'' do not intersect; see also Fig. 3(b)–(c).*

In our experiments we established Assumption 1 and Assumption 2 by removing candidates greedily. In Section 5 we show that for real-world metro maps and the considered candidate sets we remove only few labels, which indicates that those assumptions have only a little influence on the labelings.

Two stops $s, s' \in \mathcal{S}$ with $s < s'$ are *consecutive* if there is no other stop $s'' \in \mathcal{S}$ with $s < s'' < s'$; see Fig. 3(a). For two consecutive stops $s_1, s_2 \in \mathcal{S}$ we say that each two candidates $\ell_1 \in \mathcal{K}_{s_1}$ and $\ell_2 \in \mathcal{K}_{s_2}$ are *consecutive* and denote the set that contains each pair of consecutive labels in $\mathcal{L} \subseteq \mathcal{K}$ by $P_{\mathcal{L}} \subseteq \mathcal{L} \times \mathcal{L}$. Further, two consecutive labels $\ell_1, \ell_2 \in \mathcal{K}_C$ form a *switchover* (ℓ_1, ℓ_2) if they are assigned to opposite sides of C, whereas (ℓ_1, ℓ_2) denotes an ordered set indicating the order of the stops of ℓ_1 and ℓ_2. Two switchovers of C are *consecutive* in $\mathcal{L} \subseteq \mathcal{K}$ if there is no switchover in $\mathcal{L}$ in between of both. We define the set of all switchovers in $\mathcal{K}$ by $\mathcal{W}$ and the set of consecutive switchovers in $\mathcal{L} \subseteq \mathcal{K}$ by $\Gamma_{\mathcal{L}} \subseteq \mathcal{W} \times \mathcal{W}$.

Based on cartographic criteria extracted from Imhof's "general principles and requirements" for map labeling [8], we require a weighting function $w\colon 2^{\mathcal{K}} \to \mathbb{R}^+$ of the following form; see also the full version [7] for a detailed motivation of w.

Assumption 3 (Linear Weighting Function). *For any $\mathcal{L} \subseteq K$ we require*

$$w(\mathcal{L}) = \sum_{\ell \in \mathcal{L}} w_1(\ell) + \sum_{(\ell_1,\ell_2) \in P_{\mathcal{L}}} w_2(\ell_1, \ell_2) + \sum_{(\sigma_1,\sigma_2) \in \Gamma_{\mathcal{L}}} w_3(\sigma_1, \sigma_2),$$

where $w_1\colon \mathcal{L} \to \mathbb{R}$ rates a single label, $w_2\colon P_{\mathcal{L}} \to \mathbb{R}$ rates two consecutive labels and $w_3\colon \Gamma_{\mathcal{L}} \to \mathbb{R}$ rates two consecutive switchovers.

In particular, we define w such that it penalizes the following structures to sustain readbility. (1) Steep or highly curved labels. (2) Consecutive labels that lie on different sides of C, or that are shaped differently. (3) Consecutive switchovers that are placed close to each other.

If $I = (\{C\}, \mathcal{S}, \mathcal{K}, w)$ satisfies Assumption 1–3, we call MetroMapLabeling also SoftMetroLineLabeling. We now introduce an algorithm that solves this problem in $O(n^2k^4)$ time, where $n = |\mathcal{S}|$ and $k = \max\{|\mathcal{K}_s| \mid s \in \mathcal{S}\}$. Note that k is typically constant. We assume w.l.o.g. that $\mathcal{K}$ contains only candidates that do not intersect C. Omitted proofs are found in the full version [7].

Labels on One Side. We first assume that all candidate labels in $\mathcal{K}$ are assigned either to the left or to the right side of C; without loss of generality to the left side of C. For two stops $s, s' \in \mathcal{S}$ we denote the instance restricted to the stops $\{s, s'\} \cup \{s'' \in \mathcal{S} \mid s < s'' < s'\}$ by $I[s, s']$. We denote the first stop of C by $\underline{s}$ and the last stop by $\overline{s}$. The transitivity property directly yields the next lemma.

Lemma 1. *Let s, s' and s'' be stops with $s < s' < s''$, $\mathcal{L}$ be a labeling of $I[\underline{s}, s']$, $\ell \in \mathcal{L} \cap \mathcal{K}_s$ and $\ell' \in \mathcal{L} \cap \mathcal{K}_{s'}$. Any $\ell'' \in \mathcal{K}_{s''}$ intersecting ℓ also intersects ℓ'.*

Hence, the lemma states that ℓ' separates $\mathcal{L}$ from the candidates of the stops succeeding s'. We use this observation as follows. Based on $\mathcal{K}$ we define a directed acyclic graph $G = (V, E)$; see Fig. 4(a)–(b). This graph contains a vertex u for each candidate $\ell \in \mathcal{K}$ and the two vertices x and y. We call x the *source* and y the

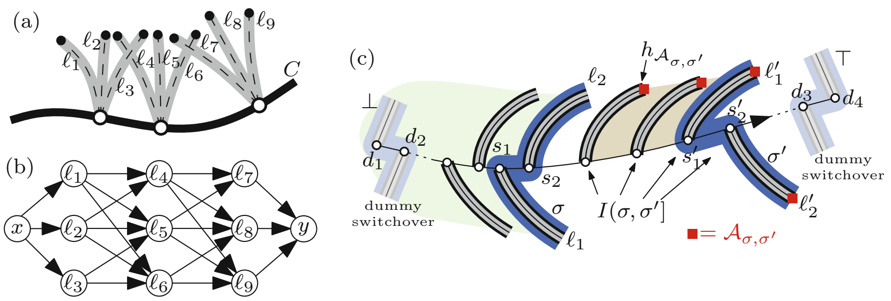


Fig. 4. Illustrations for labeling a single metro line. (a) A one-sided instance and (b) the acyclic directed graph G^I based on its labels. (c) A two-sided instance with a labeling. The switchovers σ' and σ separate the labeling into a two-sided and a one-sided instance.

target of G. Let ℓ_u denote the candidate that belongs to the vertex $u \in V \setminus \{x, y\}$. For each pair $u, v \in V \setminus \{x, y\}$ the graph contains the edge (u, v) if and only if the stop of ℓ_u lies directly before the stop of ℓ_v and, furthermore, ℓ_u and ℓ_v do not intersect. Further, for each vertex u of any candidate of $\underline{s}$ the graph contains the edge (x, u), and for each vertex u of any candidate of $\overline{s}$ the graph contains the edge (u, y). For an edge $(u, v) \in E$ we define its weight w_e as follows. For $u \neq x$ and $v \neq y$ we set $w_e = w_1(\ell_v) + w_2(\ell_u, \ell_v)$. For $x = u$ we set $w_e = w_1(\ell_v)$ and for $v = y$ we set $w_e = 0$. An *x-y path* $P \subseteq E$ in G is a path in G that starts at x and ends at y. Its weight is $w(P) = \sum_{e \in P} w_e$. The x-y path with minimum weight among all x-y path is the *shortest* x-y path.

Lemma 2. *For any x-y path P in G there is a labeling $\mathcal{L}$ of I with $w(P) = w(\mathcal{L})$ and for any labeling $\mathcal{L}$ of I there is an x-y path P in G with $w(P) = w(\mathcal{L})$.*

The lemma relies on Lemma 1 and in particular proves that a shortest x-y path P in G corresponds with an optimal labeling of I. Due to [3, Chapter 24], P can be constructed in $O(|V| + |E|)$ time using a dyn. programming approach, which we call MinPath. In particular MinPath considers each edge only once. There are $O(n \cdot k)$ vertices in G and each vertex has at most k incoming edges, which implies that there are $O(n \cdot k^2)$ edges. Since MinPath considers each edge only once, we compute the edges of G on demand, which saves storage.

Theorem 2. *If I is one-sided,* SoftMetroLineLabeling *can be optimally solved in $O(nk^2)$ time and $O(nk)$ space.*

Labels on Both Sides. If candidates lie on both sides of the metro line, we solve the problem utilizing the algorithm for the one-sided case.

Consider a labeling $\mathcal{L}$ of I and let σ, σ' be two switchovers in $\mathcal{L}$ such that σ lies before σ' and no other switchover lies in between both; see Fig. 4(c). Roughly spoken, σ and σ' induce a two-sided instance that lies before σ and a one-sided instance that lies in between both switchovers σ and σ'.

Lemma 3. *Let s, s'_1, s'_2 and s'' be stops with $s < s'_1 < s'_2 < s''$; s'_1 and s'_2 are consecutive. Let $\mathcal{L}$ be a labeling of $I[\underline{s}, s'_2]$, $\ell \in \mathcal{L} \cap \mathcal{K}_s$, $\ell'_1 \in \mathcal{L} \cap \mathcal{K}_{s'_1}$, $\ell'_2 \in \mathcal{L} \cap \mathcal{K}_{s'_2}$ s.t. (ℓ'_1, ℓ'_2) is a switchover. Any $\ell'' \in \mathcal{K}_{s''}$ intersecting ℓ intersects ℓ'_1 or ℓ'_2.*

Hence, the lemma yields that for the one-sided instance we can choose any labeling; as long as this labeling does not intersect any label of σ or σ', it composes with σ, σ' and the labeling of the two-sided instance to one labeling for the instance up to σ'. We use that observation as follows.

Let $\sigma = (\ell_1, \ell_2)$ and $\sigma' = (\ell'_1, \ell'_2)$ be two switchovers in $\mathcal{W}$. Let s_1 and s_2 be the stops of ℓ_1 and ℓ_2, and let s'_1 and s'_2 be the stops of ℓ'_1 and ℓ'_2, respectively; see Fig. 4(c). We assume that $\sigma < \sigma'$, i.e., $s_1 < s'_1$. Let $I(\sigma, \sigma']$ be the instance restricted to the stops $\{s \in \mathcal{S} \mid s_2 < s < s'_1\} \cup \{s'_1, s'_2\}$, where $(\sigma, \sigma']$ indicates that the stops of σ' belong to that instance, while the stops of σ do not.

The switchovers σ and σ' are *compatible* if ℓ_2 and ℓ'_1 are assigned to the same side of C, and there is a labeling for $I[s_1, s'_2]$ such that it contains ℓ_1, ℓ_2, ℓ'_1 and ℓ'_2 and, furthermore, σ and σ' are the only switchovers in that labeling. Let $\mathcal{L}$ be the optimal labeling among those labelings. We denote the labeling $\mathcal{L} \setminus \{\ell_1, \ell_2\}$ of $I(\sigma, \sigma']$ by $\mathcal{A}_{\sigma,\sigma'}$. Utilizing Theorem 2, we obtain $\mathcal{A}_{\sigma,\sigma'}$ in $O(n \cdot k^2)$ time.

For any labeling $\mathcal{L}$ of an instance J let $h_{\mathcal{L}} \in \mathcal{L}$ be the label of the first stop in J and let $t_{\mathcal{L}} \in \mathcal{L}$ be the label of the last stop in J; $h_{\mathcal{L}}$ is the *head* and $t_{\mathcal{L}}$ is the *tail* of $\mathcal{L}$. For technical reasons we extend $\mathcal{S}$ by the *dummy stops* d_1, d_2, d_3 and d_4 such that $d_1 < d_2 < s < d_3 < d_4$ for any stop $s \in \mathcal{S}$. For d_1 and d_2 we introduce the *dummy switchover* $\perp$ and for d_3 and d_4 the dummy switchover $\top$. We define that $\perp$ and $\top$ are compatible to all switchovers in $\mathcal{W}$ and that $\perp$ and $\top$ are compatible, if there is a one-sided labeling for I. Conceptually, each dummy switchover consists of two labels that are assigned to both sides of C. Further, neither $\perp$ nor $\top$ has any influence on the weight of a labeling. Hence, w.l.o.g. we assume that they are contained in any labeling.

Similar to the one-sided case we define a directed acyclic graph $G' = (V', E')$. This graph contains a vertex u for each switchover $\mathcal{W} \cup \{\perp, \top\}$. Let σ_u denote the switchover that belongs to the vertex $u \in V$. In particular let x denote the vertex of $\perp$ and y denote the vertex of $\top$. For each pair $u, v \in V$ the graph contains the edge (u, v) if and only if σ_u and σ_v are compatible and $\sigma_u < \sigma_v$. The weight w_e of an edge $e = (u, v)$ in G' is $w_e = w(\mathcal{A}_{\sigma_u,\sigma_v}) + w_3(\sigma_u, \sigma_v) + w_2(\ell^u_2, h_{\mathcal{A}_{\sigma_u,\sigma_v}})$, where $\sigma_u = (\ell^u_1, \ell^u_2)$. In the special case that σ_u and σ_v share a stop, we set $w_e = w_3(\sigma_u, \sigma_v) + w_2(\ell^v_1, \ell^v_2) + w_1(\ell^v_2)$, where $\sigma_v = (\ell^v_1, \ell^v_2)$.

Let P be an x-y path in G' and let $e_1 = (x = v_0, v_1), e_2 = (v_1, v_2), \dots, e_l = (v_{l-1}, v_l = y)$ be the edges of P. For a vertex v_i of P with $0 \le i \le l$ we write σ_i instead of σ_{v_i}. We denote the set $\bigcup_{i=1}^{l} A_{\sigma_{i-1},\sigma_i}$ by $\mathcal{L}_P$.

Lemma 4. *a) The graph G' has an x-y path if and only if I has a labeling.*
b) Let P be a shortest x-y path in G', then $\mathcal{L}_P$ is an optimal labeling of I.

The proof is based on Lemma 3; see [7]. By Lemma 4 a shortest x-y path P in G' corresponds with an optimal labeling $\mathcal{L}$ of C, if this exists. Using MinPath we construct P in $O(|V'| + |E'|)$ time. Since $\mathcal{W}$ contains $O(nk^2)$ switchovers, the graph G' contains $O(nk^2)$ vertices and $O(n^2k^4)$ edges. As MinPath considers

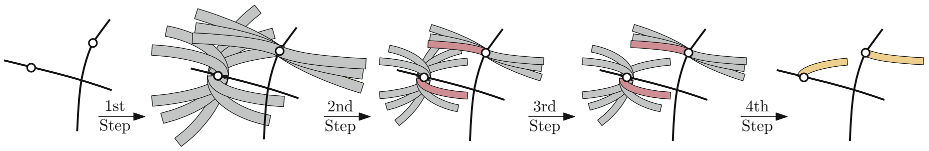


Fig. 5. Schematic illustration of the presented workflow. 1st Step: generation of candidates. 2nd Step: scaling and creation of initial labeling (red labels). 3rd Step: preselection of candidates. 4th Step: Solving the metro lines independently.

each edge only once, we compute the edges of G' on demand using $O(nk^2)$ storage. We compute the weight of the incoming edges of a vertex $v \in V'$ utilizing the one-sided case. Proceeding naively, this approach needs $O(n^3k^6)$ time overall.

Reusing already computed information, we improve that result as follows. Let $(u_1, v), \dots, (u_k, v)$ denote the incoming edges of v such that $\sigma_{u_1} \leq \dots \leq \sigma_{u_k}$, i.e., the stop of σ_{u_i}'s first label does not lie after the stop of σ_{u_j}'s first label with $i < j$. Further, let $\sigma_v = (\ell_v^1, \ell_v^2)$ and let G_i be the graph for the one-sided instance $I[s_i, s]$ considering only candidates that lie on the same side as ℓ_v^1, where s_i is the stop of the second label of σ_{u_i} and s is the stop of ℓ_v^1. Let P_i be the shortest x_i-y_i path in G_i, where x_i and y_i denote the source and target of G_i, respectively. We observe that excluding the source and target, the graph G_i is a sub-graph of G_1 for all $1 \leq i \leq k$. Further, since a sub-path of a shortest path is also a shortest path among all paths having the same end vertices, we can assume without loss of optimality that when excluding x_i and y_i from P_i, the path P_i is a sub path of P_1 for all $1 \leq i \leq k$. We therefore only need to compute G_1 and P_1 and can use sub-paths of P_1 in order to gain the weights of all incoming edges of v. Hence, we basically apply for each vertex $v \in V$ the algorithm for the one-sided case once using $O(nk^2)$ time per vertex. We then can compute the weights in $O(1)$ time per edge, which yields the next result.

Theorem 3. SoftMetroLineLabeling *can be optimally solved in* $O(n^2k^4)$ *time and* $O(nk^2)$ *space.*

4 Multiple Metro Lines

We now sketch a workflow that heuristically solves MetroMapLabeling utilizing the algorithm presented for SoftMetroLineLabeling; see Fig. 5 for a schematic illustration and the full version [7] for a detailed description.

1. *Candidate generation.* Depending on the labeling style, we generate a discrete set of candidate labels for every stop. Hence, we are given $(\mathcal{M}, \mathcal{S}, \mathcal{K}, w)$.
2. *Scaling.* We compute the scale of the map such that the existence of a feasible labeling is guaranteed. To that end we determine for each stop $s \in \mathcal{S}$ a left and a right label of $\mathcal{K}_s$. Using a simple reduction on 2SAT, we check in linear time whether those labels admit a feasible labeling. If this is not the case,

we scale the metro map. Using a binary-search-like procedure on the scaling factor we find in that manner an appropriate scaling for the metro map.

3. *Candidate pre-selection.* We greedily remove candidates from $\mathcal{K}$ such that each metro line satisfies the properties required by SoftMetroLineLabeling. Further, we ensure that no two candidates of different metro lines intersect. This allows us to label each metro line independently of the others.
4. *Final candidate selection.* For each metro line, we compute the optimal labeling using the dynamic programming approach presented in Section 3.

The workflow yields a heuristic that relies on the conjecture that using optimal algorithms in single steps is sufficient to obtain good labelings.

5 Evaluation

To evaluate our approach presented in Section 4, we did a case study on the metro systems of Sydney and Vienna, which have been used as benchmarks before [4,10,13]. For Sydney we took the curved layout from [4, Fig. 1a] and the ocitlinear layouts from [10, Fig. 9a,b], while for Vienna we took the curved layout from [4, Fig. 8c] and the ocitlinear layouts from [10, Fig. 12a,b]. Since the metro lines of Sydney are not only paths, we disassembled those metro lines into single paths. We did this by hand and tried to extract as long paths as possible. We took the positions of the stops as presented in the corresponding papers. In the curved layout of Sydney we removed the stops *Tempe* and *Martin Place* (in Fig. 6 marked as dots), because both stops are tightly enclosed by metro lines such that only the placement of very small labels is possible. This is not so much a problem of our approach, but of the given layout. In a semi-automatic approach the designer would then need to change the layout. For the curvilinear layouts we used labels of CurvedStyle and for the octilinear layouts we used labels of OctilinStyle. For the layouts of [10, Fig. 9b,13b] the authors present labelings; see Fig. 7 for an example. For any other layout they do not give labelings.

The experiments were performed on a single core of an Intel(R) Core(TM) i7-3520M CPU processor. The machine is clocked at 2.9 GHz, and has 4096 MB RAM. Our implementation is written in Java.

To assess the usefulness of the dynamic program introduced in Section 3 we further ran experiments in which the dynamic program is replaced by a simple greedy algorithm; see full version [7]. That variant of our algorithm is called Greedy, while the original one using the dynamic program is called DynProg.

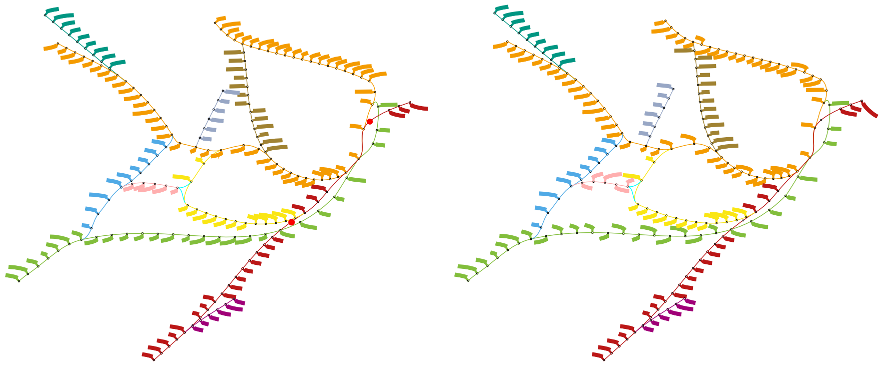
Table 1 presents our quantitative results for the considered instances. For *Sydney3* the labelings are found in Fig. 6 and for *Sydney2* a labeling created by DynProg is found in Fig. 7. The full version [7] contains all labelings.

We first note that with respect to the total number of created candidates only few labels are removed for enforcing Assumption 1 and Assumption 2; see Table 1, *A12*. This indicates that requiring those assumptions is not a real restriction on a realistic set of candidates, even though they seem to be artificial.

Running Time. Even for large networks as Sydney, our algorithm DynProg needs less than 2 seconds; see Table 1. This proves that our approach is applicable

Table 1. Experimental results for Sydney and Vienna. *Algo.*: The applied algorithm; DP=DynProg, G=Greedy. *Candidates*: No. of candidates after the first and third step. *A12*: No. of labels removed to establish Assumption 1 and Assumption 2. *Running Time*: Running time broken down into the single steps of the algorithm. Times less than 0.05 seconds are marked with $\star$. *Quality*: $\frac{w(\mathcal{L}_{\mathrm{G}})}{w(\mathcal{L}_{\mathrm{DP}})}$ = ratio between the weights of the labeling $\mathcal{L}_{\mathrm{G}}$ obtained by Greedy and $\mathcal{L}_{\mathrm{DP}}$ obtained by DynProg. SO=No. of switchovers.

				Candidates			*Running Time*					*Quality*	
Instance	*Source*	*Layout*	*Algo.*	*1st*	*3rd*	*A12*	1st	2nd	3rd	4th	$\sum$	$\frac{w(\mathcal{L}_G)}{w(\mathcal{L}_{DP})}$	SO
Sydney1	[10, Fig. 9a]	Octi.	DP	1164	1005	6	0.3	0.9	0.3	0.2	**1.7**	3.3	6
			G	1164	1006		0.3	0.9	$\star$	$\star$	**1.2**		16
Sydney2	[10, Fig. 9b]	Octi.	DP	1104	954	2	0.3	1.0	0.2	0.1	**1.6**	3.5	5
			G	1104	954		0.3	0.9	$\star$	$\star$	**1.2**		19
Sydney3	[4, Fig. 1a]	Curved	DP	1259	1020	2	0.3	1.2	0.3	0.1	**1.9**	1.2	8
			G	1259	1020		0.4	1.2	$\star$	$\star$	**1.6**		28
Vienna1	[10, Fig. 13a]	Octi.	DP	532	444	4	0.2	0.5	0.2	$\star$	**0.9**	2.1	6
			G	532	445		0.2	0.5	$\star$	$\star$	**0.7**		12
Vienna2	[10, Fig. 13b]	Octi.	DP	468	319	1	0.2	0.5	0.2	$\star$	**0.9**	2.3	7
			G	468	321		0.2	0.5	$\star$	$\star$	**0.7**		16
Vienna3	[4, Fig. 8c]	Curved	DP	600	536	0	0.3	0.9	0.2	$\star$	**1.4**	1.2	7
			G	600	536		0.3	0.9	$\star$	$\star$	**1.1**		12

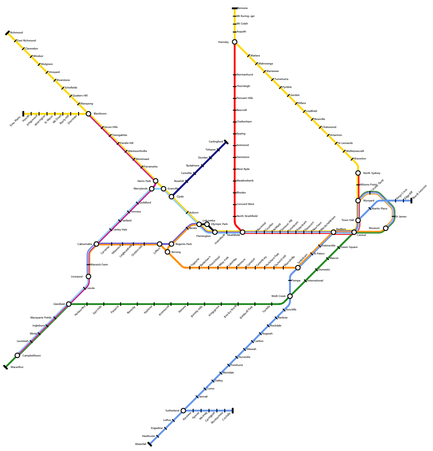


(a) *Sydney3*: DynProg, CurvedStyle Layout from [4, Fig. 1a.].

(b) *Sydney3*: Greedy, CurvedStyle Layout from [4, Fig.1a].

Fig. 6. Labelings for Sydney using DynProg and Greedy

for scenarios in which the map designer wants to adapt the layout and its labeling interactively. In particular in those scenarios not every of the four steps must


(a) Original labeling by [10]

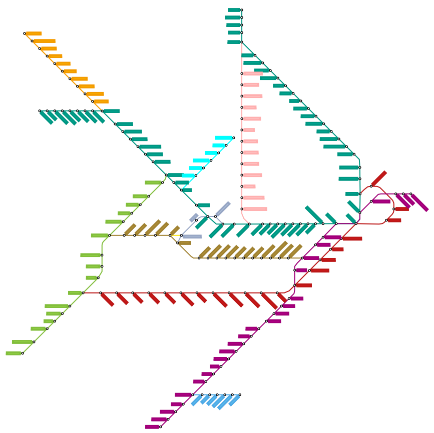

(b) *Sydney2*: DynProg, OctilinStyle Layout from [10, Fig. 9b.].

Fig. 7. Labelings for Sydney

be repeated each time, which improves computing time. For example, after once applying the scaling step (2nd step – the most time consuming step), the instance does not need to be rescaled again. Further, DynProg is only moderately slower than Greedy; 0.5 seconds in maximum, see Table 1, *Sydney1*.

Quality. We observe that in all labelings created by DynProg there are only few switchovers, namely 5–8; see Table 1, column SO. Hence, there are long *sequences* of consecutive labels that lie on the same side of their metro line; see corresponding figures. In particular the switchovers are placed such that those sequences are regularly sized. The labels of a single sequence are mostly directed into the same x-direction and in particular they are similarly shaped so that those sequences of labels form regular patterns as desired. The alignment of the labels is chosen so that they blend in with the alignment of their adjacent labels.

In contrast Greedy yields significantly more switchovers, namely 12–28. Consequently, there are many distracting switches of labels from one side to the other of the metro line; e.g. see Fig. 6. The sequences of consecutive labels lying on the same side are therefore much shorter. Further, several adjacent labels point in opposite x-directions, which results in distracting effects. Altogether, the labelings that are obtained by Greedy do not look regular, but cluttered. DynProg solves those problems since it considers the metro line *globally* yielding an optimal labeling for a single line. This observation is also reflected in Table 1, col. $\frac{w(\mathcal{L}_G)}{w(\mathcal{L}_{DP})}$, which shows that the weight computed by Greedy is significantly larger than the weight computed by DynProg. Hence, the better quality of DynProg prevails the slightly better running time of Greedy.

We observe that both Nöllenburg and Wolff's and our labelings of Sydney look quite similar, whereas our labeling has less switchovers; see Fig. 7. The same applies for the labelings of the layout of Vienna; see full version [7]. Recall that their approach needed more than 10 hours to compute a labeled metro map of Sydney. Since they need only up to 23 minutes to compute the layout without labeling, it lends itself to first apply their approach to gain a layout and then to apply our approach to construct a corresponding labeling. Similarly, it lends itself to combine our approach with that one of Wang and Chi; for a comparison see the full version [7]. Their approach is faster, however, in contrast to our approach, it does not necessarily yield occlusion-free labelings, which may result in hardly readable metro maps.

In conclusion our workflow is a reasonable alternative and improvement for the approaches presented both by Nöllenburg and Wolff, and by Wang and Chi.

References

1. Agarwal, P.K., van Kreveld, M., Suri, S.: Label placement by maximum independent set in rectangles. Comp. Geom.-Theor. Appl. **11**, 209–218 (1998)
2. Christensen, J., Marks, J., Shieber, S.: An empirical study of algorithms for point-feature label placement. Acm. T. Graphic. **14**(3), 203–232 (1995)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
4. Fink, M., Haverkort, H., Nöllenburg, M., Roberts, M., Schuhmann, J., Wolff, A.: Drawing metro maps using Bézier curves. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 463–474. Springer, Heidelberg (2013)
5. Formann, M., Wagner, F.: A packing problem with applications to lettering of maps. In: ACM Sympos. on Comput. Geom., pp. 281–288 (1991)
6. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are np-complete. Inf. Process. Lett. **12**(3), 133–137 (1981)
7. Haunert, J.-H., Niedermann, B.: An algorithmic framework for labeling network maps (2015). CoRR, abs/1505.00164
8. Imhof, E.: Positioning names on maps. Am. Cartographer, 128–144 (1975)
9. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. **11**(2), 329–343 (1982)
10. Nöllenburg, M., Wolff, A.: Drawing and labeling high-quality metro maps by mixed-integer programming. IEEE T. Vis. Comput. Gr. **17**(5), 626–641 (2011)
11. Stott, J., Rodgers, P., Martinez-Ovando, J., Walker, S.: Automatic metro map layout using multicriteria optimization. IEEE T. Vis. Comput. Gr. **17**(1), 101–114 (2011)
12. van Goethem, A., Meulemans, W., Reimer, A., Haverkort, H., Speckmann, B.: Topologically safe curved schematisation. Cartogr. J. **50**(3), 276–285 (2013)
13. Wang, Y.-S., Chi, M.-T.: Focus+context metro maps. IEEE T. Vis. Comput. Gr. **17**(12), 2528–2535 (2011)
14. Wolff, A.: Graph drawing and cartography. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, chapter 23, pp. 697–736. CRC Press (2013)

Smoothed Analysis of the Minimum-Mean Cycle Canceling Algorithm and the Network Simplex Algorithm

Kamiel Cornelissen and Bodo Manthey(✉)

University of Twente, Enschede, The Netherlands
{k.cornelissen,b.manthey}@utwente.nl

Abstract. The minimum-cost flow (MCF) problem is a fundamental optimization problem with many applications and seems to be well understood. Over the last half century many algorithms have been developed to solve the MCF problem and these algorithms have varying worst-case bounds on their running time. However, these worst-case bounds are not always a good indication of the algorithms' performance in practice. The Network Simplex (NS) algorithm needs an exponential number of iterations for some instances, but it is considered the best algorithm in practice and performs best in experimental studies. On the other hand, the Minimum-Mean Cycle Canceling (MMCC) algorithm is strongly polynomial, but performs badly in experimental studies.

To explain these differences in performance in practice we apply the framework of smoothed analysis. For the number of iterations of the MMCC algorithm we show an upper bound of $O(mn^2 \log(n) \log(\phi))$. Here n is the number of nodes, m is the number of edges, and ϕ is a parameter limiting the degree to which the edge costs are perturbed. We also show a lower bound of $\Omega(m \log(\phi))$ for the number of iterations of the MMCC algorithm, which can be strengthened to $\Omega(mn)$ when $\phi = \Theta(n^2)$. For the number of iterations of the NS algorithm we show a smoothed lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$.

1 Introduction

The minimum-cost flow (MCF) problem is a well-studied problem with many applications in, for example, modeling transportation and communication networks [1,7]. Over the last half century many algorithms have been developed to solve it. The first algorithms proposed in the 1960s were all pseudo-polynomial. These include the Out-of-Kilter algorithm by Minty [17] and by Fulkerson [8], the Cycle Canceling algorithm by Klein [13], the Network Simplex (NS) algorithm by Dantzig [5], and the Successive Shortest Path (SSP) algorithm by Jewell [11], Iri [10], and Busacker and Gowen [4]. In 1972 Edmonds and Karp [6] proposed the Capacity Scaling algorithm, which was the first polynomial MCF algorithm. In the 1980s the first strongly polynomial algorithms were developed by Tardos [24] and by Orlin [18]. Later, several more strongly polynomial algorithms

A full version with all proofs is available at http://arxiv.org/abs/1504.08251.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 701–712, 2015.
DOI: 10.1007/978-3-319-21398-9_55

were proposed such as the Minimum-Mean Cycle Canceling (MMCC) algorithm by Goldberg and Tarjan [9] and the Enhanced Capacity Scaling algorithm by Orlin [19], which currently has the best worst-case running time. For a more complete overview of the history of MCF algorithms we refer to Ahuja et al. [1].

When we compare the performance of several MCF algorithms in theory and in practice, we see that the algorithms that have good worst-case bounds on their running time are not always the ones that perform best in practice. Zadeh [25] showed that there exist instances for which the Network Simplex (NS) algorithm has exponential running time, while the Minimum-Mean Cycle Canceling (MMCC) algorithm runs in strongly polynomial time, as shown by Goldberg and Tarjan [9]. In practice however, the relative performance of these algorithms is completely different. Kovács [15] showed in an experimental study that the NS algorithm is much faster than the MMCC algorithm on practical instances. In fact, the NS algorithm is even the fastest MCF algorithm of all. An explanation for the fact that the NS algorithm performs much better in practice than indicated by its worst-case running time is that its worst-case instances are very contrived and unlikely to occur in practice. To better understand the practical performance for the NS algorithm and the MMCC algorithm, we analyze these algorithms in the framework of smoothed analysis.

Smoothed analysis was introduced by Spielman and Teng [22] to explain why the simplex algorithm usually needs only a polynomial number of iterations in practice, while in the worst case it needs an exponential number of iterations. In the framework of smoothed analysis, an adversary can specify any instance and this instance is then slightly perturbed before it is used as input for the algorithm. This perturbation can model, for example, measurement errors or numerical imprecision. In addition, it can model noise on the input that can not be quantified exactly, but for which there is no reason to assume that it is adversarial. Algorithms that have a good smoothed running time often perform well in practice. We refer to two surveys [16,23] for a summary of results that have been obtained using smoothed analysis.

We consider a slightly more general model of smoothed analysis, introduced by Beier and Vöcking [2]. In this model the adversary can not only specify the mean of the noisy parameter, but also the type of noise. We use the following smoothed input model for the MCF problem. An adversary can specify the structure of the flow network including all nodes and edges, and also the exact edge capacities and budgets of the nodes. However, the adversary can not specify the edge costs exactly. For each edge e the adversary can specify a probability density $g_e : [0,1] \to [0,\phi]$ according to which the cost of e is drawn at random. The parameter ϕ determines the maximum density of the density function and can therefore be interpreted as the power of the adversary. If ϕ is large, the adversary can very accurately specify each edge cost and we approach worst-case analysis. If $\phi = 1$, the adversary has no choice but to specify the uniform density on the interval $[0,1]$ and we have average-case analysis.

Brunsch et al. [3] were the first to show smoothed bounds on the running time of an MCF algorithm. They showed that the SSP algorithm needs $O(mn\phi)$

iterations in expectation and has smoothed running time $O(mn\phi(m+n\log(\phi)))$, since each iteration consists of finding a shortest path. They also provide a lower bound of $\Omega(m \cdot \min\{n,\phi\} \cdot \phi)$ for the number of iterations that the SSP algorithm needs, which is tight for $\phi = \Omega(n)$. These bounds show that the SSP algorithm needs only a polynomial number of iterations in the smoothed setting, in contrast to the exponential number it needs in the worst case, and explains why the SSP algorithm performs quite well in practice. In order to fairly compare the SSP algorithm with other MCF algorithms in the smoothed setting, we need smoothed bounds on the running times of these other algorithms. Brunsch et al. [3] asked particularly for smoothed running time bounds for the MMCC algorithm, since the MMCC algorithm has a much better worst-case running time than the SSP algorithm, but performs worse in practice. It is also interesting to have smoothed bounds for the NS algorithm, since the NS algorithm is the fastest MCF algorithm in practice. However, until now no smoothed bounds were known for other MCF algorithms. In this paper we provide smoothed lower and upper bounds for the MMCC algorithm, and a smoothed lower bound for the NS algorithm.

For the MMCC algorithm we prove an upper bound of $O(mn^2\log(n)\log(\phi))$ for the expected number of iterations that the MMCC algorithm needs (Section 2). For dense graphs, this is an improvement over the $\Theta(m^2n)$ iterations that the MMCC algorithm needs in the worst case, if we consider ϕ a constant (which is reasonable if it models, for example, numerical imprecision or measurement errors).

We also provide a lower bound (Section 3.1) on the number of iterations that the MMCC algorithm needs. For every n, every $m \in \{n, n+1, \ldots, n^2\}$, and every $\phi \leq 2^n$, we provide an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the MMCC algorithm requires $\Omega(m\log(\phi))$ iterations. For $\phi = \Omega(n^2)$ we can improve our lower bound (Section 3.2). We show that for every $n \geq 4$ and every $m \in \{n, n+1, \ldots, n^2\}$, there exists an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges, and $\phi = \Theta(n^2)$, for which the MMCC algorithm requires $\Omega(mn)$ iterations. This is indeed a stronger lower bound than the bound for general ϕ, since we have $m\log(\phi) = \Theta(m\log(n))$ for $\phi = \Theta(n^2)$.

For the NS algorithm we provide a lower bound (Section 4) on the number of non-degenerate iterations that it requires. In particular, we show that for every n, every $m \in \{n, \ldots, n^2\}$, and every $\phi \leq 2^n$ there exists a flow network with $\Theta(n)$ nodes and $\Theta(m)$ edges, and an initial spanning tree structure for which the NS algorithm needs $\Omega(m \cdot \min\{n,\phi\} \cdot \phi)$ non-degenerate iterations with probability 1. The existence of an upper bound is our main open problem. Note that our bound is the same as the lower bound that Brunsch et al. [3] found for the smoothed number of iterations of the SSP algorithm. This is no coincidence, since we use essentially the same instance (with some minor changes) to show our lower bound. We show that with the proper choice of the initial spanning tree structure for the NS algorithm, we can ensure that the NS algorithm performs the same flow augmentations as the SSP algorithm and therefore needs the same number of iterations (plus some degenerate ones).

In the rest of our introduction we introduce the MCF problem, the MMCC algorithm and the NS algorithm in more detail. In the rest of our paper, all logarithms are base 2.

1.1 Minimum-Cost Flow Problem

A *flow network* is a simple directed graph $G = (V, E)$ together with a nonnegative capacity function $u : E \to \mathbb{R}_+$ defined on the edges. For convenience, we assume that G is connected and that E does not contain a pair (u, v) and (v, u) of reverse edges. For the MCF problem, we also have a cost function $c : E \to [0, 1]$ on the edges and a budget function $b : V \to \mathbb{R}$ on the nodes. Nodes with negative budget require a resource, while nodes with positive budget offer it. A *flow* $f : E \to \mathbb{R}_+$ is a nonnegative function on the edges that satisfies the capacity constraints, $0 \leq f(e) \leq u(e)$ (for all $e \in E$), and flow conservation constraints $b(v) + \sum_{e=(u,v)\in E} f(e) = \sum_{e'=(v,w)\in E} f(e')$ (for all $v \in V$). The cost $c(f)$ of a flow f is defined as the sum of the flow on each edge times the cost of that edge, that is, $c(f) = \sum_{e\in E} c(e) \cdot f(e)$. The objective of the minimum-cost flow problem is to find a flow of minimum cost or conclude that no feasible flow exists.

In our analysis we often use the concept of a *residual network*, which we define here. For an edge $e = (u, v)$ we denote the reverse edge (v, u) by e^{-1}. For flow network G and flow f, the residual network G_f is defined as the graph $G_f = (V, E_f \cup E_b)$. Here $E_f = \{e \mid e \in E \text{ and } f(e) < u(e)\}$ is the set of *forward edges* with capacity $u'(e) = u(e) - f(e)$ and cost $c'(e) = c(e)$. $E_b = \{e \mid e^{-1} \in E \text{ and } f(e^{-1}) > 0\}$ is the set of *backward edges* with capacity $u'(e) = f(e^{-1})$ and cost $c'(e) = -c(e^{-1})$. Here $u'(e)$ is also called the *residual capacity* of edge e for flow f.

1.2 Minimum-Mean Cycle Canceling Algorithm

The MMCC algorithm works as follows:

- First we find a feasible flow using any maximum-flow algorithm.
- Next, as long as the residual network contains cycles of negative total cost, we find a cycle of minimum-mean cost and maximally augment flow along this cycle.
- We stop when the residual network does not contain any cycles of negative total cost.

For a more elaborate description of the MMCC algorithm, we refer to Korte and Vygen [14]. In the following, we denote the mean cost of a cycle C by $\mu(C) = \left(\sum_{e\in C} c(e)\right) / |C|$. Also, for any flow f, we denote the mean cost of the cycle of minimum-mean cost in the residual network G_f by $\mu(f)$.

Goldberg and Tarjan [9] proved in 1989 that the Minimum-Mean-Cycle Canceling algorithm runs in strongly polynomial time. Five years later Radzik and Goldberg [21] slightly improved this bound on the running time and showed that it is tight. In the following we will focus on the number of iterations the

MMCC algorithm needs, that is, the number of cycles that have to be canceled. A bound on the number of iterations can easily be extended to a bound on the running time, by noting that a minimum-mean cycle can be found in $O(nm)$ time, as shown by Karp [12]. The tight bound on the number of iterations that the MMCC algorithm needs is as follows.

Theorem 1.1 (Radzik and Goldberg). *The number of iterations needed by the MMCC algorithm is bounded by $O(nm^2)$ and this bound is tight.*

To prove our smoothed bounds in the next sections, we use another result by Korte and Vygen [14, Corollary 9.9] which states that the absolute value of the mean cost of the cycle that is canceled by the MMCC algorithm, $|\mu(f)|$, decreases by at least a factor 1/2 every mn iterations.

Theorem 1.2 (Korte and Vygen). *Every mn iterations of the MMCC algorithm, $|\mu(f)|$ decreases by at least a factor $1/2$.*

1.3 Network Simplex Algorithm

The Network Simplex (NS) algorithm starts with an initial spanning tree structure (T, L, U) and associated flow f, where each edge in E is assigned to exactly one of T, L, and U, and it holds that

- $f(e) = 0$ for all edges $e \in L$,
- $f(e) = u(e)$ for all edges $e \in U$,
- $0 \leq f(e) \leq u(e)$ for all edges $e \in T$,
- the edges of T form a spanning tree of G (if we consider the undirected version of both the edges of T and the graph G).

If the MCF problem has a feasible solution, such a structure can always be found by first finding any feasible flow and then augmenting flow along cycles consisting of only edges that have a positive amount of flow less than their capacity, until no such cycles remain. Note that the structure (T, L, U) uniquely determines the flow f, since the edges in T form a tree. In addition to the spanning tree structure, the NS algorithm also keeps track of a set of node potentials $\pi(v)$ for all nodes $v \in V$. The node potentials are defined such that the potential of a specified root node is 0 and that the potential for other nodes is such that the reduced cost $c^\pi(u, v) = c(u, v) - \pi(u) + \pi(v)$ of an edge (u, v) equals 0 for all edges $(u, v) \in T$.

In each iteration, the NS algorithm tries to improve the current flow by adding an edge to T that violates its optimality condition. An edge in L violates its optimality condition if it has strictly negative reduced cost, while an edge in U violates its optimality condition if it has strictly positive reduced cost. One of the edges e that violates its optimality condition is added to T, which creates a unique cycle C in T. Flow is maximally augmented along C, until the flow on one of the edges $e' \in C$ becomes 0 or reaches its capacity. The edge e' leaves T, after which T is again a spanning tree of G. Next we update the sets T, L, and U, the flow and the node potentials. This completes the iteration. If any edges violating their optimality condition remain, another iteration is performed. One iteration of the NS algorithm is also called a pivot. The edge e that is added to T is called the entering edge and the edge e' that leaves T is called the leaving edge. Note that in some cases the entering edge can be the same edge as the leaving edge. Also, if one of the edges in the cycle C already contains flow equal to its capacity, the flow is not changed in that iteration, but the spanning tree T still changes. Such an iteration we call degenerate.

Note that in each iteration, there can be multiple edges violating their optimality condition. There are multiple possible pivot rules that determine which edge enters T in this case. In our analysis we use the (widely used in practice) pivot rule that selects as the entering edge, from all edges violating their optimality condition, the edge for which the absolute value of its reduced cost $|c^{\pi}(e)|$ is maximum. In case multiple edges in C are candidates to be the leaving edge, we choose the one that is most convenient for our analysis.

If a strongly feasible spanning tree structure [1] is used, it can be shown that the number of iterations that the NS algorithm needs is finite. However, Zadeh [25] showed that there exist instances for which the NS algorithm (with the pivot rule stated above) needs an exponential number of iterations. Orlin [20] developed a strongly polynomial version of the NS algorithm, which uses cost-scaling. However, this algorithm is rarely used in practice and we will not consider it in the rest of our paper. For a more elaborate discussion of the NS algorithm we refer to Ahuja et al. [1].

2 Upper Bound for the MMCC Algorithm

In this section we show an upper bound of $O(mn^2 \log(n) \log(\phi))$ for the expected number of iterations that the MMCC algorithm needs starting from the initial residual network $G_{\tilde{f}}$ for the feasible starting flow $\tilde{f}$ for flow network $G = (V, E)$. Note that we assumed in Section 1.1 that G is simple and that E does not contain a pair (u, v) and (v, u) of reverse edges. This implies that for each pair of nodes $u, v \in V$, there is always at most one edge from u to v and at most one edge from v to u in any residual network G_f. We first show that the number of cycles that appears in at least one residual network G_f for a feasible flow f on G, is bounded by $(n + 1)!$, where $n = |V|$.

Lemma 2.1. *The total number of cycles that appears in any residual network G_f for a feasible flow f on G, is bounded by $(n + 1)!$.*

We next show that the probability that any particular cycle has negative mean cost close to 0 can be bounded. In the rest of this section, $\varepsilon > 0$.

Lemma 2.2. *The probability that an arbitrary cycle C has mean cost $\mu(C) \in [-\varepsilon, 0[$ can be bounded by $n\varepsilon\phi$.*

Corollary 2.3. *The probability that there exists a cycle C with $\mu(C) \in [-\varepsilon, 0[$ is at most $(n + 1)!n\varepsilon\phi$.*

Lemma 2.4. *If none of the residual networks G_f for feasible flows f on G contain a cycle C with $\mu(C) \in [-\varepsilon, 0[$, then the MMCC algorithm needs at most $mn\lceil \log_2(1/\varepsilon) \rceil$ iterations.*

Theorem 2.5. *The expected number of iterations that the MMCC algorithm needs is $O(mn^2 \log(n) \log(\phi))$.*

3 Lower Bound for the MMCC Algorithm

3.1 General Lower Bound

In this section we describe a construction that, for every n, every $m \in \{n, n+1, \dots, n^2\}$, and every $\phi \leq 2^n$, provides an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the MMCC algorithm requires $\Omega(m \log(\phi))$ iterations. For simplicity we describe the initial residual network G, which occurs after a flow satisfying all the budgets has been found, but before the first minimum-mean cycle has been canceled. For completeness, we will explain at the end of the description of G how to choose the initial network, budgets, and starting flow such that G is the first residual network.

We now describe how to construct G given n, m, and ϕ. In the following, we assume $\phi \geq 64$. If ϕ is smaller than 64, the lower bound on the number of iterations reduces to $\Omega(m)$ and a trivial instance with $\Theta(n)$ nodes and $\Theta(m)$ edges will require $\Omega(m)$ iterations. We define $k_w = \lfloor \frac{1}{2}(\log(\phi) - 4) \rfloor$ and $k_x = \lfloor \frac{1}{2}(\log(\phi) - 5) \rfloor$. Note that this implies that $k_x = k_w$ or $k_x = k_w - 1$. For the edge costs we define intervals from which the edge costs are drawn uniformly at random. We define $G = (\mathcal{V}, \mathcal{E})$ as follows.

- $\mathcal{V} = \{a, b, c, d\} \cup U \cup V \cup W \cup X$, where $U = \{u_1, \dots, u_n\}$, $V = \{v_1, \dots, v_n\}$, $W = \{w_1, \dots, w_{k_w}\}$, and $X = \{x_1, \dots, x_{k_x}\}$.
- $\mathcal{E} = E_{uv} \cup E_a \cup E_b \cup E_c \cup E_d \cup E_w \cup E_x$.
- E_{uv} is an arbitrary subset of $U \times V$ of cardinality m. Each edge (u_i, v_j) has capacity 1 and cost interval $[0, 1/\phi]$.
- E_a contains the edges (a, u_i), E_b contains the edges (u_i, b), E_c contains the edges (c, v_i), and E_d contains the edges (v_i, d) $(i = 1, \dots, n)$. All these edges have infinite capacity and cost interval $[0, 1/\phi]$.
- E_w contains the edges (d, w_i) and (w_i, a) $(i = 1, \dots, k_w)$. An edge (d, w_i) has capacity m and cost interval $[0, 1/\phi]$. An edge (w_i, a) has capacity m and cost interval $[-2^{2-2i}, -2^{2-2i} + 1/\phi]$.
- E_x contains the edges (b, x_i) and (x_i, c) $(i = 1, \dots, k_x)$. An edge (b, x_i) has capacity m and cost interval $[0, 1/\phi]$. An edge (x_i, c) has capacity m and cost interval $[-2^{1-2i}, -2^{1-2i} + 1/\phi]$.

Note that all cost intervals have width $1/\phi$ and therefore correspond to valid probability densities for the edge costs, since the costs are drawn uniformly at random from these intervals. The edges of the types (w_i, a) and (x_i, c) have a cost interval that corresponds to negative edge costs. The residual network with these negative edge costs can be obtained by having the following original instance (before computing a flow satisfying the budget requirements): All nodes, edges, costs and capacities are the same as in G, except that instead of the edges of type (w_i, a) we have edges (a, w_i) with capacity m and cost interval $[2^{2-2i} - 1/\phi, 2^{2-2i}]$ and instead of the edges of type (x_i, c) we have edges (c, x_i) with capacity m and cost interval $[2^{1-2i} - 1/\phi, 2^{1-2i}]$. In addition, node a has budget $k_w m$, node c has budget $k_x m$, the nodes of the types w_i and x_i have budget $-m$ and all other nodes have budget 0. If we now choose as the initial feasible flow the flow that sends m units from a to each node of type w_i and from c to each node of type x_i then we obtain the initial residual network G.

We now show that the MMCC algorithm needs $\Omega(m \log(\phi))$ iterations for the initial residual network G. First we make some basic observations. The minimum-mean cycle C never contains the path $P_j = (d, w_j, a)$ if the path $P_i = (d, w_i, a)$ has positive residual capacity for some $i < j$, since the mean cost of C can be improved by substituting P_j by P_i in C. Analogously, C never contains the path $P_j = (b, x_j, c)$ if the path $P_i = (b, x_i, c)$ has positive residual capacity for some $i < j$. Also, since all cycles considered have mean cost strictly less than $1/\phi$, cycles will never include more edges with cost at least $-1/\phi$ than necessary. In addition, since the edges of type (w_i, a) and (x_i, c) are saturated in the order cheapest to most expensive, none of these edges will ever be included in reverse direction in the minimum-mean cycle. The above observations lead to three candidate types for the minimum-mean cycle: cycles of type (d, w_i, a, u, v, d), of type (b, x_i, c, v, u, b), and of type $(d, w_i, a, u, b, x_j, c, v, d)$. Here u and v are arbitrary nodes in U and V, respectively. In the following series of lemmas we compare the mean costs of these cycle types. Here u and v are again arbitrary nodes in U and V, possibly different for the cycles that are compared. In our computations we always assume worst-case realization of the edge costs, that is, if we want to show that a cycle C_1 has lower mean cost than a cycle C_2, we assume that all edges in C_1 take the highest cost in their cost interval, while all edges in C_2 take the lowest cost in their cost interval (an edge that appears in both C_1 and C_2 can even take its highest cost in C_1 and its lowest cost in C_2 in the analysis).

Lemma 3.1. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, u, b)$.*

Lemma 3.2. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (d, w_{i+1}, a, u, v, d)$.*

Lemma 3.3. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (d, w_i, a, u, b, x_i, c, v, d)$.*

Lemma 3.4. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, d, w_{i+1}, a, u, b)$.*

The above observations and lemmas allow us to determine the number of iterations that the MMCC algorithm needs for residual network G.

Theorem 3.5. *The MMCC algorithm needs $m(k_w + k_x)$ iterations for residual network G, independent of the realization of the edge costs.*

Proof (sketch). Using Lemma 3.1, Lemma 3.2, Lemma 3.3, and Lemma 3.4 we show that the first m iterations cancel cycles of type (d, w_1, a, u, v, d), the next m iterations cancel cycles of type (b, x_1, c, v, u, b), the next m iterations cancel cycles of type (d, w_2, a, u, v, d), etc. The total number of iterations is therefore $m(k_w + k_x)$.

The instance G and Theorem 3.5 allow us to state a lower bound on the number of iterations that the MMCC algorithm needs in the smoothed setting.

Theorem 3.6. *For every n, every $m \in \{n, n+1, \ldots, n^2\}$, and every $\phi \leq 2^n$, there exists an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the MMCC algorithm requires $\Omega(m \log(\phi))$ iterations, independent of the realization of the edge costs.*

3.2 Lower Bound for ϕ Dependent on n

In Section 3.1 we considered the setting where ϕ does not depend on n. In this setting we showed that the MMCC algorithm needs $\Omega(m \log(\phi))$ iterations. We can improve the lower bound if ϕ is much larger than n. In this section we consider the case where $\phi = \Omega(n^2)$. In particular, we show that for every $n \geq 4$ and every $m \in \{n, \ldots, n^2\}$ there exists an instance with $\Theta(n)$ nodes, $\Theta(m)$ edges, and $\phi = \Theta(n^2)$ for which the MMCC algorithm needs $\Omega(mn)$ iterations.

The initial residual network H that we use to show our bound is very similar to the initial residual network G that was used to show the bound in Section 3.1. Below we describe the differences. We set $\phi = 400000n^2$. The constant of 400000 is large, but for the sake of readability and ease of calculations we did not try to optimize it.

- The node set W now consists of n nodes $\{w_1, \ldots, w_n\}$ and the node set X now consists of n nodes $\{x_1, \ldots, x_n\}$.
- Node a is split into two nodes a_1 and a_2. From node a_1 to a_2 there is a directed path consisting of n edges, all with infinite capacity and cost interval $[0, 1/\phi]$. Edges (a, u_i) are replaced by edges (a_2, u_i) with infinite capacity and cost interval $[0, 1/\phi]$. Edges (w_i, a) are replaced by edges (w_i, a_1) with capacity m and cost interval $[-(\frac{n-3}{n})^{2i-2}, -(\frac{n-3}{n})^{2i-2} + \frac{1}{\phi}]$.
- Node c is split into two nodes c_1 and c_2. From node c_1 to c_2 there is a directed path consisting of n edges, all with infinite capacity and cost interval $[0, 1/\phi]$. Edges (c, v_i) are replaced by edges (c_2, v_i) with infinite capacity and cost interval $[0, 1/\phi]$. Edges (x_i, c) are replaced by edges (x_i, c_1) with capacity m and cost interval $[-(\frac{n-3}{n})^{2i-1}, -(\frac{n-3}{n})^{2i-1} + \frac{1}{\phi}]$.

Note that this is a valid choice of cost intervals for the edges (w_i, a_1) and (x_i, c_1) and that they all have negative costs, since (x_n, c_1) is the most expensive of them and we have

$$-\left(\frac{n-3}{n}\right)^{2n-1}+\frac{1}{\phi} \leq -\left(1-\frac{3}{n}\right)^{2n}+\frac{1}{400000n^2} \leq -(e^{-6})^2+\frac{1}{6400000} < 0. \quad (1)$$

As in Section 3.1, there are three candidate types for the minimum-mean cost cycle: cycles of type (d, w, a, u, v, d), cycles of type (b, x, c, v, u, b), and cycles of type $(d, w, a, u, b, x, c, v, d)$. Again we assume worst-case realizations of the edge costs and compare the mean costs of cycles of the different types in a series of lemmas.

Lemma 3.7. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, u, b)$.*

Lemma 3.8. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (d, w_{i+1}, a, u, v, d)$.*

Lemma 3.9. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (d, w_i, a, u, b, x_i, c, v, d)$.*

Lemma 3.10. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, d, w_{i+1}, a, u, b)$.*

The above lemmas allow us to determine the number of iterations that the MMCC algorithm needs for initial residual network H.

Theorem 3.11. *The MMCC algorithm needs $2mn$ iterations for initial residual network H, independent of the realization of the edge costs.*

Initial residual network H and Theorem 3.11 allow us to state a lower bound for the number of iterations that the MMCC Algorithm needs in the smoothed setting for large ϕ.

Theorem 3.12. *For every $n \geq 4$ and every $m \in \{n, n+1, \ldots, n^2\}$, there exists an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges, and $\phi = \Theta(n^2)$, for which the MMCC algorithm requires $\Omega(mn)$ iterations, independent of the realization of the edge costs.*

4 Lower Bound for the Network Simplex Algorithm

In this section we provide a lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ for the number of iterations that the NS algorithm requires in the setting of smoothed analysis. The instance of the MCF problem that we use to show this lower bound is very similar to the instance used by Brunsch et al. [3] to show a lower bound on the number of iterations that the SSP algorithm needs in the smoothed setting. The differences are that they scaled their edge costs by a factor of ϕ, which we do not, that we add an extra path from node s to node t, and that the budgets of the nodes are defined slightly differently. We can show that every non-degenerate iteration of the NS algorithm for our instance corresponds with an iteration of the SSP algorithm for the instance of Brunsch et al. Because of space constraints we omit the analysis and only provide our main result.

Theorem 4.1. *For every n, every $m \in \{n, \dots, n^2\}$, and every $\phi \leq 2^n$ there exists a flow network with $\Theta(n)$ nodes and $\Theta(m)$ edges, and an initial spanning tree for which the Network Simplex algorithm needs $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ non-degenerate iterations with probability* 1.

5 Discussion

In Section 4 we showed a smoothed lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ for the number of iterations that the NS algorithm needs. This bound is the same as the smoothed lower bound that Brunsch et al. [3] showed for the SSP algorithm. For the SSP algorithm this lower bound is even tight in case $\phi = \Omega(n)$. Still, the NS algorithm is usually much faster in practice than the SSP algorithm. We believe that the reason for this difference is that the time needed per iteration is much less for the NS algorithm than for the SSP algorithm. In practical implementations, the entering edge is usually picked from a small subset (for example of size $\Theta(\sqrt{m})$) of the edges, which removes the necessity of scanning all edges for the edge which maximally violates its optimality conditions. Also, the spanning tree structure allows for fast updating of the flow and node potentials, in particular when the flow changes on only a small fraction of the edges. For the SSP algorithm an iteration consists of finding a shortest path, which takes $O(m + n\log(n))$ time. The experimental results of Kovács [15] seem to support this claim, since on all test instances the SSP algorithm is slower than the NS algorithm, but never more than a factor m. To allow a better comparison of the SSP algorithm and the NS algorithm in the smoothed setting, it would be useful to have a smoothed upper bound on the running time of the NS algorithm. Finding such an upper bound is our main open problem.

There is a gap between our smoothed lower bound of $\Omega(m\log(\phi))$ (Section 3.1) for the number of iterations that the MMCC algorithm requires and our smoothed upper bound of $O(mn^2\log(n)\log(\phi))$. Since our lower bound for the MMCC algorithm is weaker than the lower bound for the SSP algorithm, while the MMCC algorithm performs worse on practical instances than the SSP algorithm, we believe that our lower bound for the MMCC algorithm can be strengthened. Our stronger lower bound of $\Omega(mn)$ in case $\phi = \Omega(n^2)$ (Section 3.2) is another indication that this is likely possible.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin. J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall (1993)
2. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. Journal of Computer and System Sciences **69**(3), 306–329 (2004)
3. Brunsch, T., Cornelissen, K., Manthey, B., Röglin, H., Rösner, C.: Smoothed analysis of the successive shortest path algorithm. Computing Research Repository 1501.05493 [cs.DS], arXiv 2015. Preliminary version at SODA (2013)

4. Busacker, R.G., Gowen, P.J.: A procedure for determining a family of minimum-cost network flow patterns. Technical Report Technical Paper 15, Operations Research Office (1960)
5. Dantzig, G.B.: Linear programming and extensions. Rand Corporation Research Study. Princeton Univ. Press, Princeton (1963)
6. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM **19**(2), 248–264 (1972)
7. Ford, Jr. L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press (1962)
8. Fulkerson, D.R.: An out-of-kilter algorithm for minimal cost flow problems. Journal of the SIAM **9**(1), 18–27 (1961)
9. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by canceling negative cycles. J. ACM **36**(4), 873–886 (1989)
10. Iri, M.: A new method for solving transportation-network problems. Journal of the Operations Research Society of Japan **3**(1,2), 27–87 (1960)
11. Jewell, W.S.: Optimal flow through networks. Operations Research **10**(4), 476–499 (1962)
12. Karp, R.M.: A characterization of the minimum cycle mean in a digraph. Discrete Mathematics **23**(3), 309–311 (1978)
13. Klein, M.: A primal method for minimal cost flows with applications to the assignment and transportation problems. Management Science **14**(3), 205–220 (1967)
14. Korte, B., Vygen, J.: Combinatorial Optimization: Theory and Algorithms, 1st edn. Springer Publishing Company, Incorporated (2007)
15. Kovács, P.: Minimum-cost flow algorithms: An experimental evaluation. Optimization Methods and Software **30**(1), 94–127 (2015)
16. Manthey, B., Röglin, H.: Smoothed analysis: Analysis of algorithms beyond worst case. it - Information Technology **53**(6), 280–286 (2011)
17. Minty, G.J.: Monotone networks. In Proceedings of the Royal Society of London A, pp. 194–212 (1960)
18. Orlin, J.B.: Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical report, Sloan School of Management. MIT, Cambridge, Technical Report No. 1615–84 (1984)
19. Orlin, J.B.: A faster strongly polynomial minimum cost flow algorithm. Operations Research **41**(2), 338–350 (1993)
20. Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Math. Program. **77**, 109–129 (1997)
21. Radzik, T., Goldberg, A.V.: Tight bounds on the number of minimum-mean cycle cancellations and related results. Algorithmica **11**(3), 226–242 (1994)
22. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. J. ACM **51**(3), 385–463 (2004)
23. Spielman, D.A., Teng, S.-H.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. Communications of the ACM **52**(10), 76–84 (2009)
24. Tardos, É.: A strongly polynomial minimum cost circulation algorithm. Combinatorica **5**(3), 247–256 (1985)
25. Zadeh, N.: A bad network problem for the simplex method and other minimum cost flow algorithms. Mathematical Programming **5**(1), 255–266 (1973)

DD-POR: Dynamic Operations and Direct Repair in Network Coding-Based Proof of Retrievability

Kazumasa Omote and Tran Phuong Thao(✉)

Japan Advanced Institute of Science and Technology (JAIST),
1-1 Asahidai, Nomi, Ishikawa 923-1211, Japan
{omote,tpthao}@jaist.ac.jp

Abstract. POR (Proof of Retrievability) is a protocol by which clients can distribute their data to cloud servers and can check if the data stored in the servers is available and intact. Based on the POR, the network coding is applied to improve network throughput. Although many network coding-based PORs have been proposed, most of them have not achieved the following practical features: direct repair and dynamic operations. In this paper, we propose the DD-POR (Dynamic operations and Direct repair in network coding-based POR) to address these shortcomings. When a server is corrupted, the DD-POR can support the direct repair in which the data stored in the corrupted server can be repaired using the data provided directly from the healthy servers. The client is thus free from the burden of data repair. Furthermore, the DD-POR allows the client to efficiently perform dynamic operations, i.e., modification, insertion and deletion.

Keywords: Proof of retrievability · Network coding · Direct repair · Dynamic operations

1 Introduction

Since the amount of data is increasing exponentially, data storage and data management become burdensome tasks of the clients. Therefore, storage providers called clouds are proposed to allow the clients to store, manage, and share their data portably and easily from anywhere via the Internet. However, because cloud providers could not be untrustworthy, this system introduces three security challenges: data availability, data integrity and data confidentiality. Ensuring data availability and data integrity is the primary requirement before ensuring data confidentiality because data availability and data integrity are the prerequisites of the existence of a system. This paper thus focuses on data availability and data integrity. To allow the client to check whether the data stored in the cloud servers is available and intact, researchers proposed Proof of Retrievability (POR) [1–3], which is the challenge-response protocol between a client and a server. Based on the POR, the following three approaches are commonly used: replication, erasure coding, and network coding. In the replication [4–6], the client stores a file replica in each server. The client can perform periodic server checks. If a server is corrupted, the client will uses a healthy replica to repair the corruption. The drawback of this approach is the high storage cost for the redundant replicas. To address this drawback, the erasure coding was proposed [7–10]. Instead of storing file replicas as in the replication, the client stores file blocks in each server. Hence, the storage cost can be reduced. However, the drawback of this approach is that to repair a corrupted server, the client must reconstruct the original file before generating the new coded blocks. The computation cost is thus increased during data repair. To address this drawback, the network coding was proposed [11–13] in which the client does not need to reconstruct the original file before repairing the corruption. Instead, the client retrieves the coded blocks in the healthy servers to generate the new file blocks. Therefore, this paper focuses on the network coding. Furthermore, the data stored in the servers cannot be checked without an additional information: Message Authentication Code (MAC) (used for a symmetric key setting), or signature (used for an asymmetric key setting). A MAC is also called a *tag*. This paper is based on a symmetric key setting which is well-known to be more efficient than an asymmetric key setting. The MAC is thus used in the proposed scheme.

Network Coding-Based POR. Dimakis et al. [14] were the first to apply the network coding to distributed storage systems and achieve a reduction in the communication overhead of the repair component. Li et al. [15] introduced the tree-structure data regeneration with the linear network coding to achieve an efficient regeneration traffic and bandwidth capacity by using an undirected-weighted maximum spanning tree. Chen et al. then proposed the Remote Data Checking for Network Coding-based distributed storage system (RDC-NC) [16] which provides an elegant data repair by recoding encoded blocks in healthy servers during repair. H.Chen et al. proposed the NC-Cloud scheme [17] to improve cost-effective repair using the functional minimum-storage regenerating (FMSR) codes and lighten the encoding requirement of storage nodes during repair. However, most of these schemes have the following shortcomings. Firstly, the schemes can only support the indirect repair. That is, to repair a corrupted server, the client must require the healthy servers to provide aggregated coded blocks and aggregated tags, and send them back to the client. The client then checks the provided coded blocks using the tags, and computes the new coded blocks and new tags to replace the corruption. The client sends these new coded blocks and tags to the new server. Such a repair mechanism is a troublesome task for the client. Because the data repair is performed very often during the system lifetime, the client thus incurs high computation and communication costs. Secondly, the schemes do not consider the dynamic operations. That is, the client can only perform the data check and data retrieval, but cannot perform the modification, insertion and deletion. A few PORs were proposed to deal with the dynamic operations, e.g, [18–21]. However, all these schemes are based on the erasure coding, not the network coding.

There are two most notable schemes which are related to our proposed scheme. The first one is the MD-POR [22], which can support the direct repair, but cannot support the dynamic operations. The second one is the NC-Audit [23], which also considered the direct repair and dynamic operations. However, when the direct repair is supported, this scheme cannot prevent the pollution attack which is a common attack of the network coding. This is because the new server cannot check the provided coded blocks. Furthermore, the authors only discuss about the dynamic operations without clear details. For example, for the modification, the authors discuss how to update the tag without mentioning how to update the coded blocks which are related to the modified file block. For the insertion, the authors mentioned that the insertion does not work in their scheme. For the deletion, there is no concrete explanation.

Contribution. This paper proposes the DD-POR scheme with the following contributions:

- Direct repair: when a server is corrupted, the healthy servers will provide their coded blocks and tags directly to the new server without sending them back to the client. Then, the new server can check them to prevent the pollution attack, and can compute the new coded blocks and the tags for itself. The client is thus free from the repair process.
- Dynamic operations: the client not only can check and retrieve the data, but also can modify, insert and delete the data.
- Symmetric key setting: our scheme does not use any public key for the efficiency. The direct repair feature introduces a difficulty that how to allow the new server which is untrusted to check and compute the new coded blocks and the tags without using a public key, our scheme can address this problem by using an orthogonal key technique called InterMac [24].

Roadmap. The backgrounds of the POR, network coding and InterMac are described in Section 2. The adversarial model is presented in Section 3. The DD-POR scheme is proposed in Section 4. The security and efficiency analyses are given in Section 5 and 6. The conclusion is drawn in Section 7.

2 Background

2.1 The POR Framework

The POR [1–3] is a challenge-response protocol between a verifier V (client) and a prover P (server), and consists of the functions defined below.

- $\mathsf{KGen}(\lambda) \rightarrow \kappa$: run by V. This function takes a security parameter s as the input and outputs a secret key κ (For a asymmetric key system, κ is a public/private key pair.)
- $\mathsf{Encode}(F, \kappa) \rightarrow F'$: run by V. This function takes an original file F and κ as input, and outputs an encoded file F^*, and then sends F^* to P.
- $\mathsf{Check}() \rightarrow \{\mathsf{accept/deny}\}$: run by both V and P. Firstly, V generates a challenge c and sends c to P. P then computes a response r and sends r back to V. V finally verifies P based on c and r.

– Repair(): run by V. When a corruption is detected, V executes this function to repair the corruption. The technique of repair depends on each specific scheme, i.e, replication, erasure coding or network coding.

2.2 Network Coding in Distributed Storage System

The network coding [11–13] is proposed for cost-efficiency in data transmission. The model system consists of a client and multiple servers. The client owns a file F and wants to store the redundant coded blocks in the servers in a way that the client can reconstruct F and can repair the coded blocks in a corrupted server. The client firstly divides F into m blocks: $F = v_1||\cdots||v_m \in \mathbb{F}_q^z$. Each $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$ and $\mathbb{F}_q^z$ denotes a z-dimensional finite field over a prime q. The client then augments v_k with a vector of length m which contains a single '1' in the position k and $(m-1)$ single '0's elsewhere. The resulting block is called *augmented block* (says, w_k). w_k has the following form:

$$w_k = (v_k, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m}) \in \mathbb{F}_q^{z+m} \tag{1}$$

Thereafter, the client randomly chooses m coefficients $\alpha_1, \cdots, \alpha_m \in \mathbb{F}_q$ to compute coded blocks using the linear combination $c = \sum_{k=1}^{m} \alpha_k \cdot w_k \in \mathbb{F}_q^{z+m}$. The clients stores these coded blocks in the servers. To reconstruct F, any m coded blocks are required to solve m augmented blocks $w_1, \cdots, w_m$ using the accumulated coefficients contained in the last m coordinates of each coded block. After the m augmented blocks are solved, m file blocks $v_1, \cdots, v_m$ are obtained from the first coordinate of each augmented block. Finally, F is reconstructed by concatenating the file blocks. Note that the matrix consisting of the coefficients used to construct any m coded blocks should have full rank. R. Koetter et al. [13] proved that if the prime q is chosen large enough and the coefficients are chosen randomly, the probability for the matrix having full rank is high. When a server is corrupted, the client repairs it by retrieving the coded blocks from the healthy servers and linearly combining them to regenerate the new coded blocks. An example of the data repair is given in Figure 1.

2.3 InterMac

When we deal with the direct repair, the difficulty is how to allow the new server which is untrusted to check the provided coded blocks using its key without learning the secret key of the client. The InterMac [24] is a suitable technique to generate such a key for the new server. Basically, the InterMac is proposed to generate a vector which is orthogonal to a given set of vectors. Formally, given the set of vectors $\{w_1, \cdots, w_m\}$ and an integer $p \in \mathbb{Z}_q^*, p \neq 1$, the algorithm outputs a vector k_p s.t. $k_p \cdot w_k = 0, \forall k \in \{1, \cdots, m\}$.
(1) The main algorithm InterMac is described as follows:
InterMac$(\{w_1, \cdots, w_m\}, p) \rightarrow k_p$:
– Find the span π of $w_1, \cdots, w_m \in \mathbb{F}_q^{z+m}$.
– Construct the matrix M in which $\{w_1, \cdots, w_m\}$ are the rows of M.

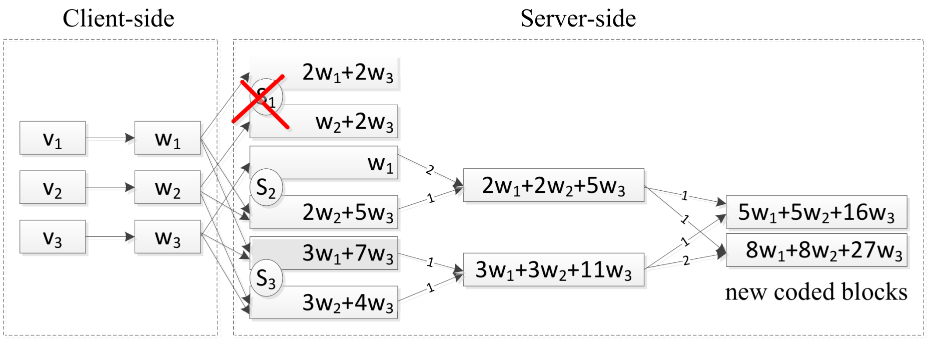


Fig. 1. The client stores the coded blocks in the server S_1, S_2, S_3. Suppose that S_1 is corrupted, the client repair it by the linear combinations of the coded blocks from S_2 and S_3.

- Find the null-space of M, denoted by $\pi_M^\perp$, which is the set of all vectors $u \in \mathbb{F}_q^{z+m}$ s.t. $M \cdot u^T = 0$.
- Find the basis vectors of $\pi_M^\perp$, denoted by $b_1, \cdots, b_z \in \mathbb{F}_q^{z+m}$ // Theorem 1 will explain why the number of the basis vectors is z.
- Let $B = \{b_1, \cdots, b_z\}$
- Compute $k_p \leftarrow \mathsf{Kg}(B)$.

(2) The sub-algorithm Kg used in InterMac is given as follows:
$\mathsf{Kg}(B = \{b_1, \cdots, b_z\}) \rightarrow k_p$:
- Let f be a Pseudorandom function s.t. $\mathcal{K} \times \mathcal{P} \times [1, z] \rightarrow \mathbb{F}_q$.
- Generate $r_i \leftarrow f(k, p, i) \in \mathbb{F}_q, \forall i \in \{1, \cdots, z\}$ where $k \in \mathcal{K}, p \in \mathcal{P}$.
- Compute $k_p \leftarrow \sum_{i=1}^{z} r_i \cdot b_i \in \mathbb{F}_q^{z+m}$.

Theorem 1. *Given* $\{w_1, \cdots, w_m\} \in \mathbb{F}_q^{z+m}$, *the number of basis vectors of* $\pi_M^\perp$ *is* z.

Proof. $\mathsf{rank}(M) = m$. Let π_M be the space spanned by the rows of M. For any $m \times (z+m)$ matrix, the rank-nullity theorem gives: $\mathsf{rank}(M) + \mathsf{nullity}(M) = z+m$ where $\mathsf{nullity}(M)$ is the dimension of $\pi_M^\perp$. Therefore, $\mathsf{dim}(\pi_M^\perp) = (z+m) - m = z$. In other words, the number of basis vectors of $\pi_M^\perp$ is z. In the InterMac, we denote the basis vectors by $\{b_1, \cdots, b_z\}$. □

3 Adversarial Model

In our scheme, the client is trusted, and the servers are untrusted. Assume that the servers do not collude with each other. The servers can perform two types of attacks:

(1) In the check phase: the servers disrupts the system or modifies the data. These attacks can be commonly prevented by the tags, we thus do not focus on these attacks.

(2) In the repair phase: the servers can perform: (i) the pollution attack which is a common attack of the network coding, and (ii) the curious attack which is a special attack of the direct repair. We focus on these in the security analysis.

- *Pollution Attack.* A malicious server firstly uses a valid coded block to pass the check phase, but then injects an invalid coded block in the repair phase to prevent data repair. An example is given as follows:
  - Encode: the client encodes the augmented blocks (w_1, w_2, w_3) to six coded blocks: c_{11}, c_{12} (stored in the server S_1), c_{21}, c_{22} (stored in the server S_2), and c_{31}, c_{32} (stored in the server S_3). Suppose that S_1 will perform a pollution attack.
  - Check: S_3 is corrupted.
  - Repair: S_3 should be repaired by two coded blocks: c'_{31} (which is a linear combination of c_{11} and c_{12}) and c'_{32} (which is a linear combination of c_{21} and c_{22}). However, S_1 is not detected because this time is the repair phase, not the check phase. The client still thinks S_1 is healthy, and thus the client requests the coded blocks from S_1 and S_2. S_1 will provide an invalid coded block c''_{31} to the client instead of c'_{31}.
- *Curious Attack.* This attack is performed by the new server in the repair phase. Every repair time, the new server is given a key k_r constructed from the secret key k_C of the client and a variant k_p. Having k_r, the new server tries to obtain k_C in order to pass the check phases in the later epochs.

4 Our Proposed Scheme

4.1 Notations

Throughout the DD-POR scheme, the following notations and definitions are used:

- $\mathcal{C}$ denotes the client.
- F denotes the original file.
- m denotes the number of file blocks.
- n denotes the number of servers.
- d denotes the number of coded blocks in each server.
- k denotes the file block index ($k \in \{1, \cdots, m\}$).
- i denotes the server index ($i \in \{1, \cdots, n\}$).
- j denotes the coded block index in each server ($j \in \{1, \cdots, d\}$).
- v_k denotes the file block ($k \in \{1, \cdots, m\}$).
- w_k denotes the augmented block of v_k.
- t_{w_k} denote the tag of w_k.
- S_i denotes the server.
- c_{ij} denotes the j-th coded block stored in S_i.
- $t_{c_{ij}}$ denotes the tag of c_{ij}.
- l denotes the number of healthy servers used for data repair.
- S_r denotes the corrupted server.
- S'_r denotes the new server which is used to replace S_r.
- $\mathbb{F}_q^z$ is the z-dimensional finite field $\mathbb{F}$ over a prime q.

4.2 Construction

Setup
(1) *Create augmented blocks*: $\mathcal{C}$ divides F into m blocks $F = v_1 || \cdots || v_m$. Each $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$. $\mathcal{C}$ creates m augmented blocks as Eq. 1.
(2) *Keygen*: $\mathcal{C}$ generates two types of keys:

– *The key of the client* (k_C): $k_C \overset{rand}{\leftarrow} \mathbb{F}_q^{z+m}$.
– *The one-time key of the new server every repair time* (k_r): $\mathcal{C}$ generates $p \overset{rand}{\leftarrow} \mathbb{Z}_q^*, p \neq 1$ (p is generated every repair time). $\mathcal{C}$ computes a key $k_p \leftarrow \mathsf{InterMac}(\{w_1, \cdots, w_m\}, p)$. The property of k_p is that $k_p \cdot w_k = 0$ where $k \in \{1, \cdots, m\}$. Let $k_r = k_C + k_p \in \mathbb{F}_q^{z+m}$. k_C is static, only k_p is re-computed every repair time. k_r is sent to the new server only when the repair phase is executed.

Encode
(1) $\mathcal{C}$ computes a tag for each augmented block as follows:

– For $\forall k \in \{1, \cdots, m\}$, $\mathcal{C}$ computes tag: $t_{w_k} = w_k \cdot k_C \in \mathbb{F}_q$.

(2) $\mathcal{C}$ computes nd coded blocks and nd corresponding tags as follows:
For $\forall i \in \{1, \cdots, n\}, \forall j \in \{1, \cdots, d\}$:

– $\mathcal{C}$ generates m coefficients: $\alpha_{ijk} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $k \in \{1, \cdots, m\}$.
– $\mathcal{C}$ computes code block: $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot w_k \in \mathbb{F}_q^{z+m}$.
– $\mathcal{C}$ compute tag: $t_{c_{ij}} = \sum_{k=1}^{m} \alpha_{ijk} \cdot t_{w_k} \in \mathbb{F}_q$.

(3) $\mathcal{C}$ sends d pairs of $\{c_{ij}, t_{c_{ij}}\}$ where $j \in \{1, \cdots, d\}$ to the server S_i.

Check
(1) $\mathcal{C}$ requires S_i to provide its aggregated coded block and aggregated tag.
(2) S_i ($i \in \{1, \cdots, n\}$) combines its coded blocks and tags as follows:

– S_i generates d coefficients: $\beta_{ij} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $\forall j \in \{1, \cdots, d\}$.
– S_i combines coded blocks: $c_{S_i} = \sum_{j=1}^{d} \beta_{ij} \cdot c_{ij} \in \mathbb{F}_q^{z+m}$.
– S_i combines tags: $t_{S_i} = \sum_{j=1}^{d} \beta_{ij} \cdot t_{c_{ij}} \in \mathbb{F}_q$.
– S_i sends $\{c_{S_i}, t_{S_i}\}$ to $\mathcal{C}$.

(3) $\mathcal{C}$ verifies S_i as follows:
For $\forall i \in \{1, \cdots, n\}$:

– $\mathcal{C}$ computes $t'_{S_i} = c_{S_i} \cdot k_C \in \mathbb{F}_q$.
– $\mathcal{C}$ checks the following equation. If it holds, $\mathcal{S}_i$ is healthy. Otherwise, $\mathcal{S}_i$ is corrupted.

$$t_{S_i} = t'_{S_i} \tag{2}$$

Repair. Suppose S_r is corrupted. A set of l healthy servers $\{S_{i_1}, \cdots, S_{i_l}\}$ are required to provide data to a new server S'_r, which is used to replace S_r.
(1)S_i ($i \in \{i_1, \cdots, i_l\}$) provides its data to S'_r as follows:

– S_i generates d coefficients: $\beta_{ij} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $j \in \{1, \cdots, d\}$.
– S_i combines coded blocks: $c_{S_i} = \sum_{j=1}^{d} \beta_{ij} \cdot c_{ij} \in \mathbb{F}_q^{z+m}$.
– S_i combines tags: $t_{S_i} = \sum_{j=1}^{d} t_{ij} \cdot \beta_{ij} \in \mathbb{F}_q$.
– S_i sends $\{c_{S_i}, t_{S_i}\}$ to S'_r.

(2) S'_r checks $S_i (i \in \{i_1, \cdots, i_l\})$ as follows:
- S'_r computes $t'_{S_i} = c_{S_i} \cdot k_r \in \mathbb{F}_q$.
- S'_r checks if the following equation holds:

$$t'_{S_i} = t_{S_i} \tag{3}$$

(3) S'_r computes d new coded blocks and d corresponding tags:
For $\forall j \in \{1, \cdots, d\}$:
- S'_r generates l coefficients: $\gamma_{ri} \overset{rand}{\longleftarrow} \mathbb{F}_q$ where $i \in \{i_1, \cdots, i_l\}$.
- S'_r computes new coded block: $c_{rj} = \sum_{i=i_1}^{i_l} \gamma_{ri} \cdot c_{S_i} \in \mathbb{F}_q^{z+m}$.
- S'_r computes new tag: $t_{rj} = \sum_{i=i_1}^{i_l} \gamma_{ri} \cdot t_{S_i} \in \mathbb{F}_q$.

4.3 Correctness

(1) We firstly prove the correctness of Eq. 2:
$t_{S_i} = \sum_{j=1}^{d} \beta_{ij} \cdot t_{c_{ij}} = \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot t_{w_k} = \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot w_k \cdot k_C$.
$t'_{S_i} = c_{S_i} \cdot k_C = \sum_{j=1}^{d} \beta_{ij} \cdot c_{ij} \cdot k_C = \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot w_k \cdot k_C = t_{S_i}$.
(2) We prove the correctness of Eq. 3
$t_{S_i} = \sum_{j=1}^{d} \beta_{ij} \cdot t_{c_{ij}} = \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot t_{w_k} = \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot w_k \cdot k_C$.
$t'_{S_i} = c_{S_i} \cdot k_r = \sum_{j=1}^{d} \beta_{ij} \cdot c_{ij} \cdot k_r = \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot w_k \cdot (k_C + k_p)$
$= \sum_{j=1}^{d} \sum_{k=1}^{m} \beta_{ij} \cdot \alpha_{ijk} \cdot w_k \cdot k_C$ // $k_p \cdot w_k = 0$
$= t_{S_i}$

4.4 Dynamic Operations

When $\mathcal{C}$ performs a dynamic operation on a file block, herein introduces a difficulty of the task: how the servers deal with the coded blocks which are related to the modified/inserted/deleted file block. The trivial solution is to encode data again. This solution incurs very high costs. In our solution, the old coded blocks and tags stored in the servers can be re-used, and only a small additional computation is needed for the dynamic operations.

Firstly, we give the following theorem, which will form the basis of the dynamic operations.

Theorem 2. *The basis vector of the matrix consisting of m augmented blocks is unique.*

Proof. Let M be the matrix in which each of m augmented blocks is a row in M:

$$M = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{pmatrix} = \begin{pmatrix} v_1, 1, 0, 0, \cdots, 0 \\ v_2, 0, 1, 0, \cdots, 0 \\ \vdots \\ v_m, 0, \cdots, 0, 1 \end{pmatrix}$$

Because the dimension of M is $m \times (m+1)$, the number of pivot variables is m and the number free variables is $(m+1) - m = 1$. Thus, the number of basis vectors of M is 1. $\square$

Modification. Suppose that $\mathcal{C}$ modifies a file block v_X to a new file block v'_X. Let w_X and w'_X be the augmented block of v_X and v'_X, respectively.

(1) *$\mathcal{C}$ re-computes k_r for the next repair time*:

Let M be the matrix consisting of m augmented blocks. After the modification, only v_X is changed and other elements in M are not changed. Namely, M is changed to M' as follows:

$$M = \begin{pmatrix} v_1, 1, 0, 0, \cdots, 0 \\ v_2, 0, 1, 0, \cdots, 0 \\ \vdots \\ \boxed{v_X}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \vdots \\ v_m, 0, \cdots, 0, 1 \end{pmatrix} \overset{modification}{\rightarrow} M' = \begin{pmatrix} v_1, 1, 0, 0, \cdots, 0 \\ v_2, 0, 1, 0, \cdots, 0 \\ \vdots \\ \boxed{v'_X}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \vdots \\ v_m, 0, \cdots, 0, 1 \end{pmatrix}$$

Observer that the modification does not affect k_C, but affect $k_r(= k_C + k_p)$ because k_p is constructed from M. This is why we need to update k_r.

– Because the number of columns in M is $(m + 1)$, and because the number of basis vectors of M is 1 (Theorem 2), we have that the unique basis vector of M consists of $(m + 1)$ elements (says, $B = [b_1, \cdots, b_{m+1}]$). Similarly, the unique basis vector of M' also consists of $(m + 1)$ elements (says, $B' = [b'_1, \cdots, b'_{m+1}]$). We need to find B' from B.
– Because only v_X in M is changed and other elements are not changed, $\mathcal{C}$ only needs to re-compute the $(X+1)$-th element of B by $(-v'_X \cdot b_1) \mod q$. Namely,

$$B' = [b_1, \cdots, b_X, \boxed{(-v'_X \cdot b_1) \mod q}, b_{X+2}, \cdots, b_{m+1}] \tag{4}$$

– $\mathcal{C}$ then computes $k'_p \leftarrow \mathsf{Kg}(B')$ (described in Section 2.3). $\mathcal{C}$ finally sends $k'_r = k_C + k'_p$ to the new server when the next repair phase is executed.

(2) *$\mathcal{C}$ computes the tag for w'_k. Each server updates its coded blocks and tags*:

– $\mathcal{C}$ computes the tag: $t'_X = w'_X \cdot k_C \in \mathbb{F}_q$, and sends $\{w'_X, t'_X\}$ to each S_i.
– S_i updates its coded blocks and tags as follows:
For $\forall j \in \{1, \cdots, d\}$: $\begin{cases} c'_{ij} = c_{ij} - \alpha_{ijX} \cdot w_X + \alpha_{ijX} \cdot w'_X = c_{ij} + \alpha_{ijX} \cdot \Delta w \\ t'_{ij} = t_{ij} - \alpha_{ijX} \cdot t_X + \alpha_{ijX} \cdot t'_X = t_{ij} + \alpha_{ijX} \cdot \Delta t \end{cases}$
where c_{ij} and t_{ij} are the old coded block and tag. $\Delta w = w'_X - w_X$, $\Delta t = t'_X - t_X$. The coefficient α_{ijX} can be found in the $(X + 1)$-th element of c_{ij}.

Insertion. Suppose that $\mathcal{C}$ inserts a file block v_I after the existing file block v_X. Let w_I be the augmented block of v_I.

(1) *$\mathcal{C}$ modifies k_C*:

Before the insertion, because an augmented block has $(m + 1)$ elements ($w_k = (v_k, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m})$), thus k_C also has $(m + 1)$ elements (says, $k_C = [k_1, \cdots, k_{m+1}]$). After the insertion, because an augmented block has

$(m+2)$ elements ($w'_k = (v_k, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m+1})$), and thus the new k'_C also has $(m+2)$ elements (says, $k'_C = [k'_1, \cdots, k'_{m+2}]$). Given k_C, we find k'_C as follows:

$$k'_C = [k_1, \cdots, k_{X+1}, \boxed{k_I}, k_{X+2}, \cdots, k_{m+1}] \tag{5}$$

- The first $(X+1)$ elements of k'_C are the same as these of k_C.
- The $(X+2)$-th element of k'_C (denoted by k_I) is computed as: $k_I \overset{rand}{\leftarrow} \mathbb{F}_q$.
- The last $(m-X)$ elements of k'_C are the same as the last $(m-X)$ elements of k_C.

The reason that we construct such k'_C will be explained in Step 3 (tag update).
(2) *$\mathcal{C}$ re-computes k_r for the next repair time*:
After the insertion, the matrix M is changed as follows:
- In each of the first X rows, a single '0' is padded in the final position.
- In the inserted row (w_I), v_I is the first element, a '1' bit is the $(X+1)$-th element counted from the second element and '0' elsewhere.
- In each of the last $(m-X)$ rows, a single '0' is padded in the final position and the single '1' is shipped to the next right position.

$$M = \underbrace{\begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ \boxed{v_X}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \boxed{v_{X+1}}, \underbrace{0, \cdots, 0, 1}_{X+1}, 0, \cdots, 0 \\ \vdots \\ v_m, 0, \cdots, 0, 1 \end{pmatrix}}_{m \times (m+1)} \overset{insertion}{\longrightarrow} M' = \underbrace{\begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ v_X, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \boxed{v_I}, \underbrace{0, \cdots, 0, 1}_{X+1}, 0, \cdots, 0 \\ v_{X+1}, \underbrace{0, \cdots, 0, 1}_{X+2}, 0, \cdots, 0 \\ \vdots \\ v_m, \underbrace{0, \cdots, 0, 1}_{m+1} \end{pmatrix}}_{(m+1) \times (m+2)}$$

We now update k'_r as follows:
- Let $B = [b_1, \cdots, b_{m+1}]$ and $B' = [b'_1, \cdots, b'_{m+2}]$ be the basis vector of M and M', respectively. Given B, we firstly find B':
  - The first $(X+1)$ elements of B' are the same as these of B:
  $$(b'_1 = b_1), \cdots, (b'_{X+1} = b_{X+1})$$
  - The last $(m-X+1)$ elements of B' are simply computed as:
  $$\begin{cases} b'_{X+2} = (-v_I \cdot b_1) \mod q \\ b'_t = b_{t-1} \quad where \quad t \in \{X+3, \cdots, m+2\} \end{cases}$$

  In other words:
$$B' = [b_1, \cdots, b_{X+1}, \boxed{(-v_I \cdot b_1) \mod q}, b_{X+2}, \cdots, b_{m+1}] \tag{6}$$

- $\mathcal{C}$ then computes $k'_p \leftarrow \mathsf{Kg}(B')$ (described in Section 2.3). $\mathcal{C}$ finally sends $k'_r = k'_C + k'_p$ to the new server when the next repair phase is executed.

(3) *$\mathcal{C}$ computes a tag for w_I. Each server S_i updates its coded blocks and tags:*

- $\mathcal{C}$ computes a tag for w_I as: $t_I = w_I \cdot k'_C$, then sends $\{w_I, t_I\}$ to S_i.
- S_i updates its coded blocks as follows:
  An old coded block has $(m+1)$ elements: $c_{ij} = (\sum_{k=1}^{m} \alpha_{ijk} \cdot w_k, \alpha_{ij1}, \cdots, \alpha_{ijm})$. Let $c_{ij}[x]$ denote the x-th element of c_{ij} ($x \in \{1, \cdots, m+1\}$). We compute the new coded block as:

$$\begin{aligned} c'_{ij} &= (\textstyle\sum_{k=1}^{m} \alpha_{ijk} w_k + \alpha_{ijI} w_I, \alpha_{ij1}, \cdots, \alpha_{ijX}, \boxed{\alpha_{ijI}}, \alpha_{ij(X+1)}, \cdots, \alpha_{ijm}) \\ &= (c_{ij}[1] + \alpha_{ijI} w_I, c_{ij}[2], \cdots, c_{ij}[X+1], \boxed{\alpha_{ijI}}, c_{ij}[X+2], \cdots, c_{ij}[m+1]) \end{aligned} \quad (7)$$

  where $\alpha_{ijI} \leftarrow \mathbb{F}_q$.
- S_i updates its tags as follows:
  By constructing k'_C as Step 1, the tags of the augmented blocks before the insertion are:

$$\begin{pmatrix} t_{w_1} \\ \vdots \\ t_{w_X} \\ t_{w_{(X+1)}} \\ \vdots \\ t_{w_m} \end{pmatrix} = M \cdot k_C = \begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ \boxed{v_X}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \boxed{v_{X+1}}, \underbrace{0, \cdots, 0, 1}_{X+1}, 0, \cdots, 0 \\ \vdots \\ v_m, 0, \cdots, 0, 1 \end{pmatrix} \begin{pmatrix} k_1 \\ \vdots \\ k_{(X+1)} \\ k_{(X+2)} \\ \vdots \\ k_{m+1} \end{pmatrix} = \begin{pmatrix} v_1 k_1 + k_2 \\ \vdots \\ v_X k_1 + k_{(X+1)} \\ v_{(X+1)} k_1 + k_{(X+2)} \\ \vdots \\ v_m k_1 + k_{m+1} \end{pmatrix}$$

  The tags of the augmented blocks after the insertion are:

$$\begin{pmatrix} t'_{w_1} \\ \vdots \\ t'_{w_X} \\ \boxed{t_I} \\ t'_{w_{(X+1)}} \\ \vdots \\ t'_{w_{m+1}} \end{pmatrix} = M' \cdot k'_C = \begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ v_X, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \boxed{v_I}, \underbrace{0, \cdots, 0, 1}_{X+1}, 0, \cdots, 0 \\ v_{X+1}, \underbrace{0, \cdots, 0, 1}_{X+2}, 0, \cdots, 0 \\ \vdots \\ v_m, \underbrace{0, \cdots, 0, 1}_{m+1} \end{pmatrix} \begin{pmatrix} k_1 \\ \vdots \\ k_{(X+1)} \\ \boxed{k_I} \\ k_{(X+2)} \\ \vdots \\ k_{m+1} \end{pmatrix} = \begin{pmatrix} v_1 k_1 + k_2 \\ \vdots \\ v_X k_1 + k_{(X+1)} \\ \boxed{v_I k_1 + k_I} \\ v_{(X+1)} k_1 + k_{(X+2)} \\ \vdots \\ v_m k_1 + k_{m+1} \end{pmatrix}$$

Observe that before and after the insertion, the first $(X+1)$ tags and the last $(m-X)$ tags are not changed; only a new tag t_I, which is the tag of w_I, is inserted. Furthermore, the old tag of c_{ij} is computed as $t_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot t_{w_k}$. We are now ready to compute the tag for c'_{ij} as follows:

$$t_{c'_{ij}} = \sum_{k=1}^{X} \alpha_{ijk} t_{w_k} + \boxed{\alpha_{ijI}} t_I + \sum_{k=X+1}^{m} \alpha_{ijk} t_{w_k} = t_{c_{ij}} + \boxed{\alpha_{ijI} t_I} \quad (8)$$

where α_{ijI} is the same as in Eq. 7.

Deletion. Suppose that $\mathcal{C}$ deletes the X-th file block (v_X). Let w_X be the augmented block of v_X.

(1) *$\mathcal{C}$ modifies k_C*:

Similar to the insertion operation, before the deletion, the key of $\mathcal{C}$ has the form: $k_C = [k_1, \cdots, k_{m+1}]$. After the deletion, $\mathcal{C}$ simply removes the $(X+1)$-th element in k_C. Namely,

$$k'_C = [k_1, \cdots, k_X, k_{X+2}, \cdots, k_{m+1}] \quad (9)$$

The reason to construct such k'_C will be explained in Step 3 (tag update).

(2) *$\mathcal{C}$ re-computes k_r for the next repair time*:

After the deletion, the matrix M is now changed as follows:

– In each of the first $(X-1)$ rows, the single '0' in the final position is removed.
– The X-th row is removed.
– In each of the last $(m-X)$ rows, the single '1' is shipped to the previous left position and the single '0' in the final position is removed.

$$M = \underbrace{\begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ v_{X-1}, \underbrace{0, \cdots, 0, 1}_{X-1}, 0, \cdots, 0 \\ \boxed{v_X}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ v_{X+1}, \underbrace{0, \cdots, 0, 1}_{X+1}, 0, \cdots, 0 \\ \vdots \\ v_m, \underbrace{0, \cdots, 0, 1}_{m} \end{pmatrix}}_{m \times (m+1)} \overset{deletion}{\rightarrow} M' = \underbrace{\begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ v_{X-1}, \underbrace{0, \cdots, 0, 1}_{X-1}, 0, \cdots, 0 \\ v_{X+1}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \vdots \\ v_m, \underbrace{0, \cdots, 0, 1}_{m-1} \end{pmatrix}}_{(m-1) \times m}$$

We now update k'_r as follows:

– Let $B = [b_1, \cdots, b_{m+1}]$ be the basis vector of M.
– To compute the basis vector B' of M', $\mathcal{C}$ simply removes the $(X+1)$-th element of B. Namely,

$$B' = [b_1, \cdots, b_X, b_{X+2}, \cdots, b_{m+1}] \quad (10)$$

– $\mathcal{C}$ then computes $k'_p \leftarrow \mathsf{Kg}(B')$ (Section 2.3). $\mathcal{C}$ finally sends $k'_r = k'_C + k'_p$ to the new server when the next repair phase is executed.

(3) *S_i updates its coded blocks and tags*:

– S_i updates its coded blocks:
Because an old coded block has $(m+1)$ elements: $c_{ij} = (\sum_{k=1}^{m} \alpha_{ijk} \cdot w_k, \alpha_{ij1}, \cdots, \alpha_{ijm})$ and let $c_{ij}[x]$ denote the x-th element of c_{ij} where $x \in \{1, \cdots, m+1\}$, we compute the new coded block as follows:

$$\begin{aligned} c'_{ij} &= (\textstyle\sum_{k=1}^{X-1} \alpha_{ijk} w_k + \sum_{k=X+1}^{m} \alpha_{ijk} w_k, \alpha_{ij1}, \cdots, \alpha_{ij(X-1)}, \alpha_{ij(X+1)}, \cdots, \alpha_{ijm}) \\ &= (c_{ij}[1] - \alpha_{ijX} w_X, c_{ij}[2], \cdots, c_{ij}[X], c_{ij}[X+2], \cdots, c_{ij}[m+1]) \end{aligned} \tag{11}$$

where $\alpha_{ijX} = c_{ij}[X+1]$.

– S_i updates its tags as follows:
By constructing k'_C as Step 1, the tags of the augmented blocks before the deletion are:

$$\begin{pmatrix} t_{w_1} \\ \vdots \\ t_{w_{X-1}} \\ t_{w_X} \\ t_{w_{(X+1)}} \\ \vdots \\ t_{w_m} \end{pmatrix} = M \cdot k_C = \begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ v_{X-1}, \underbrace{0, \cdots, 0, 1}_{X-1}, 0, \cdots, 0 \\ \boxed{v_X}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ v_{X+1}, \underbrace{0, \cdots, 0, 1}_{X+1}, 0, \cdots, 0 \\ \vdots \\ v_m, 0, \cdots, 0, 1 \end{pmatrix} \begin{pmatrix} k_1 \\ \vdots \\ k_{(X+1)} \\ \vdots \\ k_{m+1} \end{pmatrix} = \begin{pmatrix} v_1 k_1 + k_2 \\ \vdots \\ v_{X-1} k_1 + k_X \\ v_X k_1 + k_{(X+1)} \\ v_{X+1} k_1 + k_{(X+2)} \\ \vdots \\ v_m k_1 + k_{m+1} \end{pmatrix}$$

The tags of all augmented blocks after the insertion are:

$$\begin{pmatrix} t'_{w_1} \\ \vdots \\ t'_{w_{X-1}} \\ t'_{w_{(X+1)}} \\ \vdots \\ t'_{w_{m+1}} \end{pmatrix} = M' \cdot k'_C = \begin{pmatrix} v_1, 1, 0, \cdots, 0 \\ \vdots \\ v_{X-1}, \underbrace{0, \cdots, 0, 1}_{X-1}, 0, \cdots, 0 \\ v_{X+1}, \underbrace{0, \cdots, 0, 1}_{X}, 0, \cdots, 0 \\ \vdots \\ v_m, \underbrace{0, \cdots, 0, 1}_{m-1} \end{pmatrix} \begin{pmatrix} k_1 \\ \vdots \\ k_X \\ k_{(X+2)} \\ \vdots \\ k_{m+1} \end{pmatrix} = \begin{pmatrix} v_1 k_1 + k_2 \\ \vdots \\ v_{X-1} k_1 + k_X \\ v_{(X+1)} k_1 + k_{(X+2)} \\ \vdots \\ v_m k_1 + k_{m+1} \end{pmatrix}$$

Observe that before and after the deletion, the X-th tag is removed and the other tags are still the same. Furthermore, the old tag of c_{ij} is computed as: $t_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot t_{w_k}$. We are now ready to compute the tag for c'_{ij} as follows:

$$t_{c'_{ij}} = \sum_{k=1}^{X-1} \alpha_{ijk} t_{w_k} + \sum_{k=X+1}^{m} \alpha_{ijk} t_{w_k} = t_{c_{ij}} - \boxed{\alpha_{ijX} t_X} \tag{12}$$

where α_{ijX} is the same as in Eq. 11.

5 Security Analysis

We firstly show our scheme is secure from the pollution attack and curious attack as follows.

Theorem 3. *The DD-POR is secured from the pollution and curious attacks.*

Proof. (1) Pollution attack: suppose that $\mathcal{A}$ is a malicious server in a set of l servers which are used to repair the corrupted server. $\mathcal{A}$ injects an invalid pair of $(c_\mathcal{A}, t_\mathcal{A})$ to the new server S'_r. Then S'_r will check $(c_\mathcal{A}, t_\mathcal{A})$ using the key $k_r \in \mathbb{F}_q^{z+m}$. Because S'_r is assumed to not collude with the other servers, $k_r \in \mathbb{F}_q^{z+m}$ is only known by S'_r. Thus, $\mathcal{A}$ can only pass the verification of S'_r with a probability $1/q^{z+m}$ via the brute-force search. If q is chosen large enough (e.g, 160 bits), the probability is $1/(2^{160})^{z+m}$, which is negligible.

Consider that the S'_r itself is a malicious server who will perform the pollution attack in the next epoch. Even though S'_r holds k_r, S'_r cannot pass the verification in the repair phase because k_r is a one-time repair key. Another new server will be given a key $k'_r \neq k_r$.

(2) Curious attack: the new server is given the key $k_r = k_C + k_p \in \mathbb{F}_q^{z+m}$. Similar to the pollution attack, the probability of the new server to learn k_C is $1/q^{z+m}$ via the brute-force search. This probability is from learning k_C directly or learning k_p and then obtaining k_C by $k_C = k_r - k_p$. If q is chosen large enough (e.g, 160 bits), the probability is $1/(2^{160})^{z+m}$, which is negligible. □

We also describe the condition to reconstruct F via the following theorem.

Theorem 4. *F can be reconstructed if in any epoch, at least l out of n servers collectively store m coded blocks which are linearly independent combinations of m augmented blocks, and the matrix consisting of the accumulated coefficients has full rank (i.e, the rank is m).*

Proof. S_i contains d coded blocks: $\{c_{ij}\}(j \in \{1, \cdots, d\})$. c_{ij} is computed from m augmented blocks $w_1, \cdots, w_m$ by $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot w_k \in \mathbb{F}_q^{z+m}$. Therefore, to reconstruct F, m augmented blocks are viewed as the unknowns that need to be solved. To solve these unknowns, at least m coded blocks are required s.t the coefficient matrix has full rank.

$$\begin{cases} c_{(ij)_1} = \sum_{k=1}^{m} \alpha_{(ijk)_1} \cdot w_k \\ c_{(ij)_2} = \sum_{k=1}^{m} \alpha_{(ijk)_2} \cdot w_k \\ \cdots \\ c_{(ij)_m} = \sum_{k=1}^{m} \alpha_{(ijk)_m} \cdot w_k \end{cases}$$

Let l be the number of servers ($l < n$) which collectively stores these m coded blocks. Because each server stores d coded blocks, $l = \lceil \frac{m}{d} \rceil$. □

6 Efficiency Analysis

The efficiency comparison is given in Table 1. Because the MD-POR and NC-Audit schemes focus on the public authentication, the system models have one more entity called TPA (Third Party Auditor) who is delegated the task of checking the servers by $\mathcal{C}$. For the fair comparison, we assume that the check task in these schemes is performed by $\mathcal{C}$.

Table 1. The comparison

		RDC-NC [16]	**MD-POR** [22]	**NC-Audit** [23]	**DD-POR**
Feature	Direct repair	No	Yes	Not completed (*)	Yes
	Dynamic operations	No	No	No	Yes
	Symmetric key	Yes	Yes	Yes	Yes
Encode Computation	Client-side	$O(mnd)$	$O(mnd)$	$O(mnd)$	$O(mnd)$
	Server-side	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Check Computation	Client-side	$O(n)$	$O(n)$	$O(n)$	$O(n)$
	Server-side	$O(dn)$	$O(dn)$	$O(dn)$	$O(dn)$
Repair Computation	Client-side	$O(dl)$	$O(1)$	$O(1)$	$O(1)$
	Server-side	$O(dl)$	$O(dl)$	$O(dl)$	$O(dl)$
Modification Computation	Client-side	N/A	N/A	N/A	$O(1)$
	Server-side	N/A	N/A	N/A	$O(dn)$
Insertion Computation	Client-side	N/A	N/A	N/A	$O(1)$
	Server-side	N/A	N/A	N/A	$O(dn)$
Deletion Computation	Client-side	N/A	N/A	N/A	$O(1)$
	Server-side	N/A	N/A	N/A	$O(dn)$

N/A means not applicable due to the lack of support. (*) In the NC-Audit, the direct repair can lead to the pollution attack because the new server cannot check the provided coded blocks.

Encode Computation. In all the schemes, $\mathcal{C}$ needs $O(m)$ to compute m tags for m augmented blocks, and $O(mnd)$ to compute nd coded blocks along with the tags. The complexity on the client-side is thus $O(mnd)$. Meanwhile, the servers only need to receive the coded blocks and tags from $\mathcal{C}$ without any computation. The complexity on the server-side is thus $O(1)$.

Check Computation. In all the schemes, $\mathcal{C}$ needs $O(1)$ to verify the aggregated coded block and tag of each server. Therefore, the complexity on the client-side is $O(n)$ to verify n servers. Meanwhile, each server needs to combine d coded blocks and d tags to compute the aggregated coded block and aggregated tag, respectively. Therefore, the complexity of n servers is $O(dn)$.

Repair Computation. In the RDC-NC scheme, $\mathcal{C}$ needs $O(l)$ to check l pairs of the provided coded block and tag from the healthy servers, and needs $O(dl)$ to compute d pairs of new coded blocks and tags using the linear combinations of l pairs of the provided coded blocks and tags. Therefore, the complexity on the client-side is $O(dl)$. In the MD-POR, NC-Audit and DD-POR schemes, the complexity on the client-side is $O(1)$ because $\mathcal{C}$ does not need to do anything due to the direct repair.

In the RDC-NC scheme, each of l servers combines its d coded blocks and d tags to compute the aggregated coded block and aggregated tag, respectively. Therefore, the complexity on the server-side is $O(dl)$. In the MD-POR, NC-Audit and DD-POR schemes, l healthy servers perform as in the RDC-NC ($O(dl)$), and the new server performs the task of $\mathcal{C}$ as in the RDC-NC ($O(dl)$). Therefore, the complexity on the server-side is $O(dl)$.

Modification Computation. In the DD-POR, $\mathcal{C}$ only needs $O(1)$ to re-compute k_r (Step 1), and $O(1)$ to compute the new tag of the modified augmented block (Step 2). Therefore, the complexity on the client-side is $O(1)$. Meanwhile, each server needs $O(d)$ to update the coded blocks and tags (Step 2). Therefore, the complexity of n servers is $O(dn)$.

Insertion Computation. In the DD-POR, $\mathcal{C}$ only needs $O(1)$ to modify k_C (Step 1), $O(1)$ to re-compute k_r (Step 2), and $O(1)$ to compute the tag of the inserted augmented block (Step 3). Therefore, the complexity on the client-side is $O(1)$. Meanwhile each server needs $O(d)$ to update the coded blocks and tags (Step 3). Therefore, the complexity of n servers is $O(dn)$.

Deletion Computation. In the DD-POR, $\mathcal{C}$ only needs $O(1)$ to modify k_C (Step 1), and $O(1)$ to re-compute k_r (Step 2). Thus, the complexity on the client-side is $O(1)$. Meanwhile each server needs $O(d)$ to update the coded blocks and tags (Step 3). Thus, the complexity of n servers is $O(dn)$.

7 Conclusion

This paper proposes a network coding-based POR scheme, name DD-POR, to support the direct repair and the dynamic operations in a symmetric key setting. The idea is based on the InterMac technique which can generate a key such s.t the key is orthogonal to the augmented blocks. The security analysis is showed to prevent the pollution attack and curious attack. The efficiency analysis is given based on complexity theory to compare with the previous scheme.

References

1. Juels, A., Kaliski, B.: PORs: Proofs of retrievability for large files. In: 14th ACM Computer and Communications Security Conf., CCS, pp. 584–597 (2007)
2. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)

3. Bowers, K., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: ACM Workshop on Cloud Computing Security, CCSW, pp. 43–54 (2009)
4. Bolosky, W.J., Douceur, J.R., Ely, D., Theimer, M.: Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. SIGMETRICS **2000**, 34–43 (2000)
5. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR-PDP: Multiple-Replica Provable Data Possession. In: 28th Distributed Computing Systems Conf., ICDCS, pp. 411–420 (2008)
6. Zhang, Z., Lian, Q., Lin, S., Chen, W., Chen, Y., Jin, C.: Bitvault: A highly reliable distributed data retention platform. ACM SIGOPS Operating Systems Review **41**(2), 27–36 (2007)
7. Aguilera, M.K., Janakiraman, R., Xu, L.: Efficient fault-tolerant distributed storage using erasure codes. Tech. Rep., Washington University in St. Louis (2004)
8. Bowers, K., Juels, A., Oprea, A.: HAIL: A high-availability and integrity layer for cloud Storage. In: 16th ACM Computer and Communications Security Conf., CCS, pp. 187–198 (2009)
9. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of Retrievability via Hardness Amplification. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 109–127. Springer, Heidelberg (2009)
10. Hendricks, J., Ganger, G.R., Reiter, M.: Verifying distributed erasure-coded data. In: 26th ACM Principles of Distributed Computing Symposium, pp. 163–168 (2007)
11. Ahlswede, R., Cai, N., Li, S., Yeung, R.: Network information flow. IEEE Trans. **46**(4), 1204–1216 (2000)
12. Li, S., Yeung, R., Cai, N.: Linear Network Coding. IEEE Trans. **49**(2), 371–381 (2003)
13. Koetter, R., Muriel, M.: An Algebraic Approach to Network Coding. IEEE/ACM Trans. on Networking (TON) **11**(5), 782–795 (2003)
14. Dimakis, A., Godfrey, P., Wu, Y., Wainwright, M., Ramchandran, K.: Network coding for distributed storage systems. IEEE Trans. Information Theory **56**(9), 4539–4551 (2010)
15. Li, J., Yang, S., Wang, X., Xue, X., Li, B.: Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding. In: 29th IEEE Information Commun. Conf., pp. 2892–2900 (2000)
16. Chen, B., Curtmola, R., Ateniese, G., Burns, R.: Remote Data Checking for Network Coding-based Distributed Storage Systems. In: ACM Workshop on Cloud Computing Security, pp. 31–42 (2010)
17. Chen, H.C.H., Hu, Y., Lee, P.P.C., Tang, Y.: NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds. IEEE Trans. on Computers **63**(1), 31–44 (2014)
18. Cash, D., Küpçü, A., Wichs, D.: Dynamic Proofs of Retrievability via Oblivious RAM. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 279–295. Springer, Heidelberg (2013)
19. Elaine, S., Emil, S., Charalampos, P.: Practical dynamic proofs of retrievability. In: CCS, pp. 325–336 (2013)
20. Chen, B., Curtmola, R.: Robust dynamic remote data checking for public clouds. In: Proc of. ACM Conf. on Computer and Communications Security, CCS, pp. 1043–1045 (2012)
21. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling Public Auditability and Data Dynamics for Storage Security in cloud Computing. IEEE Trans. Parallel and Distributed System **22**(5), 847–859 (2011)

22. Omote, K., Thao, T.: MD-POR: Multi-source and Direct Repair for Network Coding-based Proof of Retrievability. Int. Journal of Distributed Sensor Networks (IJDSN) ArticleID:586720, January 2015
23. Le, A., Markopoulou, A.: NC-Audit: Auditing for network coding storage. In: NetCod 2012, pp. 155–160 (2012)
24. Le, A., Markopoulou, A.: On detecting pollution attacks in inter-session network coding. 31st IEEE Conf. on Computer Communications, INFOCOM, pp. 343–351 (2012)

Evaluating Bayesian Networks via Data Streams

Andrew McGregor and Hoa T. Vu(✉)

University of Massachusetts, Amherst, USA
{mcgregor,hvu}@cs.umass.edu

Abstract. Consider a stream of n-tuples that empirically define the joint distribution of n discrete random variables $X_1, \ldots, X_n$. Previous work of Indyk and McGregor [6] and Braverman et al. [1,2] addresses the problem of determining whether these variables are n-wise independent by measuring the ℓ_p distance between the joint distribution and the product distribution of the marginals. An open problem in this line of work is to answer more general questions about the dependencies between the variables. One powerful way to express such dependencies is via Bayesian networks where nodes correspond to variables and directed edges encode dependencies. We consider the problem of testing such dependencies in the streaming setting. Our main results are:

1. A tight upper and lower bound of $\tilde{\Theta}(nk^d)$ on the space required to test whether the data is consistent with a given Bayesian network where k is the size of the range of each X_i and d is the max in-degree of the network.
2. A tight upper and lower bound of $\tilde{\Theta}(k^d)$ on the space required to compute any 2-approximation of the log-likelihood of the network.
3. Finally, we show space/accuracy trade-offs for the problem of independence testing using ℓ_1 and ℓ_2 distances.

1 Introduction

The problem of testing n-wise independence in data streams has attracted recent attention in streaming algorithms literature [1,2,6]. In that problem, the stream consists of a length m sequence of n-tuples that empirically defines a joint distribution of n random variables $X_1, X_2, \ldots, X_n$ where each X_i has range $[k] := \{1, 2, \ldots k\}$. Specifically, the stream defines the joint probability mass function (pmf):

$$\mathcal{P}(x_1, \ldots, x_n) = \mathbb{P}(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) := \frac{f(x_1, x_2, \ldots, x_n)}{m} , \quad (1)$$

where $f(x_1, x_2, \ldots, x_n)$ is the number of tuples equal to $(x_1, x_2, \ldots, x_n)$. The marginal probability of a subset of variables $\{X_j\}_{j \in S}$ is defined as:

$$\mathbb{P}(X_j = x_j \quad \forall j \in S) := \sum_{x_\ell \in [k] \text{ for all } \ell \notin S} \mathbb{P}(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) .$$

This work was supported by NSF Awards CCF-0953754, IIS-1251110, CCF-1320719, and a Google Research Award.

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 731–743, 2015.
DOI: 10.1007/978-3-319-21398-9_57

The goal of the previous work was to determine whether this distribution is close to being a product distribution or equivalently, whether the corresponding random variables are close to being independent by estimating:

$$\left(\sum_{x_1,\ldots,x_n \in [k]} \left| \mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) - \mathbb{P}(X_1 = x_1) \ldots \mathbb{P}(X_n = x_n) \right|^p \right)^{1/p}$$
$$:= \left\| \mathbb{P}(X_1, \ldots, X_n) - \mathbb{P}(X_1) \ldots \mathbb{P}(X_n) \right\|_p := \mathcal{E}_p(\emptyset).$$

However, it is natural to ask more general questions about the dependencies between the variables, e.g., can we identify an X_i such that the other random variables are independent conditioned on X_i or whether there is an ordering $X_{\sigma(1)}, X_{\sigma(2)}, X_{\sigma(3)}, \ldots$ such that $X_{\sigma(i)}$ is independent of $X_{\sigma(1)}, X_{\sigma(2)}, \ldots, X_{\sigma(i-2)}$ conditioned on $X_{\sigma(i-1)}$.

The standard way to represent such dependencies is via Bayesian networks. A Bayesian network is an acyclic graph G with a node X_i corresponding to each variable X_i along with a set of directed edges E that encode a factorization of the joint distribution. Specifically, if $\mathrm{Pa}(X_i) = \{X_j : (X_j \to X_i) \in E\}$ are the parents of X_i in G then the Bayesian network represents the assertion that for all $x_1, x_2, \ldots, x_n$, the joint distribution can be factorized as follows:

$$\mathbb{P}(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_{i=1}^{n} \mathbb{P}(X_i = x_i | X_j = x_j \ \forall\ X_j \in \mathrm{Pa}(X_i)) \ .$$

For example, $E = \emptyset$ corresponds to the assertion that the X_i are fully independent whereas the graph on nodes $\{X_1, X_2, X_3\}$ with directed edges $X_1 \to X_2, X_1 \to X_3$ corresponds to the assertion that X_2 and X_3 are independent conditioned on X_1.

Bayesian networks have been extensively studied and applied in artificial intelligence, machine learning, data mining, and other areas. In these applications the focus is typically on Bayesian networks where d is small, since we wish to be able to compactly represent the joint distribution through local conditional probability distributions.

In this paper we consider the problem of evaluating how well the observed data fits a Bayesian network. The data stream of tuples in $[k]^n$ and a Bayesian network G defines an empirical distribution $\mathcal{P}_G$ with the pmf:

$$\mathcal{P}_G(x_1, \ldots, x_n) := \prod_{i=1}^{n} \mathbb{P}(X_i = x_i | X_j = x_j \ \forall\ X_j \in \mathrm{Pa}(X_i)), \tag{2}$$

where

$$\mathbb{P}(X_i = x_i | X_j = x_j \text{ for all } j \in \mathrm{Pa}(X_i)) = \frac{\mathbb{P}(X_i = x_i)}{\mathbb{P}(X_j = x_j, \ \forall X_j \in \mathrm{Pa}(X_i))}. \tag{3}$$

is just the fraction of tuples whose ith coordinate is x_i amongst the set of tuples whose jth coordinate is x_j for all $X_j \in \mathrm{Pa}(X_i)$. We then define the error of G

to be the ℓ_p norm, for $p \in \{1, 2\}$, of the difference between the joint distribution and the factorization $\mathcal{P}_G$:

$$\mathcal{E}_p(G) := \Big(\sum_{x_1,\ldots,x_n \in [k]} |\mathcal{P}(x_1, \ldots, x_n) - \mathcal{P}_G(x_1, \ldots, x_n)|^p \Big)^{\frac{1}{p}} := \|\mathcal{P} - \mathcal{P}_G\|_p.$$

Clearly, if the factorization implied by G is valid then $\mathcal{E}_p(G) = 0$. More generally, if $\mathcal{E}_p(G)$ is small then we consider the factorization to be close to valid. The use of ℓ_p distance to measure "closeness" was considered previously in the Bayesian network literature [7,11]. However, the space required to compute these measures was considered a major drawback because it was assumed that it would be necessary to explicitly store the full joint distribution whose space complexity is $O(k^n)$. Our results show that this is not the case. Note that when G is the empty graph, $\mathcal{E}_p(\emptyset)$ is the quantity measured in [1,2,6].

In many applications, data comes in a streaming fashion. When it comes to very large data volume, it is important to maintain a data structure that uses small memory and estimates different statistics about the data accurately at the same time. As the space requirement to measure the accuracy of Bayesian networks is as large as $O(k^n)$ and as the size of our data set m increases, our problem of evaluating Bayesian networks via data streams with small memory is of considerable importance.

1.1 Our Results

Here, and henceforth we use k, n, d and m to denote the range of the variables, the number of the variables, the maximum in-degree of the network and the length of the stream respectively.

1. *Testing and Estimating ℓ_p Accuracy.* For any Bayesian network G, we present a single-pass algorithm using $\tilde{\Theta}(nk^d)$ space[1] for the problem of testing whether the data is consistent with G, i.e., $\mathcal{E}_p(G) = 0$. We prove a matching lower bound showing that the dependence on n, k, and d is optimal. We also present a $\tilde{O}(\varepsilon^{-2}nk^{d+1})$-space algorithm for estimating $\mathcal{E}_p(G)$ up to a $(1+\varepsilon)$ factor. The lower bound is based on the Local Markov Property, a result from Bayesian Networks literature, and a reduction from communication complexity.
2. *Estimating Log-Likelihood.* Next, we present a single-pass $\tilde{O}(nk^d)$-space algorithm that estimates the log-likelihood of a given network. We also prove a lower bound of $\Omega(k^d)$ for any factor 2 approximation of this quantity. As an application, we can find the branching tree network that approximately maximizes the log-likelihood of the observed streaming data in space $\tilde{O}(n^2k)$ with $O(n^2)$ post-processing time. Our algorithm is based on the Chow-Liu tree [4] construction.

[1] $\tilde{O}$ omits all poly-logarithmic factors of m, n, and k.

3. *Trade-offs for Independence Testing.* We revisit the problem of independence testing in Section 5 and present space/accuracy trade-offs for estimating $\mathcal{E}_p(\emptyset)$. Specifically, for $p = 1$, we can achieve an $(n-1)/t$-approximation for any constant $1 \leq t < n/2$ using $O(\text{poly}\, n)$ space compared to the $(1 \pm \varepsilon)$-approximation algorithm in [2] with space that is doubly-exponential in n. For $p = 2$, we present an $O(\text{poly}\, n)$-space algorithm with additive error compared to the $O(3^n)$-space algorithm in [1] with multiplicative error.

1.2 Notation

$A \perp B \mid C$ denotes the assertion that random variables A, B are independent conditioned on C, i.e., $\mathbb{P}(A = a, B = b|C = c) = \mathbb{P}(A = a|C = c)\mathbb{P}(B = b|C = c)$ for all a, b, c in the range of A, B, C. $\text{Pa}(X_i)$ denotes the set of variables that are parents of X_i and $\text{ND}(X_i)$ denotes the set of variables that are non-descendants of X_i, other than $\text{Pa}(X_i)$. If $X_1, \ldots, X_n \in [k]$ then we use $(X_1, \ldots, X_n)$ denote a tuple of n variables in $[k]^n$ or equivalently a single variable in the range $[k^n]$.

2 Algorithms for Estimating $\mathcal{E}_p(G)$

In this section, we present approximation algorithms for estimating $\mathcal{E}_p(G)$ for an arbitrary Bayesian network G and a more efficient algorithm just to test if $\mathcal{E}_p(G) = 0$.

2.1 $(1 + \varepsilon)$-Approximation Using $\tilde{O}(nk^{d+1})$ Space

We first note that the factorized distribution $\mathcal{P}_G$ can be computed and stored exactly in $O(nk^{d+1} \log m)$ bits since, by Eq. (1) and Eq. (3), it suffices to compute

$$\frac{\sum_{\mathbf{a} \in [k]^n \,:\, a_j = x_j \ \forall j \ s.t \ X_j \in \{X_i\} \cup \text{Pa}(X_i)} f(\mathbf{a})}{\sum_{\mathbf{a} \in [k]^n \,:\, a_j = x_j \ \forall j \ s.t \ X_j \in \text{Pa}(X_i)} f(\mathbf{a})} \,.$$

for each $i \in [n]$ and each of at most k^{d+1} combinations of values for X_i and $\text{Pa}(X_i)$. Given this observation, it is straightforward to approximate $\mathcal{E}_p(G)$ given any data stream "sketch" algorithm that returns a $(1 + \varepsilon)$ estimate for the ℓ_p norm of a vector v. Kane et al. [10] presented such as algorithm that uses space that is logarithmic in the dimension of the vector.

Specifically, we apply the algorithm on a vector v defined as follows. Consider v to be indexed as $[k] \times [k] \times \ldots \times [k]$. On the arrival of tuple $(x_1, \ldots, x_n)$, we increment the coordinate corresponding to $(x_1, \ldots, x_n)$ by $1/m$. At the end of the stream, v encodes the empirical joint distribution. For each $(x_1, \ldots, x_n)$, we now decrement the corresponding coordinate by $\mathcal{P}_G(x_1, \ldots, x_n)$. At this point, $v_{x_1,\ldots,x_n} = \mathcal{P}(x_1, \ldots, x_n) - \mathcal{P}_G(x_1, \ldots, x_n)$ and hence the ℓ_p norm of v is $\mathcal{E}_p(G)$. Hence, returning the estimate from the algorithm yields a $1 + \varepsilon$ approximation to $\mathcal{E}_p(G)$ as required.

Note that this simple approach also improves over existing work [2] on the case of measuring $\ell_p(G)$ when G has no edges (i.e., measuring how far the data is from independent) unless n is very small compared to k. The space used in previous work is doubly-exponential in n but logarithmic in k whereas our approach uses $\tilde{O}(nk)$ space and hence, our approach is more space-efficient unless $k > 2^{n^n}/n$.

Theorem 1. *There exists a single-pass algorithm that computes* $(1 \pm \varepsilon)\,\mathcal{E}_p(G)$ *with probability at least* $1-\delta$ *using* $\tilde{O}(\varepsilon^{-2}k^{d+1}n\log\delta^{-1})$ *space.*

2.2 $2n$-Approximation Using $\tilde{O}(\mathrm{poly}(n)k^d)$ Space

We now give an alternative algorithm with a weaker approximation guarantee but requires a smaller space in terms of k and d.

Theorem 2. *There exists a single-pass* $\tilde{O}(\mathrm{poly}(n)\cdot k^d)$*-space algorithm that computes an* $O(n)$*-approximation of* $\mathcal{E}_1(G)$ *with probability at least* $1-\delta$.

We first briefly describe the algorithm. Without loss of generality, assume X_n, X_{n-1}, ..., X_1 form a topological order in G. Such an order must always exist since G is acyclic. Let $\mathbf{X}(i,n)$ denote $(X_i,\ldots,X_n)$.

1. For each $i \in [n-1]$, compute a $(1+\varepsilon)$-factor approximation of
$$v_i := \left|\mathbb{P}(\mathbf{X}(i,n)) - \mathbb{P}(X_i|\,\mathrm{Pa}(X_i))\mathbb{P}(\mathbf{X}(i+1,n))\right|.$$
We shall explain how to get the approximation shortly.
2. Return the sum of the estimators above.

Proof. We start by showing that $\mathcal{E}_1(G) \le \sum_{i=1}^{n-1} v_i \le 2n\,\mathcal{E}_1(G)$. The first inequality is derived as follows.

$$\begin{aligned}\mathcal{E}_1(G) \le{}& \left|\mathbb{P}(\mathbf{X}) - \mathbb{P}(X_1|\,\mathrm{Pa}(X_1))\mathbb{P}(\mathbf{X}(2,n))\right| \\ &+ \left|\mathbb{P}(X_1|\,\mathrm{Pa}(X_1))\mathbb{P}(\mathbf{X}(2,n)) - \mathbb{P}(X_1|\,\mathrm{Pa}(X_1))\mathbb{P}(X_2|\,\mathrm{Pa}(X_2))\mathbb{P}(\mathbf{X}(3,n))\right| \\ &+ \ldots + \left|\prod_{i=1}^{n-2}\left(\mathbb{P}(X_i|\,\mathrm{Pa}(X_i)\right)\mathbb{P}(X_{n-1},X_n) - \prod_{i=1}^{n}\mathbb{P}(X_i|\,\mathrm{Pa}(X_i))\right| \qquad (4)\\ ={}& \sum_{i=1}^{n-1}\left|\mathbb{P}(\mathbf{X}(i,n)) - \mathbb{P}(X_i|\,\mathrm{Pa}(X_i))\mathbb{P}(\mathbf{X}(i+1,n))\right| = \sum_{i=1}^{n-1} v_i.\end{aligned}$$

The first inequality follows from the triangle-inequality. For each ith term in Equation (4), we can factor out $\{\mathbb{P}(X_j|\,\mathrm{Pa}(X_j))\}_{j\in[i-1]}$ which sums (over X_j) to 1 as the inner factors do not involve X_j.

Next, we show that $v_i \le 2\,\mathcal{E}_1(G)$. By the triangle-equality we have:

$$\begin{aligned}v_i \le{}& \left|\mathbb{P}(\mathbf{X}(i,n)) - \prod_{j\ge i}\mathbb{P}(X_j|\,\mathrm{Pa}(X_j))\right| \\ &+ \left|\prod_{j\ge i}\mathbb{P}(X_j|\,\mathrm{Pa}(X_j)) - \mathbb{P}(X_i|\,\mathrm{Pa}(X_i))\mathbb{P}(\mathbf{X}(i+1,n)\right|.\end{aligned}$$

To bound each term on the right, we first introduce the following notation:

$$g_k(\mathbf{x}) = \mathbb{P}(X_k = \mathbf{x}_k | X_j = \mathbf{x}_j \text{ for all } X_j \in \mathrm{Pa}(X_k))$$

Then,

$$\begin{aligned}
&\left|\mathbb{P}(\mathbf{X}(i,n)) - \prod_{j\geq i}\mathbb{P}(X_j |\, \mathrm{Pa}(X_j))\right| \\
&= \sum_{\mathbf{b}\in[k]^{n-i+1}}\left|\sum_{\mathbf{a}\in[k]^{i-1}}\mathbb{P}(\mathbf{X}(1,i-1)=\mathbf{a},\mathbf{X}(i,n)=\mathbf{b}) - \left(\sum_{\mathbf{a}\in[k]^{i-1}}\prod_{1\leq q<i} g_q(\mathbf{ab})\right)\cdot\prod_{j\geq i} g_j(\mathbf{ab})\right)\right| \\
&\leq \sum_{\substack{\mathbf{a}\in[k]^{i-1}\\ \mathbf{b}\in[k]^{n-i+1}}}\left|\mathbb{P}(\mathbf{X}=\mathbf{ab}) - \prod_{1\leq q<i} g_q(\mathbf{ab})\cdot\prod_{j\geq i} g_j(\mathbf{ab})\right| = \mathcal{E}_1(G).
\end{aligned}$$

The second term is equal to $|\prod_{j\geq i+1}\mathbb{P}(X_j|\,\mathrm{Pa}(X_j)) - \mathbb{P}(\mathbf{X}(i+1,n)|$ which is of similar form as the first term and can be upper bounded by $\mathcal{E}_1(G)$ similarly. We approximate each v_i as follows. For each $c \in [k^d]$, we have:

$$\begin{aligned}
v_i(c) = \mathbb{P}(\mathrm{Pa}(X_i)=c)\Big|&\mathbb{P}(\mathbf{X}(i,n)|\mathrm{Pa}(X_i)=c) \\
&- \mathbb{P}(X_i|\,\mathrm{Pa}(X_i)=c)\mathbb{P}(\mathbf{X}(i+1,n)|\,\mathrm{Pa}(X_i)=c)\Big|.
\end{aligned}$$

We can compute $\mathbb{P}(\mathrm{Pa}(X_i)=c)$ exactly and approximate $v_i(c)$ using Theorem 9. Because $v_i = \sum_{c\in[k^d]} v_i(c)$, to get an estimate for v_i, we simply take the sum of the estimates for each $v_i(c)$. Since we need to do this for all $i\in[n], c\in[k^d]$, the space usage is $\tilde{O}(\mathrm{poly}(n)\cdot k^d)$.

2.3 Decision Problem

We now show that testing $\mathcal{E}_p(G)=0$ can indeed be done in space that is tight with the lower bound in terms of n, k, d.

Definition 1. *A Bayesian network G with vertices $X_1, .., X_n$ satisfies the* Local Markov Property *if $X_i \perp \mathrm{ND}(X_i) \mid \mathrm{Pa}(X_i)$ for all $i\in[n]$.*

We rely on the following theorem. Its proof can be found in many Bayesian networks literature such as [8].

Theorem 3. *(Local Markov Property) Any given Bayesian network G satisfies $\mathcal{E}_p(G)=0$ iff it satisfies the Local Markov Property.*

The idea is to check the Local Markov Property for each variable in the network. However, to match the lower bound, we also need to resolve a subtle issue regarding storing the random vectors.

Theorem 4. *There exists an $\tilde{O}(k^d n)$-space single-pass algorithm that tests $\mathcal{E}_p(G)=0$ with probability at least $1-\delta$.*

Proof. For each X_i, because $|\operatorname{ND}(X_i)| \leq n$, $\operatorname{ND}(X_i)$ can be viewed as a single variable that takes at most k^n different values. We need to check if:

$$\eta(i) := \left\| \mathbb{P}(X_i, \operatorname{ND}(X_i) | \operatorname{Pa}(X_i)) - \mathbb{P}(X_i | \operatorname{Pa}(X_i)) \mathbb{P}(\operatorname{ND}(X_i) | \operatorname{Pa}(X_i)) \right\|_2 = 0.$$

Call this testing algorithm $\mathcal{A}_i$. We define $\eta(i,c)$ to be the distance above with $\operatorname{Pa}(X_i) = c$. We have $\eta(i) = \sum_{c \in [k]^{|\operatorname{Pa}(X_i)|}} \eta(i,c)$. For any fixed c, we can test $\eta(i,c) = 0$ by running the algorithm from Theorem 9.

For some $S \subseteq \{X_1, \ldots, X_n\}$ and $\mathbf{x} \in [k]^n$, let x_S denote the tuple of $\{x_j : X_j \in S\}$. The algorithm in Theorem 9 incrementally maintains the following sketches:

$$t_1 = \sum_{a \in [k], b \in [k]^{|\operatorname{ND}(X_i)|}, \mathbf{x}: x_i = a, x_{\operatorname{ND}(X_i)} = b, x_{\operatorname{Pa}(X_i)} = c} f(\mathbf{x}) \gamma_a \lambda_b$$

$$t_2 = \sum_{a \in [k], \mathbf{x}: x_i = a, x_{\operatorname{Pa}(X_i)} = c} f(\mathbf{x}) \gamma_a \quad \text{and} \quad t_3 = \sum_{b \in [k]^{|\operatorname{ND}(X_i)|}, \mathbf{x}: x_{\operatorname{ND}(X_i)} = b, x_{\operatorname{Pa}(X_i)} = c} f(\mathbf{x}) \lambda_b$$

where $\lambda, \gamma \in \{-1, 1\}^{k^n}$ are 4-wise independent vectors. The space required to store these vectors is $O(\log k^n) = O(n \log k)$. It can be shown that [1,6]:

$$\mathbb{E}[(\frac{t_1}{m} - \frac{t_2 t_3}{m^2})^2] = \eta(i,c)^2 \text{ and } \mathbb{V}ar[(\frac{t_1}{m} - \frac{t_2 t_3}{m^2})^2] \leq 9\eta(i,c)^4.$$

Hence, to have a factor 10 approximation of the distance that tests if $\eta(i,c) = 0$ with probability at least $1 - \delta/(k^d n)$, we need to use $O(\log k^d + \log \delta^{-1} + \log n)$ independent λ, γ's in parallel and take the median of the estimators. We need to do this for all $c \in [k]^{|\operatorname{Pa}(X_i)|}$. Run $\mathcal{A}_i$ for all $i \in [n]$. *The key observation* is that all $\mathcal{A}_i$'s may use the same set of these 4-wise independent vectors. So the total space to run $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is:

$$O(\underbrace{nk^d(\log k^d + \log \delta^{-1} + \log n) \log m}_{\text{space to store the sketches}} + \underbrace{k^d(\log k^d + \log \delta^{-1} + \log n) n \log k}_{\text{space to store the random vectors}}) = \tilde{O}(nk^d).$$

By the union bound, we can tell if there is an X_i that does not satisfy the local Markov property with probability at least $1 - \delta$ in the space that is optimal up to a polylogarithmic factor.

3 Lower Bounds for Estimating $\mathcal{E}_p(G)$

Next, we show that the decision algorithm and the approximation algorithm above are optimal and near-optimal respectively. It has been shown that independence testing via ℓ_p distance can be done in $O(\operatorname{polylog} k)$ space. The open question we are trying to answer is whether it is still possible to test more general dependencies in $O(\operatorname{polylog} k)$ space. Unfortunately, the answer is, in general, no. We first prove that for testing whether two variables are perfectly independent given the third variable, any constant-pass streaming algorithm requires $\Omega(k)$ space.

The proofs of our lower bounds use the standard technique of reducing from a communication complexity problem. In particular, we consider the disjointness problem where Alice and Bob each have a string $x \in \{1,2\}^k$ and $y \in \{1,2\}^k$ respectively and want evaluate $\mathrm{DISJ}(x,y)$ where

$$\mathrm{DISJ}(x,y) = \begin{cases} 0 & \text{if there exists } i \text{ such that } x_i = y_i = 1 \\ 1 & \text{otherwise} \end{cases}$$

A classic result [9] shows that any (randomized) protocol with constant number of rounds for this problem requires $\Omega(k)$ bits to be communicated. The following remark is useful in our reduction.

Lemma 1. *Given a stream of two binary samples in the format (A,B) as $(a,2),(2,b)$. Then, A,B are independent iff a,b are not both equal 1.*

Proof. If $a = b = 1$, then $\mathbb{P}(A=1,B=2) = 0.5 \neq \mathbb{P}(A=1)\mathbb{P}(B=2) = 0.5 \times 0.5 = 0.25$. Otherwise, one can easily check that $\mathbb{P}(A,B) = \mathbb{P}(A)\mathbb{P}(B)$.

Proposition 1. *There exists a network G such that any constant-pass algorithm that decides if $\mathcal{E}_p(G) = 0$ with probability at least 2/3 requires $\Omega(k^d)$ space.*

Proof. Consider the Bayesian network G with vertices $X_1,\ldots,X_d,Y,Z$ where each X_i is a parent of $X_1,X_2,\ldots,X_{i-1},Y$, and Z. Let $\mathbf{X} = (X_1,\ldots,X_d)$. Then,

$$\begin{aligned}\mathcal{E}_p(G) &= \left\| \mathbb{P}(Y,Z|\mathbf{X})\mathbb{P}(\mathbf{X}) - \mathbb{P}(Y|\mathbf{X})\mathbb{P}(Z|\mathbf{X})\big(\prod_{i=1}^{d}\mathbb{P}(X_i|X_{i+1},\ldots,X_d)\big)\right\|_p \\ &= \left\| \mathbb{P}(Y,Z|\mathbf{X})\mathbb{P}(\mathbf{X}) - \mathbb{P}(Y|\mathbf{X})\mathbb{P}(Z|\mathbf{X})\mathbb{P}(\mathbf{X})\right\|_p . \qquad (5)\end{aligned}$$

We make the reduction from DISJ where Alice and Bob, with bit strings a and b of length k^d, generate the stream S_A and S_B of $(Y,Z,\mathbf{X})$-tuples respectively:

$$S_A = \{(a_1, 2, \mathbf{x}) : \mathbf{x} \in [k]^d\} \quad , \quad S_B = \{(2, b_1, \mathbf{x}) : \mathbf{x} \in [k]^d\} \ .$$

By Equation (5), we have that $\mathcal{E}_p(G) = 0$ iff $Y \perp Z | \{\mathbf{X} = c\}$ for all $c \in [k]^d$. By Lemma 1, this is satisfied iff $\mathrm{DISJ}(a,b) = 1$. Therefore, any constant-pass algorithm that decides if $\mathcal{E}_p(G) = 0$ requires $\Omega(k^d)$ space.

We now construct a more sophisticated reduction to incorporate n in the lower bound.

Theorem 5. *There exists a Bayesian network G such that any constant-pass algorithm that determines if $\mathcal{E}_p(G) = 0$ with probability at least 2/3 requires $\Omega(nk^d)$ space.*

Proof. Without loss of generality, assume n is a power of 2. Let $x \in \{1,2\}^{nk}, y \in \{1,2\}^{nk}$ be an instance of DISJ where it be convenient to index x and y by $[n] \times [k]$. The Bayesian network we consider is balanced binary tree with leaves A_1, B_1, A_2, $B_2,\ldots,A_n$, B_n and internal nodes R_i^j where R_i^1 in the parent of

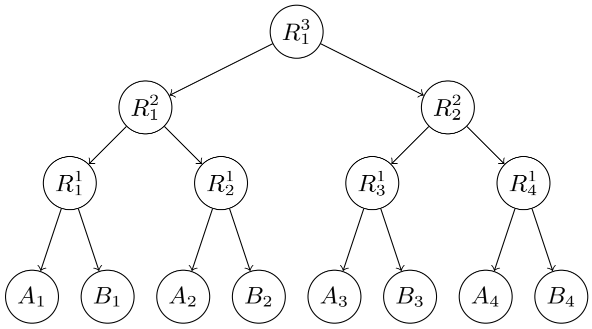


Fig. 1. Construction for $n = 4$

A_i and B_i and R_i^j is the parent of R_{2i-1}^{j-1} and R_{2i}^{j-1} for $j > 1$. The root node is $R_1^{\log n+1}$. See Figure 1. The variables R_i^j will take $3k$ different values and it will be convenient to index these values as $[3] \times [k]$. The leaf variables take either the value 1 or 2.

Alice generates a stream that defines samples from the joint distribution based on x. Each sample generated satisfies the following criteria and all distinct samples that obey this criteria are generated:

1. $R_1^{\log n+1} \in \{(1, z), (2, z) : z \in [k]\}$.
2. If $R_i^j = (1, z)$ for $j > 1$:
   – The left child $R_{2i-1}^{j-1} \in \{(1, z), (2, z)\}$ and the right child $R_{2i}^{j-1} = (3, z)$.
3. If $R_i^j = (2, z)$ for $j > 1$:
   – The left child $R_{2i-1}^{j-1} = (3, z)$ and the right child $R_{2i}^{j-1} \in \{(1, z), (2, z)\}$.
4. If $R_i^j = (3, z)$ for $j > 1$:
   – Both the values for the children R_{2i-1}^{j-1} and R_{2i}^{j-1} are $(3, z)$.
5. If $R_i^1 \in \{(1, z), (2, z)\}$:
   – The values for the children are $A_i = x_{i,z}$, $B_i = 2$
6. If $R_i^1 = (3, z)$:
   – The values for the children are $A_i = 2$, $B_i = 2$

Bob then generates a series samples in a similar manner except that Rule 5 becomes: If $R_i^1 \in \{(1, z), (2, z)\}$, then $A_i = 2$, $B_i = y_{i,z}$.

Note that each sample defined by either Alice or Bob specifies a path from the root to a pair A_i, B_i as following: Starting from the root, if the current node's value is equal to $(1, z)$, then go to its left child; on the other hand, if its value is equal to $(2, z)$, then go to the right child. Once we commit to a direction, every descendant on the other direction is set to $(3, z)$ for the R nodes and 2 for the A and B nodes.

First assume that $\mathrm{DISJ}(x, y) = 0$. Then $x_{i,z} = y_{i,z} = 1$ for some $z \in [k], i \in [n]$. By Lemma 1 we infer that A_i and B_i are not independent conditioned on either $R_i^1 = (1, z)$ or $R_i^1 = (2, z)$ and hence, $\mathcal{E}_p(G) \neq 0$.

Conversely, assume that $\text{DISJ}(x, y) = 1$. The Local Markov Property says that if every vertex is independent of its non-descendants given its parents then $\mathcal{E}_p(G) = 0$.

- First we show that it is true for any R_i^j variable. Conditioned on the parent of R_i^j taking the value $(3, z)$, R_i^j is constant and hence independent of non-descendants. Conditioned on the parent of R_i^j taking the value $(1, z)$ or $(2, z)$, the values of the non-descendants of R_i^j are fixed and hence independent of R_i^j.
- Next, we show that it is true for any A_i variable. The argument for B_i is identical. Conditioned on $R_i^1 = (3, z)$, then A_i is constant and hence independent of all non-descendants. If $R_i^1 = (1, z)$ or $R_i^1 = (2, z)$, the values of all non-descendants, except possibly B_i, are fixed. But by Lemma 1, B_i is independent of A_i conditioned on R_i^1 since $\text{DISJ}(x, y) = 1$.

Hence, $\text{DISJ}(x, y) = 1$ iff $\mathcal{E}_p(G) = 0$ and therefore testing if $\mathcal{E}_p(G) = 0$ requires $\Omega(nk)$ space.

To extend the lower bound to $\Omega(nk^d)$ consider an instance of DISJ of length nk^d. Let the variables in G be children of all $d-1$ new variables $D_1, \ldots, D_{d-1}$ where there is a directed edge between $D_i \to D_j$ for $i > j$. Call the new network G'. Similar to the proof of Proposition 1, to solve DISJ on the wth pair of bit strings of length nk where $w \in [k^{d-1}]$, Alice and Bob generate samples with variables in G as described above and set $(D_1, \ldots, D_{d-1}) = w$. Hence, any streaming algorithm that decides if $\mathcal{E}_p(G') = 0$ requires $\Omega(nk^d)$ space.

4 Log-Likelihood and Approximate Chow-Liu Trees

While it is natural to test the networks using ℓ_1 or ℓ_2 distance, it is more convenient to use the log-likelihood to learn the structure of certain types of Bayesian networks. Let $\mathbf{x}^{(j)}$ be the jth sample in the stream. The log-likelihood of G given the data stream is:

$$\mathcal{L}(D, G) = \frac{1}{m}\sum_{i=1}^{m} \log \mathcal{P}_G(\mathbf{x}^{(i)}) = -\sum_{j=1}^{n} H(X_j \,|\, \text{Pa}(X_j))$$

By using the entropy estimation algorithm of Chakrabarti et al. [3] to estimate the conditional entropies $H(X_j \,|\, \text{Pa}(X_j))$ for each of the $O(k^d)$ possible values of $\text{Pa}(X_j)$, we can approximate $\mathcal{L}(D, G)$ up to a factor $1 + \varepsilon$.

Theorem 6. *There is a single-pass algorithm that returns a $(1+\varepsilon)$ approximation of $\mathcal{L}(D, G)$ for a given Bayesian network G w.h.p using $\tilde{O}(\varepsilon^{-2} nk^d)$ space.*

We prove that the above algorithm is tight in terms of k and d.

Theorem 7. *There exists a Bayesian network G with such that any single-pass streaming algorithm that outputs a 2-approximation of $\mathcal{L}(D, G)$ requires $\Omega(k^d)$ space.*

Proof. Let $t = 10dk^d \log k$. Consider the network with nodes $\{X_i\}_{i\in[t]}$ that are all children of $\{Y_i\}_{i\in[d]}$. Let $\mathbf{Y} = (Y_1, \ldots, Y_d)$. Then,

$$\mathcal{L}(D, G) = -\sum_{i=1}^{t} H(X_i|\mathbf{Y}) - \sum_{i=1}^{d} H(Y_i).$$

Using ideas from [6], we make the following reduction. Given an instance of DISJ with bit strings a, b of length k^d where we may assume $|\{i : a_i = 1\}| = |\{i : b_i = 1\}| = k^d/4$. If Alice and Bob generate samples from the joint distribution $(X_1, \ldots, X_t, \mathbf{Y})$:

$$S_A = \{(1, \ldots, 1, i) : a_i = 1\}, \qquad S_B = \{(2, \ldots, 2, i) : b_i = 1\} .$$

where $i \in [k^d]$ specifies the values for $\mathbf{Y}$. If $\mathrm{DISJ}(a, b) = 1$ then $H(X_i|\mathbf{Y}) = 0$ and furthermore $\sum_{i=1}^{t} H(X_i|\mathbf{Y}) = 0$. If $\mathrm{DISJ}(a, b) = 1$ then $H(X_i|\mathbf{Y}) \geq 4/k^d$ and hence, $\sum_{i=1}^{t} H(X_i|\mathbf{Y}) \geq 40d \log k$. Because $\sum_{i=1}^{d} H(Y_i) \leq d \log k$, a 2-approximation of $\mathcal{L}(D, G)$ distinguishes $\sum_{i=1}^{t} H(X_i|\mathbf{Y}) = 0$ from $\sum_{i=1}^{t} H(X_i|\mathbf{Y}) \geq 40d \log k$.

The famous Chow-Liu tree [4], T_{CL}, is the tree with $d = 1$ that maximizes the log-likelihood. Chow-Liu tree is particularly important as it is the only known closed form structural learning algorithm that is polynomial time. We show that there is a single-pass algorithm that approximates T_{CL}.

Theorem 8. *There is a single-pass algorithm that outputs a rooted tree T such that $\mathcal{L}(D, T) \geq (1 - \varepsilon)\mathcal{L}(D, T_{CL})$ with probability at least $1 - \delta$ in $\tilde{O}(n^2 k \varepsilon^{-2} \log(\delta^{-1}))$ space. The post processing time is $O(n^2)$.*

5 Space-Accuracy Trade-offs in Independence Testing

From previous work on independence testing [1,2,6], we may assume:

Theorem 9. *There exist single-pass algorithms that computes a $(1 \pm \varepsilon)$-approximation of $\mathcal{E}_p(\emptyset)$ with probability at least $1 - \delta$ and uses*

1. *$O((\varepsilon^{-1} \log(mk\delta^{-1}))^{O(n)^n})$ space for $p = 1$*
2. *$O(3^n \varepsilon^{-2} (\log k + \log m) \log \delta^{-1})$ space for $p = 2$.*

By simply appealing to Theorem 2, we have an interesting trade-off between the space usage and the approximation accuracy when testing n-wise independence using ℓ_1 distance. Specifically, we can have an $O(n)$-approximation of $\mathcal{E}_1(\emptyset)$ but using only $O(\mathrm{poly}(n)\,\mathrm{polylog}(k))$ space compared to the space of doubly-exponential in n in Theorem 9.

Proposition 2. *There is a single-pass algorithm that outputs a $O(n)$-approximation of $\mathcal{E}_1(\emptyset)$ using $O(\mathrm{poly}(n, \varepsilon^{-1}) \cdot \mathrm{polylog}(m, k, \delta^{-1}))$ space.*

We can even achieve a stronger approximation guarantee:

Theorem 10. *For any constant $1 \leq t < n/2$, there is a single-pass algorithm that outputs a $(1 \pm \varepsilon)(n-1)/t$-approximation for $\mathcal{E}_1(\emptyset)$ using $\tilde{O}(\text{poly}(n, \varepsilon^{-1}))$ space.*

We can approximate $\mathcal{E}_2(\emptyset)$ as stated in Theorem 9 up to a factor $(1 \pm \varepsilon)$ using $O(3^n)$ space. However, if we allow the error to be additive, we only need $O(n)$ space.

Theorem 11. *There exists an $O(n^3 \varepsilon^{-2} \log(mk) \log \delta^{-1})$-space single-pass algorithm that outputs $\mathcal{E}_2(\emptyset) \pm \varepsilon$ with probability at least $1-\delta$.*

Proof. The main idea is to rewrite $\mathcal{E}_2(\emptyset)$ as follows:

$$\mathcal{E}_2(\emptyset) = \sum_{\mathbf{x} \in [k]^n} \mathbb{P}(\mathbf{X} = \mathbf{x})^2 + \prod_{i=1}^{n} \sum_{x_i \in [k]} \mathbb{P}(X_i = x_i)^2 - 2 \sum_{\mathbf{x} \in [k]^n} \mathbb{P}(\mathbf{X} = \mathbf{x}) \prod_{i=1}^{n} \mathbb{P}(X_i = x_i).$$

It is possible to estimate the values of $\sum_{x_i \in [k]} \mathbb{P}(X_i = x_i)^2$ for all $i \in [n]$ and $\sum_{\mathbf{x} \in [k]^n} \mathbb{P}(\mathbf{X} = \mathbf{x})^2$ up to a multiplicative factor of $(1+\varepsilon/n)$ in $O(n^3 \varepsilon^{-2} \log(km + \delta^{-1}))$ space using an existing algorithm for estimating the second frequency moment [10]. This implies a $(1+\varepsilon)$ multiplicative approximation for the first two terms. However, since $\sum_{\mathbf{x} \in [k]^n} \mathbb{P}(\mathbf{X} = \mathbf{x})^2 \leq 1$ and $\prod_{i=1}^{n} \sum_{x_i \in [k]} \mathbb{P}(X_i = x_i)^2 \leq 1$ this implies an additive 2ε approximation to the first two terms.

It remains to show we can approximate $\sum_{\mathbf{x} \in [k]^n} \mathbb{P}(\mathbf{X} = \mathbf{x}) \prod_{i=1}^{n} \mathbb{P}(X_i = x_i)$ in small space. To argue this let

$$H = \{(x_1, \ldots, x_n) \in [k]^n : \mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) \geq \varepsilon)\} .$$

We will show that it is possible to construct a set H' such that $H \subseteq H'$ and for all $(x_1, \ldots, x_n) \in H'$, we may estimate $\mathbb{P}(X_i = x_i)$ and $\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n)$ up to a factor $(1+\varepsilon)$.

To do this we use the Count-Min sketch [5] which has the following properties:

Claim. There exists a $O(\varepsilon^{-2} \log \delta^{-1} (\log m + \log t))$-space streaming algorithm that, when run on any stream of length m defining a frequency vector y of length t, returns a set of indices and estimates $C = \{(i, \tilde{y}_i) : y_i \leq \tilde{y}_i \leq (1+\varepsilon) y_i\}$ such that $(i, \tilde{y}_i) \in C$ for all $y_i \geq \varepsilon |y|$. We call $S = \{i : (i, \tilde{y}_i) \in C\}$ the ε-cover of y.

In our case y will be a pmf vector, i.e., the frequency vector normalized by dividing each coordinate by m and hence $|y| = 1$. Thus we can find an ε-cover S of the joint pmf and an ε-cover S_i of the marginal pmf of each variable X_i. Let

$$H' = \{(x_1, \ldots, x_n) \in S : x_i \in S_i \text{ for all } i \in [n]\} .$$

Note that if $\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) \geq \varepsilon$ then $\mathbb{P}(X_1 = x_1) \geq \varepsilon, \ldots, \mathbb{P}(X_n = x_n) \geq \varepsilon$. Therefore, the ε-covers constructed using the Count-Min sketch give a multiplicative estimate $\sum_{\mathbf{x} \in H'} \mathbb{P}(\mathbf{X} = \mathbf{x}) \prod_{i=1}^{n} \mathbb{P}(X_i = x_i)$. Furthermore,

$$\begin{aligned} \sum_{\mathbf{x} \notin H'} \mathbb{P}(\mathbf{X} = \mathbf{x}) \prod_{i=1}^{n} \mathbb{P}(X_i = x_i) &\leq \sum_{\mathbf{x} : \mathbb{P}(\mathbf{X} = \mathbf{x}) < \varepsilon} \mathbb{P}(\mathbf{X} = \mathbf{x}) \mathbb{P}(X_1 = x_n) \\ &\leq \varepsilon \sum_{x_1 \in [k]} \mathbb{P}(X_1 = x_1) = \varepsilon. \end{aligned}$$

and therefore the total additive error in our estimate of $\mathcal{E}_2(\emptyset)$ is $O(\varepsilon)$.

References

1. Braverman, V., Chung, K.M., Liu, Z., Mitzenmacher, M., Ostrovsky, R.: Ams without 4-wise independence on product domains. In: STACS, pp. 119–130 (2010)
2. Braverman, V., Ostrovsky, R.: Measuring independence of datasets. In: STOC, pp. 271–280 (2010)
3. Chakrabarti, A., Cormode, G., McGregor, A.: A near-optimal algorithm for estimating the entropy of a stream. ACM Transactions on Algorithms 6(3) (2010)
4. Chow, C., Liu, C.: Approximating discrete probability distributions with dependence trees. IEEE Trans. Inf. Theor. **14**(3), 462–467 (2006). http://dx.doi.org/10.1109/TIT.1968.1054142
5. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms **55**(1), 58–75 (2005)
6. Indyk, P., McGregor, A.: Declaring independence via the sketching of sketches. In: SODA, pp. 737–745 (2008)
7. Jensen, F.V., Nielsen, T.D.: Bayesian networks and decision graphs. Springer (2007)
8. Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods for graphical models. Springer (1998)
9. Kalyanasundaram, B., Schintger, G.: The probabilistic communication complexity of set intersection. SIAM Journal on Discrete Mathematics **5**(4), 545–557 (1992)
10. Kane, D.M., Nelson, J., Porat, E., Woodruff, D.P.: Fast moment estimation in data streams in optimal space. In: STOC, pp. 745–754 (2011)
11. Pappas, A., Gillies, D.F.: A New Measure for the Accuracy of a Bayesian Network. In: Coello Coello, C.A., de Albornoz, Á., Sucar, L.E., Battistutti, O.C. (eds.) MICAI 2002. LNCS (LNAI), vol. 2313, pp. 411–419. Springer, Heidelberg (2002)

Algorithm

On Energy-Efficient Computations with Advice

Hans-Joachim Böckenhauer[1](✉), Richard Dobson[2], Sacha Krug[1], and Kathleen Steinhöfel[2]

[1] Department of Computer Science, ETH Zürich, Zürich, Switzerland
{hjb,sacha.krug}@inf.ethz.ch

[2] Department of Informatics, King's College London, London, UK
{richard.dobson,kathleen.steinhofel}@kcl.ac.uk

Abstract. Online problems have always played an important role in computer science. Here, not the whole input is known at the beginning, but it is only revealed gradually. These problems frequently occur in practice, and therefore the performance of algorithms for such problems is of great theoretical and practical interest. One such online problem is energy management in electronic devices, e. g., smartphones. As such a device is usually not being used permanently, it is reasonable to change to a lower-energy state (like hibernation) after a certain idle time. Resuming from hibernation, however, also needs a certain amount of energy; therefore, hibernation should only happen if the idle period is long.

Advice complexity is a recent approach for measuring the information content of an online problem, i. e., the amount of knowledge about the future parts of the input that is necessary to compute a high-quality solution. The approach allows for a more fine-grained analysis of the hardness of online problems than the classical competitive analysis.

We analyze the advice complexity of this problem. For systems with two states, we construct an online algorithm with advice that is 1.8-competitive with one advice bit and 1.6-competitive with five advice bits, whereas every deterministic algorithm without advice is known to be no better than 2-competitive. Moreover, the algorithm's competitive ratio converges fast towards $\mathrm{e}/(\mathrm{e}-1)$ with an increasing number of advice bits. For competitive ratios in the range $[1, \mathrm{e}/(\mathrm{e}-1)]$, we present two complementary algorithms: one behaves optimally on a certain prefix, and the other falls asleep on the longest phases. Conversely, we show that every algorithm with a competitive ratio less than $1 + 1/(4w + 2)$, where w is the wake-up energy, needs to read a linear number of advice bits.

1 Introduction

When a computer is turned on, it consumes energy. Even if the system is idle, it will continue to consume energy despite not being actively used. Many modern devices are equipped with several low-power sleep states, which can reduce the amount of energy consumed when the system is idle. One of the most common

This work was partially supported by SNF grant 200021-141089, EPSRC grant EP/H501312/1, and LMS grant SC7-1314-08.

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 747–758, 2015.
DOI: 10.1007/978-3-319-21398-9_58

low-power states is to dim or turn off the screen of the device; this has been shown to significantly reduce power consumption in smart phones [10].

Sleep state management is a fundamental energy efficiency problem; it is considered to be of great importance as it has the potential to significantly reduce the energy consumption of a computer system without reducing its performance when it is in use. Any offline sleep state problem can be solved optimally using a simple algorithm, but the online version of the problem is much harder.

An online algorithm is used to solve a problem where information becomes available over time. Sleator and Tarjan [17] introduced the idea of assessing the worst-case performance of an online algorithm in terms of its competitive ratio, i. e., the ratio between the cost of its solution and an optimal offline solution.

For many problems, lower bounds on the competitive ratio achievable by deterministic online algorithms were shown [9]. To quantify the gap in knowledge between an offline algorithm and an online algorithm $\mathtt{A}$, the framework of *online computation with advice* comes in handy [7,11]. Here, $\mathtt{A}$ is supported by an all-knowing oracle that prepares an infinitely long binary advice tape ϕ before the computation begins. $\mathtt{A}$ may access the bits of ϕ in sequential order at any time during the computation. We are interested in the *advice complexity* of $\mathtt{A}$, i. e., the total number of advice bits that $\mathtt{A}$ reads from ϕ. The goal for the oracle and $\mathtt{A}$ is thus to agree on a method to encode information in as few advice bits as possible. Many online problems have been analyzed in this framework, e. g., graph coloring [3,4,17], disjoint path allocation [2], k-server [6,13,16], or paging [7].

In this paper, we apply this framework to the sleep states management problem with two states. We devise an algorithm with advice that can optimally solve any sleep state problem using $n/2$ advice bits, where n is the length of the input. Starting from this, we devise a c-competitive online algorithm for the problem, for $1 \leq c \leq n$, thus establishing a direct tradeoff between advice complexity and competitive ratio. We construct a complementary algorithm that reads the indices of the longest phases of the input. Moreover, already with only a constant number of advice bits, we can achieve a lot. We establish a parameterized upper bound by constructing another c-competitive online algorithm for the problem, for $c \geq \mathrm{e}/(\mathrm{e}-1)$. In particular, this algorithm is 1.8-competitive with one advice bit and 1.6-competitive with only five advice bits. We complement these upper bounds with a lower bound of $1 + 1/(4w+2) - \varepsilon$ on the competitive ratio of any deterministic online algorithm reading sublinear advice, where w is the wake-up energy.

2 Preliminaries and Related Work

Before we begin, we need to define formally the framework we are using. For an instance I of an online problem, let $\mathcal{O}pt(I)$ denote an optimal solution for I and let $\mathtt{A}(I)$ denote the solution computed by an online algorithm $\mathtt{A}$ on I. Moreover, let $\mathrm{cost}(\mathcal{O}pt(I))$ and $\mathrm{cost}(\mathtt{A}(I))$ denote the costs of the respective solutions. We can now define the competitive ratio formally.

Definition 1. *An online algorithm* $\mathtt{A}$ *for an online minimization problem* U *is* c-competitive *if there is a non-negative constant* α *such that, for any instance* I *of* U*, we have* $\mathrm{cost}(\mathtt{A}(I)) \leq c \cdot \mathrm{cost}(\mathcal{O}pt(I)) + \alpha$*. If* $\alpha = 0$*, then we say that* $\mathtt{A}$ *is* strictly c-competitive. *Moreover,* $\mathtt{A}$ *is* optimal *if it is strictly* 1*-competitive.*

The framework of online computation with advice is defined as follows.

Definition 2. *An* online algorithm with advice *computes, on an input sequence* $I = (r_1, r_2, \ldots, r_n)$*, the output sequence* $\mathtt{A}^{\phi}(I) = (y_1, y_2, \ldots, y_n)$*, where* y_i *is computed from* $r_1, r_2, \ldots, r_i, \phi$*, and* ϕ *is the content of the advice tape, i. e., an infinite binary sequence.*

Definition 3. *An online algorithm* $\mathtt{A}$ *is* c-competitive with advice complexity $s(n)$ *if there is a non-negative constant* α *such that, for every* n *and every instance* I *of length at most* n*, there is a* ϕ *such that* $\mathrm{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \mathrm{cost}(\mathcal{O}pt(I)) + \alpha$ *and at most* $s(n)$ *bits of* ϕ *have been accessed during the computation of* $\mathtt{A}^{\phi}(I)$*. As above, if* $\alpha = 0$*, then* $\mathtt{A}$ *is* strictly c-competitive with advice complexity $s(n)$*, and if* $\mathtt{A}$ *is* optimal *if it is strictly* 1*-competitive.*

In some cases, an algorithm with advice may need to know the entire input; in other cases, a few bits are sufficient to find the optimal solution. For the ski-rental problem [9], for instance, a single advice bit is sufficient to be optimal. For the classic knapsack problem, Böckenhauer et al. [8] showed that a single advice bit is sufficient to be 2-competitive, even though no deterministic online algorithm for this problem can have a bounded competitive ratio.

We consider a computer system that can be in one of two states. There is a wake state in which it can process work and a sleep state that needs less energy but in which the system cannot process work.[1] In every time step, the system either receives a job to process or not. When it is in the sleep state and such a job arrives, it has to wake up; this needs additional energy. Clearly, it is a good idea to go to sleep at the beginning of a long idle period. In an online setting, the algorithm obviously does not know when this is the case. This motivates our treatment of the problem in the advice complexity setting. The formal problem definition is as follows.

Definition 4. *The* sleep states management problem with 2 states (2-SSM) *is the following online minimization problem.*

Let $\{s_1, s_2\}$ *be the set of states;* s_1 *is the wake state and* s_2 *the sleep state. Moreover, let* p *and* w *be positive integers. (If the system is in state* s_1*, it needs* p *energy per time step; if it is in state* s_2*, it needs* w *energy to "wake up.")*

The input $I = (r_1, r_2, \ldots, r_n)$ *consists of a sequence of* n *requests* $r_i \in \{0, 1\}$*. (*$r_i = 1$ *means that there is a job to be processed in the current time step.) After every request* r_i*, an online algorithm* $\mathtt{A}$ *decides which state* s_i *it wants to change to, with the restriction that, if* $r_i = 1$*, then* $\mathtt{A}$ *has to change to state* s_1*. (It has to wake up to process the job.) Formally, the output sequence is* $\mathtt{A}(I) = (y_1, y_2, \ldots, y_n)$*,*

[1] Of course, these states are just what is usually called "on" and "off." To stay consistent with the literature, however, we use the terms "wake state" and "sleep state."

with $y_i = f(r_1, r_2, \ldots, r_{i-1}) \in S$, for some computable function f, and $y_i = s_1$ whenever $r_i = 1$. Initially, $\mathtt{A}$ is in state s_1.

The cost of $\mathtt{A}(I)$ is defined as follows. Every request after which $\mathtt{A}$ changes to or remains in state s_1 costs p. Moreover, every request after which $\mathtt{A}$ changes from state s_2 to state s_1 costs w.

Irani et al. [14] noted that this problem is an instance of the iterated ski rental problem. The offline problem is simple and can be solved optimally using a system that switches off at the beginning of every idle period that is longer than w/p. There is a simple deterministic 2-competitive online algorithm for 2-SSM, and this is also a lower bound for deterministic algorithms [1]. Using randomization, one can construct an online algorithm for 2-SSM with an expected competitive ratio of $\mathrm{e}/(\mathrm{e}-1)$, and this bound is tight for randomized algorithms [15].

3 Upper Bounds

We now consider online algorithms with advice for 2-SSM. We present two c-competitive algorithms, one for $c \in (\mathrm{e}/(\mathrm{e}-1), 2]$ and one for $c \in [1, \mathrm{e}/(\mathrm{e}-1)]$.

We begin by observing that already one advice bit helps us to beat the deterministic competitive ratio of 2 significantly.

Theorem 1. *There is a strictly* 1.8*-competitive online algorithm for* 2*-SSM that reads one advice bit.*

Proof. Let $\mathtt{A}_1$ be the deterministic algorithm that goes to sleep after $b := w/p$ time steps in each idle sequence, and let $\mathtt{A}_2$ be the algorithm that goes to sleep already after $b/2$ time steps. The advice bit then simply indicates which of the two algorithms should be used.

First, note that $\mathtt{A}_1$ is optimal for idle phases of length at most b, and 2-competitive for longer phases. For $\mathtt{A}_2$, we distinguish three cases. For *short* idle phases of length less than $b/2$, $\mathtt{A}_2$ is also optimal. For *medium* idle phases of length l, with $b/2 \leq l < b$, it incurs costs of $b/2 \cdot p + w = 1.5w$, whereas the optimal solution has cost $l \cdot p$. The competitive ratio is therefore $1.5b/l$. Finally, for *long* idle phases of length at least b, $\mathtt{A}_2$ again has cost $b/2 \cdot p + w$, but the optimal solution has cost w, resulting in a competitive ratio of 1.5.

In the following, we assume for computational ease that all idle phases have length at most b. (The adversary has no advantage of producing longer phases, i. e., it cannot increase the algorithms' competitive ratio.)

Let x_0, x_1, x_2 be the fraction of short, medium, and long phases, respectively. The competitive ratio of $\mathtt{A}_1$ is therefore $x_0 \cdot 1 + x_1 \cdot 1 + x_2 \cdot 2$, and the one of $\mathtt{A}_2$ is $x_0 \cdot 1 + x_1 \cdot 1.5b/l + x_2 \cdot 1.5 \leq x_0 \cdot 1 + x_1 \cdot 3 + x_2 \cdot 1.5$. Since our algorithm takes the better one, it obtains a competitive ratio of $\min\{x_0 + x_1 + 2x_2, x_0 + 3x_1 + 1.5x_2\}$. As we are interested in an upper bound, we need to find values that maximize this term. Since both $\mathtt{A}_1$ and $\mathtt{A}_2$ are optimal on short periods, we can safely ignore them for our worst-case analysis. That is, we set $x_0 := 0$ above. Also, we have $x_0 + x_1 + x_2 = 1$ and hence $x_1 = 1 - x_2$. The expression above thus simplifies to

$$\min\{0 + 1 - x_2 + 2x_2, 0 + 3(1 - x_2) + 1.5x_2\} = \min\{1 + x_2, 3 - 1.5x_2\}.$$

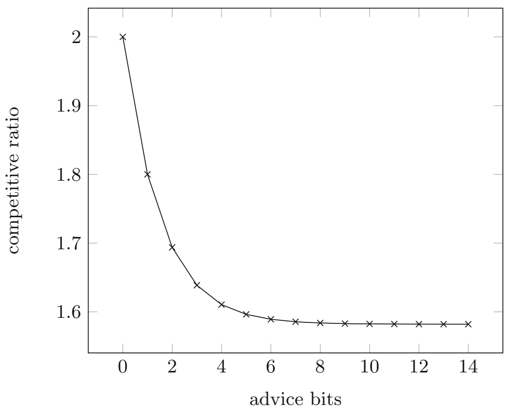


Fig. 1. The upper bound from Theorem 2, for $1 \leq l \leq 14$

Setting $1 + x_2 = 3 - 1.5x_2$ yields $x_2 = 0.8$ and thus $x_1 = 0.2$. The competitive ratio of our algorithm is thus at most $\min\{1 + 0.8, 3 - 1.5 \cdot 0.8\} = 1.8$. $\square$

We can generalize this approach to more advice bits. For this, we need the following technical result. The proof has been omitted due to space restrictions.

Lemma 1. *For any two positive integers l and i, we have*

$$\frac{(1+2^l)^{i-1}2^l}{2^{(i+1)l}} = \frac{1}{2^l} + \sum_{j=1}^{i-1} \frac{(1+2^l)^{j-1}}{2^{(j+1)l}}.$$

The following upper bound starts at a competitive ratio of 2 with no advice and converges fast to $\mathrm{e}/(\mathrm{e}-1) \approx 1.5820$, as one can see in Section 3. Note that already five advice bits suffice to achieve a competitive ratio below 1.6.

Theorem 2. *There is an online algorithm for 2-SSM that reads l advice bits, where $l \geq 0$ may be a function of n, and achieves a competitive ratio of*

$$1 + \frac{1}{\left(1+\frac{1}{2^l}\right)^{2^l} - 1}.$$

Proof. Let again $b := w/p$, and let $\mathtt{A}_i$ be the deterministic algorithm that goes to sleep after $ib2^{-l}$ time steps in each idle sequence, for $1 \leq i \leq 2^l$. The advice bits again simply indicate which of the 2^l algorithms to use. On idle periods of length at most $2^{-l}b$, all 2^l algorithms are optimal. The more interesting part, however, is to figure out the algorithms' competitive ratio for longer idle periods. To this end, we first determine the power consumption of each algorithm and then the power consumption of an optimal algorithm.

In the following, a *j-period,* for $0 \leq j \leq 2^l-1$, is an idle period of length in the interval $(j2^{-l}b, (j+1)2^{-l}b]$, and a *2^l-period* is an idle period of length b. (As in the proof of Theorem 1, we assume without loss of generality that all idle phases have length at most b.) Moreover, for $1 \le i \le 2^l$ and $0 \le j \le 2^l$, let $\text{cost}(\texttt{A}_i(\text{IP}_j))$ denote the total cost of algorithm $\texttt{A}_i$ on a j-period. We define $\text{cost}(\mathcal{O}pt(\text{IP}_j))$ analogously. Finally, let $\text{CR}_i(\text{IP}_j) = \text{cost}(\texttt{A}_i(\text{IP}_j))/\,\text{cost}(\mathcal{O}pt(\text{IP}_j))$.

One easily verifies that $\texttt{A}_i$ is optimal on idle periods of length at most $i2^{-l}b$. Formally, $\text{CR}_i(\text{IP}_j)) = 1 \iff j \le i-1$. We are, however, more interested in the values $\text{CR}_i(\text{IP}_j)$, for $j \ge i$.

$\texttt{A}_i$ falls asleep after time step $i2^{-l}b$ and then needs w energy to wake up again, so $\text{cost}(\texttt{A}_i(\text{IP}_j)) = i2^{-l}bp + w$, for $1 \le i \le 2^l$. An optimal algorithm stays awake all the time. That is, $\text{cost}(\mathcal{O}pt(\text{IP}_j)) \ge j2^{-l}bp$, for $0 \le j \le 2^l$. Combining all of this yields

$$\text{CR}_i(\text{IP}_j) = 1, \quad \text{for } 0 \le j \le i-1, \quad \text{and}$$
$$\text{CR}_i(\text{IP}_j) = \frac{\text{cost}(\texttt{A}_i(\text{IP}_j))}{\text{cost}(\mathcal{O}pt(\text{IP}_j))} \le \frac{i2^{-l}bp + w}{j2^{-l}bp} = \frac{i+2^l}{j}, \quad \text{for } i \le j \le 2^l.$$

To continue our analysis, let x_j be the fraction of j-periods in the whole input, for $0 \le j \le 2^l$. Obviously, we have $\sum x_j = 1$. For notational ease, let $\boldsymbol{x} := (x_1, x_2, \ldots, x_{2^l})$, and let $\text{CR}_i(\boldsymbol{x})$ denote the competitive ratio of $\texttt{A}_i$ on an instance where the fractions of idle periods correspond to $\boldsymbol{x}$.

Note that there may be arbitrarily many jobs between two idle periods. If there are such jobs, however, they only improve the algorithm's competitive ratio. Therefore, without loss of generality, we do not consider these jobs, but only the competitive ratio of $\texttt{A}_i$ on the idle periods. We get

$$\text{CR}_i(\boldsymbol{x}) \le \sum_{j=0}^{2^l} x_j \,\text{CR}_i(\text{IP}_j) = \sum_{j=0}^{i-1} x_j + (i+2^l)\sum_{j=i}^{2^l} \frac{x_j}{j}. \tag{1}$$

The oracle writes the index of the best algorithm on the tape. The adversary tries to construct a worst-case instance by selecting the values x_j appropriately. We can safely assume that it sets $x_0 := 0$, since *all* algorithms are optimal on these very short idle periods. Therefore, $\texttt{Adv}$ tries to maximize $\min_{1 \le i \le 2^l}\{\text{CR}_i(\boldsymbol{x})\}$ by choosing $\boldsymbol{x} = (x_1, x_2, \ldots, x_{2^l}) \in [0,1]^{2^l}$ such that $\sum x_j = 1$.

We now show that the best strategy for the adversary is to select values $x_1, x_2, \ldots, x_{2^l}$ such that all algorithms $\texttt{A}_i$ can guarantee the same competitive ratio on corresponding instances. Assume for contradiction that there was some better strategy for $\texttt{Adv}$, i. e., one in which the competitive ratios of all 2^l algorithms is higher. Let $\boldsymbol{x}' := (x'_1, x'_2, \ldots, x'_{2^l})$ be the respective values. Clearly, $\boldsymbol{x} \ne \boldsymbol{x}'$. Let j be the highest index at which $\boldsymbol{x}$ and $\boldsymbol{x}'$ differ. That is, $x_i = x'_i$, for all $i > j$. In other words, let $j \in \{2, 3, \ldots, 2^l\}$ be the maximum value such that we have

$$\sum_{i=1}^{j-1} x'_i < \sum_{i=1}^{j-1} x_i \quad \text{and} \quad \sum_{i=j}^{2^l} x'_i > \sum_{i=j}^{2^l} x_i \quad \text{or} \tag{2}$$

$$\sum_{i=1}^{j-1} x'_i > \sum_{i=1}^{j-1} x_i \quad \text{and} \quad \sum_{i=j}^{2^l} x'_i < \sum_{i=j}^{2^l} x_i. \tag{3}$$

If (2) holds, then we have $\mathrm{CR}_1(\boldsymbol{x}') < \mathrm{CR}_1(\boldsymbol{x})$, because the coefficients of the x_i in $\mathrm{CR}_1(\boldsymbol{x})$ decrease, and therefore the overall value becomes smaller again. If (3) holds, then we have $\mathrm{CR}_j(\boldsymbol{x}') < \mathrm{CR}_j(\boldsymbol{x})$, since the coefficients of all x_i with $i < j$ in $\mathrm{CR}_j(\boldsymbol{x})$ are 1 and thus smaller than those with $i \geq j$. Therefore, the overall value becomes smaller again. In both cases, the competitive ratio of at least one algorithm decreases, and thus selecting the x_i such that all algorithms can guarantee the same competitive ratio is indeed the best strategy for the adversary. We claim that the values

$$x_{2^l} = \frac{1}{1+S} \quad \text{and} \quad x_{2^l-j} = \frac{(2^l-j)(1+2^l)^{j-1}}{2^{(j+1)l}} x_{2^l}, \text{ for } 1 \leq j \leq 2^l-1, \tag{4}$$

satisfy this condition, where $S = \sum_{j=1}^{2^l-1}(2^l-j)(1+2^l)^{j-1}/2^{(j+1)l}$.

First, note that $\sum_{j=1}^{2^l} x_j = \sum_{j=1}^{2^l-1} x_j + x_{2^l} = x_{2^l}S + x_{2^l} = \frac{S}{1+S} + \frac{1}{1+S} = 1$.

Second, we need to show that indeed all algorithms achieve the same competitive ratio. We consider the difference of the competitive ratios of algorithms $\mathtt{A}_{2^l-i+1}$ and $\mathtt{A}_{2^l-i}$, for $1 \leq i \leq 2^l-1$. Using (1) and the fact that $x_0 = 0$, we get

$$\begin{aligned}
\mathrm{CR}(\mathtt{A}_{2^l-i+1}(I)) - \mathrm{CR}(\mathtt{A}_{2^l-i}(I)) &= \sum_{j=1}^{2^l-i} x_j + (2^l-i+1+2^l) \sum_{j=2^l-i+1}^{2^l} \frac{x_j}{j} \\
&\quad - \sum_{j=1}^{2^l-i-1} x_j - (2^l-i+2^l) \sum_{j=2^l-i}^{2^l} \frac{x_j}{j} \\
&= x_{2^l-i} - (2^{l+1}-i)\frac{x_{2^l-i}}{2^l-i} + \sum_{j=2^l-i+1}^{2^l} \frac{x_j}{j} \\
&= x_{2^l-i} - (2^{l+1}-i)\frac{x_{2^l-i}}{2^l-i} + \frac{x_{2^l}}{2^l} + \sum_{j=1}^{i-1} \frac{x_{2^l-j}}{2^l-j}
\end{aligned}$$

Using (4), we obtain

$$\begin{aligned}
&\mathrm{CR}(\mathtt{A}_{2^l-i+1}(I)) - \mathrm{CR}(\mathtt{A}_{2^l-i}(I)) \\
&= x_{2^l} \cdot \left(\frac{(2^l-i)(1+2^l)^{i-1}}{2^{(i+1)l}} - \frac{(2^{l+1}-i)(1+2^l)^{i-1}}{2^{(i+1)l}} + \frac{1}{2^l} + \sum_{j=1}^{i-1} \frac{(1+2^l)^{j-1}}{2^{(j+1)l}} \right) \\
&= x_{2^l} \cdot \left(-\frac{(1+2^l)^{i-1}2^l}{2^{(i+1)l}} + \frac{1}{2^l} + \sum_{j=1}^{i-1} \frac{(1+2^l)^{j-1}}{2^{(j+1)l}} \right),
\end{aligned}$$

where the second factor is 0 due to Lemma 1. Thus, all algorithms achieve the same competitive ratio on instances with idle periods according to (4). Using (1) and the fact that $x_0 = 0$, we get $\mathrm{CR}(\mathtt{A}_{2^l}(I)) \leq \sum_{j=1}^{2^l-1} x_j + 2^{l+1}x_{2^l}/2^l = \sum_{j=1}^{2^l-1} x_j + 2x_{2^l} = 1 + x_{2^l} = 1 + 1/(1+S)$. It remains to show that $1+1/(1+S) = 1 + 1/((1+1/2^l)^{2^l} - 1)$, which is equivalent to showing that $S = (1+1/2^l)^{2^l} - 2$. We have

$$\begin{aligned}
S &= \sum_{j=1}^{2^l-1} \frac{(2^l-j)(1+2^l)^{j-1}}{2^{(j+1)l}} = \frac{1}{2^{2^l \cdot l}} \cdot \sum_{j=1}^{2^l-1} 2^{l(2^l-1-j)} \cdot (2^l-j) \cdot (1+2^l)^{j-1} \\
&= \frac{1}{2^{2^l \cdot l}} \cdot \sum_{j=1}^{2^l-1} 2^{l(j-1)} \cdot j \cdot (1+2^l)^{2^l-1-j} = \frac{(1+2^l)^{2^l-1}}{2^{(2^l+1)l}} \cdot \sum_{j=1}^{2^l-1} j \cdot \frac{2^{lj}}{(1+2^l)^j} \\
&= \frac{1}{(1+2^l)^2} \cdot \left(\frac{1+2^l}{2^l}\right)^{2^l+1} \cdot \sum_{j=1}^{2^l-1} j \cdot \left(\frac{2^l}{1+2^l}\right)^j .
\end{aligned}$$

We have now simplified the sum to a standard form. It is well-known that $\sum_{j=1}^{n} jx^j = ((nx-n-1)x^{n+1}+x)/(1-x)^2$ [12]. Using this, we get

$$\begin{aligned}
S &= \frac{1}{(1+2^l)^2} \cdot \left(\frac{1+2^l}{2^l}\right)^{2^l+1} \cdot \frac{((2^l-1)\frac{2^l}{1+2^l} - (2^l-1) - 1)\left(\frac{2^l}{1+2^l}\right)^{2^l} + \frac{2^l}{1+2^l}}{\left(\frac{1}{1+2^l}\right)^2} \\
&= \left(\frac{1+2^l}{2^l}\right)^{2^l+1} \cdot \left(\left(\frac{2^l-1}{2^l+1}2^l - 2^l\right)\left(\frac{2^l}{1+2^l}\right)^{2^l} + \frac{2^l}{1+2^l}\right) \\
&= (1+2^l)\left(\frac{2^l-1-(2^l+1)}{2^l+1}\right) + \left(\frac{1+2^l}{2^l}\right)^{2^l} = -2 + \left(1+\frac{1}{2^l}\right)^{2^l} . \qquad \square
\end{aligned}$$

Note that this result implies that, for an arbitrarily small $\varepsilon > 0$, there is an $(\mathrm{e}/(\mathrm{e}-1)+\varepsilon)$-competitive online algorithm that reads only a constant number of advice bits. For the range $[1, \mathrm{e}/(\mathrm{e}-1)]$, we consider two different algorithms.

For the first one, first observe that at least one job has to be processed between any two idle periods. Therefore, an input instance of length n contains at most $n/2$ idle periods. For optimality, an algorithm can simply read one advice bit at the beginning of each idle period to determine the state it should change to. Generalizing this, we obtain an algorithm that provides advice for a certain prefix of the input instance.

Theorem 3. *There is an online algorithm* A *for 2-SSM that achieves a strict competitive ratio of* $n - 4l + (4l^2+2l)/n$ *and reads at most* l *advice bits.*

Proof. First, A reads the l advice bits from the tape. This is sufficient to be optimal on a prefix of the instance of length $2l$, since the number of idle periods can at most be half of the number of requests, as we already observed above.

More precisely, A behaves as follows on the first $2l$ requests. Whenever a new idle period begins, it reads an advice bit to determine whether to fall asleep or not. (There can be at most l idle periods in this part of the instance.) For the remaining part of the instance, A always stays awake.

Let us now analyze the competitive ratio of A. On the instance prefix of length $2l$, it is optimal, i.e., it has a competitive ratio of $\mathrm{CR}_1 = 1$. The worst case for the remaining instance is that there is a job at the beginning and then only idle requests until the very end. The algorithm stays awake and incurs a cost of $n-2l$, although an optimal algorithm would fall asleep immediately after the job and thus incur a cost of 1. Therefore, A achieves a competitive ratio of $\mathrm{CR}_2 = n - 2l$ on the second part of the instance. The total competitive ratio is the weighted average of these two competitive ratios, i. e.,

$$\frac{2l}{n} \cdot \mathrm{CR}_1 + \frac{n-2l}{n} \cdot \mathrm{CR}_2 = \frac{2l}{n} \cdot 1 + \frac{n-2l}{n} \cdot (n-2l) = n - 4l + \frac{4l^2+2l}{n}. \qquad \square$$

For $l = n/2$, this yields that $n/2$ advice bits suffice to be optimal.

For the second algorithm, the oracle encodes the starting indices of the longest idle phases. The algorithm then falls asleep during those phases and remains awake at all others.

Theorem 4. *There is an online algorithm* A *for* 2*-SSM with a strict competitive ratio of* $1 - 1/\lfloor l/\lceil \log n \rceil \rfloor + n/\lfloor l/\lceil \log n \rceil \rfloor^2$ *that reads at most* l *advice bits.*

Proof. The algorithm A first reads the number $\lceil \log n \rceil$ from the advice tape. This needs at most $\lceil \log n \rceil$ advice bits. Then, A reads the starting indices of the $k := \lfloor l/\lceil \log n \rceil \rfloor - 1$ longest idle periods from the advice tape for which the optimal strategy is to fall asleep. (If there are less such phases, we can simply encode some indices several times.) Each such index can be encoded using $\lceil \log n \rceil$ advice bits; therefore, A reads $\lceil \log n \rceil + k\lceil \log n \rceil \leq l$ advice bits in total.

The behavior of A is now fairly obvious: For all these idle periods, it falls asleep. For all other idle periods, it stays awake.

It remains to analyze the competitive ratio of A. In the following, we assume, without loss of generality, that $k+1$ divides n. (Otherwise, our bound only gets better.) Clearly, the worst case is when there is one long idle period for which A has no advice. The maximum possible length of this idle period is $n/(k+1)$, which is, e. g., the case when there are $k+1$ idle periods in total, all of equal length. This results in a competitive ratio of 1 on a fraction of $k/(k+1)$ of the input, and a competitive ratio of $n/(k+1)$ on the remaining fraction of $1/(k+1)$ of the input. The competitive ratio of A is the weighted sum of these two competitive ratios, i. e., $k/(k+1) \cdot 1 + 1/(k+1) \cdot n/(k+1) = k/(k+1) + n/(k+1)^2$. $\square$

Observe that the bound of Theorem 4 is strictly greater than 1 for any n. This means that, for every n, there is a $c \in (1, \mathrm{e}/(\mathrm{e}-1)]$ such that only Theorem 3 establishes an upper bound in the range $[1, c]$. Conversely, for the values $n = 2^{2^m}$, for some $m \in \mathbb{N}$, and $l = n/4 = 2^{2^k-2}$, Theorem 4 yields an upper bound of $1 - 1/(2^{2^m-m-2}) + 1/(2^{2^m-2m-4})$, which is clearly much better than the bound $2^{2^m-2} + 1/2$ from Theorem 3. Moreover, it tends to 1 for increasing n and thus is below $\mathrm{e}/(\mathrm{e}-1)$ for infinitely many n. Hence, the two results are complementary.

4 Lower Bounds

Recently, a general technique was established to prove lower bounds on the advice complexity of online problems. This technique consists of a reduction from the string guessing problem, which was studied by Böckenhauer et al. [5]. As the name implies, the problem consists of guessing a string over an alphabet of fixed size. Here, we consider the variant with known history, i. e., the algorithm gets immediate feedback whether its guess for the current character was correct.

Definition 5. *The* bit guessing problem with known history *(BGKH) is the following online minimization problem. The input $I = (n, d_1, d_2, \dots, d_n)$ consists of a natural number n and the bits $d_1, d_2, \dots, d_n$, which are revealed one by one. An online algorithm $\mathtt{A}$ computes the output sequence $\mathtt{A}(I) = y_1y_2 \dots y_n$, where $y_i = f(n, d_1, d_2, \dots, d_{i-1}) \in \{0, 1\}$, for some computable function f. The algorithm is not required to respond with any output in the last time step. The cost of a solution $\mathtt{A}(I)$ is the number of wrongly guessed bits, i. e., the Hamming distance* $\mathrm{Ham}(d, \mathtt{A}(I))$ *between $d = d_1d_2 \dots d_n$ and $\mathtt{A}(I)$.*

The following lower bound for BGKH is known.

Theorem 5 (Böckenhauer et al. [5]). *Every deterministic algorithm for BGKH that can guarantee to be correct in more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1 - H(\alpha))n$ many advice bits, where H denotes the binary entropy function, i. e., $H(\alpha) := -(1 - \alpha) \log_2(1 - \alpha) - \alpha \log_2 \alpha$.* □

We use this to establish a linear lower bound on the advice complexity for c-competitiveness. The next result shows that linear advice is necessary not only to achieve optimality, but also to get arbitrarily close to it. (Note that we can always scale p and w and therefore assume $p = 1$ without loss of generality.)

Theorem 6. *Every online algorithm for 2-SSM needs to read at least*

$$\frac{1 - H(2w + 2 - (2w + 1)c)}{2w + 2} \cdot n \in \Theta(n)$$

advice bits to be strictly $(c - \varepsilon)$-competitive, for $1 < c \leq 1 + \frac{1}{4w+2}$ and any $\varepsilon > 0$.

Proof. We prove the statement by a reduction from BGKH. For any string guessing instance $(m, d_1, d_2, \dots, d_m)$, we construct a corresponding 2-SSM instance of length $(2w + 2)m$, consisting of subintervals of the form

$$(r_1, r_2, \dots, r_{2w+2}) = (1, \underbrace{0, 0, \dots, 0}_{w-1}, 1, \underbrace{0, 0, \dots, 0, 0, 0}_{w+1}) \qquad \text{if } d_i = 0 \quad \text{and}$$

$$(r_1, r_2, \dots, r_{2w+2}) = (1, \underbrace{0, 0, \dots, 0, 0, 0}_{w+1}, 1, \underbrace{0, 0, \dots, 0}_{w-1}) \qquad \text{if } d_i = 1,$$

where we write r_j instead of $r_{i,j}$, for $1 \leq j \leq 2w + 2$, to improve readability.

We first look at what an optimal algorithm $\mathcal{O}pt$ does on such an instance. Since $\mathcal{O}pt$ always falls asleep for idle periods that are longer than the break-even point $b := w/p = w$, it is asleep during the second idle period for intervals of the first type, and during the first idle period for intervals of the second type.

Put differently, for the first type of interval, $\mathcal{O}pt$ falls asleep on r_{w+2}, so far having a cost of $w + 1$. To wake up when the next interval starts, it incurs a cost of w. The total cost is therefore $2w + 1$. On intervals of the second type, $\mathcal{O}pt$ falls asleep on r_2, then wakes up again on r_{w+3} and stays awake until the end of the subinterval. Therefore, it has cost $1 + w + w = 2w + 1$. That is, $\mathcal{O}pt$ incurs a cost of $(2w + 1)m$.

Suppose now that there was an online algorithm A that achieves a competitive ratio of $c - \varepsilon$, for $1 < c \leq 1 + 1/(4w + 2)$, and reads less than $(1 - H(2w + 2 - (2w + 1)c))m$ advice bits. We assume, without loss of generality, that A knows the structure of the instances and hence always behaves optimal for the second idle period of a subinterval, i. e., for the requests $r_{w+2}, r_{w+3}, \ldots, r_{2w+2}$. That is, if $d_i = 0$, then A sleeps on those requests, and if $d_i = 1$, then A is awake on the requests $r_{w+3}, r_{w+4}, \ldots, r_{2w+2}$. (Whether it is already awake on r_{w+2} depends on previous requests.)

For r_1, A always has to be awake. For $r_2, r_3, \ldots, r_w$, A does not know the type of the current subinterval. We now argue briefly that we can assume, without loss of generality, that, in each subinterval, A is either awake on all these requests or asleep on all these requests. In other words, it does not fall asleep or wake up when processing such a request. For the latter, the argumentation is trivial. No reasonable algorithm ever wakes up on an idle request.[2] On the other hand, the algorithm knows that there is always an idle period on the requests $r_2, r_3, \ldots, r_w$. Therefore, any algorithm that falls asleep on any of these requests can be replaced by an algorithm that falls asleep on r_2 and that has at least the same competitive ratio. From r_{w+1} on, A knows the type of the current subinterval and therefore behaves like $\mathcal{O}pt$.

For an interval of the first type, if A falls asleep on the second request, it incurs cost $1+w+1+w = 2w+2$. If it stays awake, it incurs cost $1+(w-1)\cdot 1+1+w = 2w+1$. For an interval of the second type, if A falls asleep on the second request, it incurs cost $1 + w + 1 + (w - 1) \cdot 1 = 2w + 1$. If it stays awake, it incurs cost $1 + (w + 1) \cdot 1 + 1 + (w - 1) \cdot 1 = 2w + 2$.

We can use A to construct an online algorithm A′ for BGKH as follows. The algorithm A′ simply outputs 1 whenever A falls asleep on the second request, and it outputs 0 otherwise. One easily sees that, if A outputs a solution with cost $(2w + 1)m + k$, then A′ outputs k wrong bits. We already know that A is $(c - \varepsilon)$-competitive, i. e., it outputs a solution of cost less than

$$(2w + 1)mc = (2w + 1)m(1 + c - 1) = (2w + 1)m + \underbrace{(2w + 1)m(c - 1)}_{k}.$$

Therefore, A′ outputs less than $k = (2w+1)m(c-1)$ wrong bits, i. e., A′ outputs more than $m - k = (2w + 2 - (2w + 1)c)m$ correct bits. Moreover, since A reads less than $(1 - H(2w + 2 - (2w + 1)c))m$ advice bits, so does A′. Setting $\alpha := 2w + 2 - (2w + 1)c$, however, this contradicts Theorem 5, and therefore there is no such algorithm A′. Note that the condition $1/2 \leq \alpha < 1$ holds due to the range of c. □

[2] This could be characterized as being "lazy", using the same term as in the k-server problem, where it denotes a similar property of algorithms.

References

1. Albers, S.: Energy-efficient algorithms. Comm. of the ACM **53**, 86–96 (2010)
2. Barhum, K., Böckenhauer, H.-J., Forišek, M., Gebauer, H., Hromkovič, J., Krug, S., Smula, J., Steffen, B.: On the Power of Advice and Randomization for the Disjoint Path Allocation Problem. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 89–101. Springer, Heidelberg (2014)
3. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Keller, L.: Online Coloring of Bipartite Graphs with and without Advice. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 519–530. Springer, Heidelberg (2012)
4. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Krug, S., Steffen, B.: On the Advice Complexity of the Online $L(2,1)$-Coloring Problem on Paths and Cycles. Theoretical Computer Science **554**, 22–39 (2014)
5. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The String Guessing Problem as a Method to Prove Lower Bounds on the Advice Complexity. Theoretical Computer Science **554**, 95–108 (2014)
6. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.): ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
7. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the Advice Complexity of Online Problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
8. Böckenhauer, H.-J., Komm, D., Královič, R., Rossmanith, P.: On the Advice Complexity of the Knapsack Problem. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012)
9. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
10. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: USENIX 2010, pp. 21–21 (2010)
11. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoretical Computer Science **412**(24), 2642–2656 (2011)
12. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics. A Foundation for Computer Science, 2nd edn. Addison-Wesley (1994)
13. Gupta, S., Kamali, S., López-Ortiz, A.: On Advice Complexity of the k-server Problem under Sparse Metrics. In: Moscibroda, T., Rescigno, A.A. (eds.) SIROCCO 2013. LNCS, vol. 8179, pp. 55–67. Springer, Heidelberg (2013)
14. Irani, S., Shukla, S., Gupta, R.: Algorithms for power savings. ACM Transactions on Algorithms 3(4) (2007)
15. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.: Competitive randomized algorithms for non-uniform problems. In: SODA 1990, pp. 301–309 (1990)
16. Renault, M.P., Rosén, A.: On Online Algorithms with Advice for the k-Server Problem. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 198–210. Springer, Heidelberg (2012)
17. Seibert, S., Sprock, A., Unger, W.: Advice Complexity of the Online Coloring Problem. In: Spirakis, P.G., Serna, M. (eds.) CIAC 2013. LNCS, vol. 7878, pp. 345–357. Springer, Heidelberg (2013)
18. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM **28**(2), 202–208 (1985)

Multi-Radio Channel Detecting Jamming Attack Against Enhanced Jump-Stay Based Rendezvous in Cognitive Radio Networks

Yang Gao[1(✉)], Zhaoquan Gu[1], Qiang-Sheng Hua[2], and Hai Jin[2]

[1] Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, People's Republic of China
y-gao13@mails.tsinghua.edu.cn

[2] Services Computing Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, People's Republic of China

Abstract. Rendezvous problem has been attracting much attention as a fundamental process to construct the Cognitive Radio Network (CRN). Many elegant algorithms have been proposed to achieve rendezvous on some common channel in a short time, but they are vulnerable to the jamming attacks where a jammer may exist listening or blocking the channels[14]. In this paper, we propose the Multi-Radio Channel Detecting Jamming Attack (MRCDJA) problem where the jammer can access multiple channels simultaneously. We assume the users adopting the Enhanced Jump Stay (EJS)[8] algorithm, which guarantees rendezvous by generating a channel hopping sequence, and our goal is to determine the hopping sequence as quickly as possible.

1 Introduction

Cognitive Radio Network (CRN) has become a new paradigm to alleviate the spectrum scarcity problem since the unlicensed spectrum is overcrowded while the licensed spectrum has low utilization[15]. There are two kinds of users in CRN, the so-called primary users (PUs) who own the licensed spectrum and the secondary users (SUs) who can use the particular licensed spectrum that are not occupied. Unless otherwise specified, 'users' mentioned hereafter refers to SUs.

In constructing such a CRN, *rendezvous* is a fundamental procedure for the users to establish a link on a common licensed channel for communication[10]. Many algorithms have been proposed to reduce the time needed to rendezvous[2, 4,8,9] by constructing a hopping sequence. Let U be the set of the available channels and $|U| = M$, the state-of-the-art results guaranteed rendezvous in $O(M^2)$ time slots[8]. However, these algorithms are vulnerable to Channel Detecting Jamming Attack (CDJA)[14] where a jammer exists trying to listen or block the

This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61103186, 61033001, 61361136003.

channels[14]. The intuition of the CDJA is to determine one user's channel hopping sequence based on the listened channels and to jam the predicted channels against rendezvous.

Recently, multiple radios architecture has been widely used in wireless networks[1,7,12,13] and many problems can be solved efficiently. In this paper, we propose Multi-Radio Channel Detecting Jamming Attack (MRCDJA) where the jammer can listen or block n ($n > 1$) channels simultaneously and the goal is to determine the user's channel hopping sequence as quick as possible. Since Enhanced Jump Stay (EJS)[8] is one representative state-of-the-art algorithm, we design multi-radio jamming algorithm against EJS.

In this paper, we first review the EJS algorithm and the CDJA algorithm as the cornerstones, then we design an efficient algorithm to determine the channel hopping sequence when the users can use any channel in the set U. We show that the sequence can be figured out in $O(\frac{M}{n})$ expected time, which is n times better than the result in [14]. Moreover, when some channels are occupied by the PUs, we provide another efficient algorithm for the MRCDJA problem, we show that if the ratio of the available channels is more than 40%, our algorithms can guarantee fast successful attack with probability more than 80%. We also evaluate our proposed algorithms and the results show that our algorithms can determine the hopping sequence quickly.

The rest of the paper is organized as follows. Preliminaries are provided in the next section. In Section 3, we review the EJS algorithm. In Section 4, we show our main results by providing efficient algorithms to figure out the channel hopping sequences. In Section 5, we conduct simulations for our algorithms. Finally, we conclude the paper in Section 6.

2 Preliminaries

We consider a CRN with two users that are trying to rendezvous with each other and a jammer who aims to break the rendezvous process by blocking some licensed channels. We suppose the licensed spectrum is divided into M non-overlapping channels labeled $U = \{1, 2, \ldots, M\}$ and the labels are known to the users and the jammer.[1]

Time is supposed to be divided into slots of equal length of $2t$, where t is the time duration for establishing a connection. In each time slot, the user can access one channel for rendezvous attempt, while the jammer can choose $n > 1$ channels for listening (detecting) or blocking. Here listening or detecting means that the jammer can check some channels to see if the user is accessing these channels in the same time. The users can achieve rendezvous only if they access the same channel in the same time slot and the jammer doesn't block the channel. We assume the users start the rendezvous process asynchronously, i.e. the two users doesn't need to start in the same time slot. Obviously, we can

[1] It is reasonable to make this assumption since the channels are easy to be distinguished by frequency, however there are some research on the case when the labels are not known to the users[5,6].

consider the rendezvous process as slot-aligned even in the asynchronous setting since the duration of each time slot is $2t$[6]. In this paper, we assume the jammer starts detecting the channels before all users starting to access channels.

Before the users start the rendezvous process, they sense the licensed spectrum to check whether the channels in U are occupied. We say that a channel is *available* for the user if it is not occupied by any nearby PUs. After the spectrum sensing stage, each user i has a set of available channels $C_i \subseteq U$ and they can begin rendezvous attempt by accessing the available channels. The users are said to be *symmetric* if the available channel set is U for both of them, i.e. $C_1 = C_2 = U$[8], otherwise, they are *asymmetric*, i.e. $C_1 \subseteq U, C_2 \subseteq U$[8]. In this paper, we design efficient algorithms for the jammer to detect the channel hopping sequence to prevent rendezvous between the users. We assume the users adopting the Enhanced Jump Stay (EJS)[8] algorithm and formulate the problem as:

Problem 1 (MRCDJA). Suppose two users run the EJS algorithm for rendezvous, while the jammer who can access $n > 1$ channels simultaneously tries to determine the channel hopping sequence of one user. The goal is to design an efficient algorithm to detect the sequence as quickly as possible.

3 Review of the Enhanced Jump-Stay Algorithm

Since Enhanced Jump Stay (EJS)[8] is one representative rendezvous algorithm, we review the intuition of the algorithm in this section. The algorithm works as follows: each user generates its own channel hopping sequence in rounds. Each round consists of three jump patterns and one stay pattern where each pattern lasts for P time slots (P is the smallest prime number such that $P \geq M$). In the first round, the user chooses a random channel $i \in [1, P]$ as a start channel and a random number $r \in C$ as the step length, where C is the available channel set of this user. In the jump patterns of the first round where time t suits $0 \leq t < 3P$, the user chooses channel $(i + tr - 1) \mod P + 1$, then in the stay pattern $(3P \leq t < 4P)$, the user stays on channel r. The j-th $(j > 1)$ round is generated in the same way, except that the start channel is $(i + j - 2)\%P + 1$ and the step length remains r. For example $M = 5, P = 5, i = 3, r = 1$, we show the first two rounds of the channel sequence:

3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,1,1,1,1,1,(3+1),5,1,2,3,4,5,1,2,3,4,5,1,2,3,1,1,1,1,1,...

Notice that $(3 + 1)$ represents the start channel of the second round as described above.

There are two special situations we need to consider. The first one is $P \neq M$, channel x $(x > M)$ doesn't exist and thus we map it to $x - M$. The second one is the asymmetric situation, where some channels in U may be occupied by the PUs, and thus the user just choose a random channel in its available channel set to replace the unavailable channel.

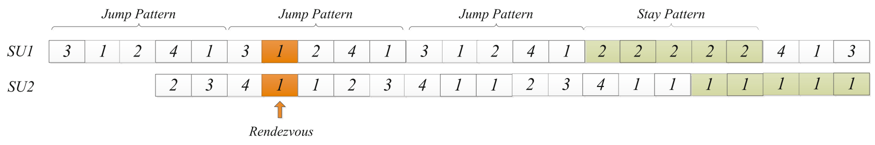


Fig. 1. Example for rendezvous of two users

For example, suppose $M = 4, P = 5$, the available channel set $C = \{1, 3, 4\}$, $i = 3, r = 1$. It is obvious that channel 5 doesn't exist and it should be mapped to channel 1. Since channel 2 doesn't belong to C, the user chooses a random channel from C once it should access channel 2. Therefore, the new sequence is:

3,4,1,1,3,3,4,1,1,4,3,4,1,1,1,1,1,1,1,1,(3+1),1,1,1,3,4,1,1,4,3,4,1,1,4,3,1,1,1,1,1,...

In this paper, we use two important lemmas in [11] as:

Lemma 1. *Given a positive integer P, if $r \in [1, P)$ is relatively prime to P, i.e., the common factor between them is 1, then for any $x \in [0, P)$ the sequence $S =< x\%P + 1, (x + r)\%P + 1, ..., (x + (P - 1)r)\%P + 1 >$ is a permutation of $< 1, 2, ..., P >$.*[2] *[11]*

Lemma 2. *Given a prime number P, if r_1 and r_2 are two different numbers in $(0, P)$, then for any $x_1, x_2 \in [0, P)$, there must be an integer $k \in [0, P)$ such that $(x_1 + kr_1)\%P = (x_2 + kr_2)\%P$. [11]*

Combining these two Lemmas, the following theorem is proved.

Theorem 1. *Under the symmetric setting, any two users performing EJS achieve rendezvous in at most $4P$ time slots, under asymmetric setting the time is at most $4P(P + 1 - G)$ time slots, where P is the smallest prime number greater or equal to M and G is the number of channels commonly available to the two users. [8]*

Fig. 1 is an example of rendezvous under symmetric setting. Here $U = \{1, 2, 3, 4\}, P = 5$, user 1 chooses channel 3 as its start channel and 2 as its step length, user 2 chooses channel 2 as its start channel and 1 as its step length. User 2 starts 3 time slots later than user 1, they rendezvous at the 4th time slot (the start time is based on the later user's clock) on channel 1, here channel 1 is mapped from channel 5 for both users.

Remark 1. The Expected Time To Rendezvous (ETTR) of EJS is proved to be no more than $\frac{3}{2}P + 3$ in [8]. However, it is a very loose bound. In our work, we derive a much more accurate estimation $(\frac{13}{24}P + O(1))$. Due to the page limits, we left the proof and simulation in the full version [3].

[2] We use the symbol % as the meaning of mod function in this paper.

4 Multi-Radio Jamming Attack Algorithm

In this section, we design efficient algorithms for the jammer to attack the rendezvous process. Clearly, if the jammer can figure out one user's hopping sequence, it can block the predicted channels to prevent rendezvous between the users. First, we review the existing work of CDJA[14] problem, then we design efficient algorithms for both symmetric and asymmetric situations when the jammer can access n channels. We show that we can improve the efficiency to figure out the hopping sequence by adopting multiple radios.

4.1 The Existing Work

In [14], the attack scheme against the Jump Stay (JS) algorithm[11] under symmetric setting is proposed and the basic idea is to find the start channel and the step length of one user. The key intuition is that if the jammer detects that the user has accessed channel c_1 at time t_1, and c_2 at t_2 (t_1, t_2 are based on the jammer's own clock), where $(t_1 - t_2)\%P \neq 0$, it can calculate the step length r used by this user. This is due to the reason that c_1 and c_2 can be written as:

$$c_1 = (i + (t_1 + \Delta t)r - 1)\%P + 1$$
$$c_2 = (i + (t_2 + \Delta t)r - 1)\%P + 1$$

here Δt is the time difference between the jammer and the user's clocks. We have $((t_2 - t_1)r)\%P = (c_2 - c_1)\%P$ and the r is unique. Formally, see Alg. 1.

Algorithm 1. StepLength

```
1: Input: $t_1, c_1, t_2, c_2$
2: Output: the step length $r$;
3: for $r = 1$ to $P$ do
4:     if $((t_2 - t_1)r)\%P = (c_2 - c_1)\%P$ then
5:         Return r;
6:     end if
7: end for
```

However, the algorithm in [14] restricts that the jammer can choose channels only in $\{P - M + 1, P - M + 2, ...M\}$ since this wouldn't cause any ambiguity. For example in Fig. 1, the jammer shouldn't choose to listen on channel 1, otherwise it cannot tell whether it is the actual channel 1 or it is mapped from channel 5 in the user's original sequence when it detects that the user is accessing channel 1.

In the next subsections we give our attack scheme to the Enhanced Jump Stay Algorithm (EJS) under both symmetric and asymmetric settings. We assume the jammer can access n ($n > 1$) channels simultaneously and our scheme doesn't need the channel selection restriction. The two improvements make the attack more efficient.

Algorithm 2. Multi-Radio Attack Algorithm Under Symmetric Setting

```
1: Input: $M, P, n$;
2: Output: the step length $r$;
3: Choose $n$ random channels $A = \{a_1, a_2...a_n\}$ from $[1, M]$;
4: Keep listening on channels in $A$ until the first signal appears, say it is on channel $b_x$ at time $t_x$;
5: Randomly select another channel $d$ in $[1, M]$ but not in $A$, $A = A \setminus \{b_x\} \bigcup d$;
6: Keep listening on channels in $A$ until the first signal appears, say it is on channel $b_y$ at time $t_y$;
7: if $b_x > P - M$ and $b_y > P - M$ then
8:     Return $StepLength(t_x, b_x, t_y, b_y)$;
9: else if $b_x \le P - M$ and $b_y > P - M$ then
10:     $r_1 = StepLength(t_x, b_x, t_y, b_y)$, $r_2 = StepLength(t_x, b_x + M, t_y, b_y)$;
11:     Check $((b_y + r_1 - 1)\%P)\%M + 1$ and $((b_y + r_2 - 1)\%P)\%M + 1$ at time $t_y + 1$
12:     Return the right step length;
13: else if $b_x > P - M$ and $b_y \le P - M$ then
14:     $r_1 = StepLength(t_x, b_x, t_y, b_y)$, $r_2 = StepLength(t_x, b_x, t_y, b_y + M)$;
15:     Check $((b_y + r_1 - 1)\%P)\%M + 1$ and $((b_y + M + r_2 - 1)\%P)\%M + 1$ at time $t_y + 1$
16:     Return the right step length;
17: else
18:     $r_1 = StepLength(t_x, b_x, t_y, b_y)$, $r_2 = StepLength(t_x, b_x + M, t_y, b_y)$, $r_3 = StepLength(t_x, b_x, t_y, b_y + M)$, $r_4 = StepLength(t_x, b_x + M, t_y, b_y + M)$;
19:     Check the four channels at time $t_y + 1$: $((b_y + r_1 - 1)\%P)\%M + 1$, $((b_y + r_2 - 1)\%P)\%M + 1$, $((b_y + M + r_3 - 1)\%P)\%M + 1$, or $((b_y + M + r_4 - 1)\%P)\%M + 1$, if the jammer doesn't have enough radios, keep on checking at time $t_y + 2$;
20:     Return the right step length;
21: end if
```

4.2 Symmetric Situation

We first give the intuition for finding the step length r: the jammer chooses n channels randomly from $[1, M]$ and keeps listening on these channels, one radio to one channel. Once getting a signal from some channel b_x at time t_x, the jammer randomly chooses another channel replacing b_x and keeps on detecting until the second signal appears on channel b_y at time t_y. If both b_x and b_y are greater than $P - M$, which means they won't cause any ambiguity, the jammer can find r immediately. If one channel is not greater than $P - M$, for example $b_x < P - M$, then the jammer needs to consider two cases: $\{b_x, b_y\}$ and $\{b_x + M, b_y\}$, for either of them the jammer can get a step length. Since the jammer can access more than one channels each time slot, it can check which of the two cases is right in the next time slot. It is similar when b_x and b_y are all not greater than $P - M$, in this case the jammer should check out 4 step lengths in the next 1 or 2 time slots. Formally, please refer to Alg. 2.

In Alg. 2, the input M denotes the number of available channels, P is the smallest prime that $P \ge M$ and n is the number of radios the jammer has. The function *StepLength* refers to Alg. 1. In line 3-6 the jammer gets the two

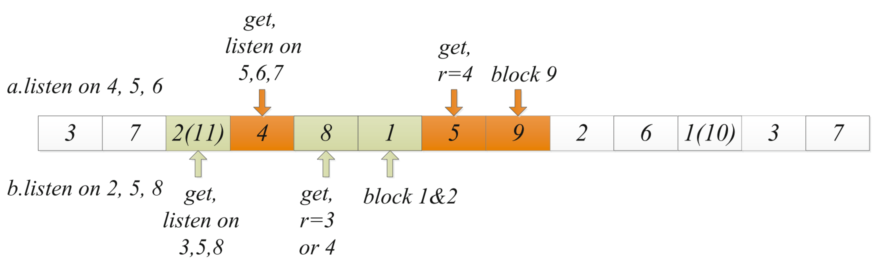


Fig. 2. Example of Alg. 2

signals $(t_x, b_x), (t_y, b_y)$, in line 7-21 the jammer calculates the step length based on different situations of b_x, b_y.

We show two examples of Alg. 2 in Fig. 2. The blocks represent the first jump pattern of the user. Here $M = 9, P = 11$, the user's start channel is 3 and the step length is 4. The jammer has 3 radios.

Case a is shown above the blocks. From the beginning, the jammer chooses channel 4,5,6. When the user chooses channel 4 (the dark yellow block labeled 4) in the 4-th time slot, the jammer gets the signal and changes to listen on channel 5,6,7. In the 7-th time slot a signal on channel 5 appears. Thus the jammer can calculate the step length of the user, it knows the next channel chosen by the user is 9, and so on.

Case b is shown below the blocks. The jammer chooses channel 2,5,8 from the beginning. In the 3-rd time slot, it gets the signal from channel 2 (the light green block labeled 2(11) which means it is mapped from 11), then it changes to channel 3,5,8 until a signal on channel 8 appears in the 5-th time slot. But the jammer cannot get the step length immediately because it doesn't know whether channel 2 is the actual 2 or it is mapped from 11. If it is mapped from 11, the step length should be 4, and the next channel should be 1, otherwise the step length should be 3 and the next channel should be 2. So in the 6-th time slot the jammer listens on channel 1 and 2 and makes sure that 4 is the real step length.

After getting the step length, another important problem we should solve is the start time of the user, because without knowing the start time the jammer cannot decide when will the user enter the stay pattern. From Lemma 1, we can easily derive that each available channel will appear in each jump pattern. If the jammer starts detecting earlier than the user starts, it can get the step length or some alternative step lengths in the first jump pattern of user, i.e. in P time slots after the user starts. So the jammer can confirm the right step length r before the user entering the stay pattern. After the jammer gets r, it allocates one radio staying on channel r. When it gets 3 consecutive signals on channel r, say at time $t, t+1, t+2$, it knows the user has entered the stay pattern, either t, $t+1$ or $t+2$ is the start time of the stay pattern. Thus the jammer can calculate the alternative start time and start channels of the user, when the stay pattern ends and the next round starts, the jammer can make sure which is right.

4.3 Analysis of Alg. 2

In this section, we analyze the expected time and maximum time of Alg. 2. There are two important lemmas as follows.

Lemma 3. *Given $m \geq n \geq 3$, if we sample n numbers from $\{1, 2...m\}$ without repeating, say the numbers are $a_1 < a_2 < a_3... < a_n$, then the expected value of a_2 and a_3 are $E(a_2) = \frac{2(m+1)}{n+1}$,$E(a_3) = \frac{3(m+1)}{n+1}$.*

Due to the page limits, the proof of the lemma can be found in the full version [3].

Remark 2. Actually we can prove that $E(a_i) = \frac{i\times(m+1)}{n+1}$ for all i in $\{1, 2, 3...n\}$.

Lemma 4. *Given $m > n \geq 1$, If we sample $n+1$ numbers from $\{1, 2...m\}$ without repeating, say they are $S = \{a_1, a_2, a_3...a_{n+1}\}$, let $p = min\{a_1, a_2...a_n\}$, $q = min\{x : x \in S \setminus \{p\}, x > p\}$, then $E(q) = \frac{(m+1)(2n+3)}{(n+1)(n+2)}$.*

Proof. We divide the problem into two cases. First, $a_{n+1} < p$, which means a_{n+1} is the smallest in S, so q is the third smallest number in S, from Theroem 3, $E(q) = \frac{3(m+1)}{n+2}$. Second, $a_{n+1} > p$, in this case, q is the second smallest number in S, $E(q) = \frac{2(m+1)}{n+2}$. So we have:

$$\begin{aligned}
E(q) &= E(q|a_{n+1} < p)Pr\{a_{n+1} < p\} + E(q|a_{n+1} > p)Pr\{a_{n+1} > p\} \\
&= \frac{3(m+1)}{n+2}\frac{1}{n+1} + \frac{2(m+1)}{n+2}\frac{n}{n+1} \\
&= \frac{(m+1)(2n+3)}{(n+1)(n+2)}
\end{aligned}$$

According to the two lemmas, we can derive the theorem as:

Theorem 2. *By using Alg. 2 for finding the sequence generated by the user, the maximum time is $P - n + 4$, and the expected time is $\frac{(M+1)(2n+3)}{(n+1)(n+2)} + O(1)$. M is the number of channels, P is the smallest prime that $P \geq M$, n is the number of radios the jammer has.*

Proof. From Section 4.2 we know that Alg. 2 ends in at most 2 time slots after detecting two signals, and all the channels appear in the first jump pattern of the sequence generated by the user. So the worst case is that the jammer chooses n channels which appears at the end of the first jump pattern, thus the maximum time is no more than $P - n + 2 + 2 = P - n + 4$.

Then we prove the expected time, we focus only on the first jump pattern of the sequence because Alg. 2 ends in the first jump pattern. Since $P - M$ is much smaller comparing to P and M, we assume $M = P$, then we know the jump pattern is a permutation of $\{1, 2, ..., P\}$. Notice that even though each channel appears on each position of the permutation with equal probability, the permutation is not a completely random permutation. However, since the

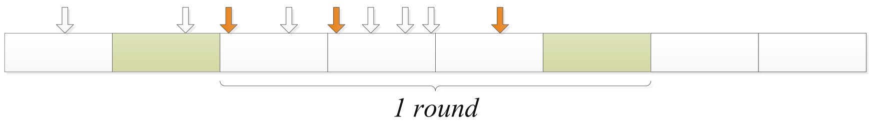


Fig. 3. Example of Alg. 3

jammer chooses n completely random channels to detect, we can just consider the case when the permutation is $1, 2, 3, ..., P$. Suppose the jammer chooses channel set $S = \{a_1, a_2, ..., a_n\}$ to detect, the first signal will appear on channel p where $p = min\{S\}$, then the jammer changes the channel from p to a_{n+1}, now it listens on $S \setminus \{p\} \bigcup \{a_{n+1}\}$, and the second signal should be on channel q where $q = \{x : x = min\{S\}, x > p\}$. We can see that the problem is the same with Lemma 4, so the expected time is $\frac{(M+1)(2n+3)}{(n+1)(n+2)} + O(1)$.

4.4 Asymmetric Situation

Due to the spectrum sensing technique, the user may find some channels occupied by the PUs which means the user cannot access these channels, this is the so-called asymmetric setting. We assume that the jammer knows the available channel set of the user due to the same spectrum sensing technique. In Remark 3 we briefly discuss the situation where the jammer doesn't know the set at all.

Similar to the symmetric setting, the purpose of the jammer is to find the step length and the start time chosen by the user. The main intuition is the same as Alg. 2: the jammer waits on some channels and then calculates the step length along with the start time. The difference is that the jammer may detect some channels that are randomly chosen by the user, this may interfere the jammer and the jammer may get the wrong step length.

Our solution is that the jammer keeps detecting until there are 3 signals, say channel b_1 at time t_1, b_2 at t_2, b_3 at t_3, that satisfy:

$$StepLength(t_1, b_1, t_2, b_2) = StepLength(t_2, b_2, t_3, b_3) = StepLength(t_3, b_3, t_1, b_1)$$

The jammer considers $StepLength(t_1, b_1, t_2, b_2)$ as the step length, and $\{b_1, b_2, b_3\}$ as the channels not randomly chosen by the user. It uses $StepLength(t_1, b_1, t_2, b_2)$ and b_3 for generating the remain sequence. We call $\{b_1, b_2, b_3\}$ "good channels".

Take Fig. 3 as an example, the arrows point to the channels that the jammer gets, the dark yellow arrows point to the "good channels". If they are not chosen randomly by the user and are in the same round as the picture shows, the jammer can get the right step length from the 3 signals. Formally, please refer to Alg. 3. In Alg. 3, the input P, M, n are defined the same way with Alg. 2. Initially, the jammer holds a set B in line 3, when it hears channel b at time t, it adds (t, b) into B. In line 6, the jammer judges all the triplets in B, and if it fails to find the step length, in line 9 it changes the channel from b to another channel and keeps on listening.

Algorithm 3. Multi-Radio Attack Algorithm Under Asymmetric Model

```
1: **Input:** $P, M, n$, the available channel set $C = \{c_1, c_2, ..., c_k\}$ of the user;
2: **Output:** the step length $r$;
3: Set $B = \emptyset$;
4: Sample $n$ channels $A = \{a_1, a_2 ... a_n\}$ without repeating from $C$;
5: Keep listening on channels in $A$ until the first signal appears, say it is on channel
   $b$ at time $t$, $B = B \bigcup \{(t, b)\}$;
6: **if** there exists $(t_x, b_x), (t_y, b_y), (t_z, b_z)$ in $B$ such that $StepLength(t_x, b'_x, t_y, b'_y) =$
   $StepLength(t_x, b'_x, t_z, b'_z) = StepLength(t_y, b'_y, t_z, b'_z)$, here $b'_x = b_x$ or $b_x + M$(only
   when $b_x \leq P - M$), so are $b'_y$ and $b'_z$ **then**
7:     Return $StepLength(t_x, b'_x, t_y, b'_y)$;
8: **else**
9:     Randomly select another channel $d \in C \setminus A$, $A = A \setminus \{b\} \bigcup \{d\}$, goto Line 5.
10: **end if**
```

Notice that "good channels" are not always good. Sometime three channels not in the same round or containing random channels can also satisfy the above equation. This means our algorithm cannot guarantee finding the right step length. We discuss why our algorithm still works well in the next subsection.

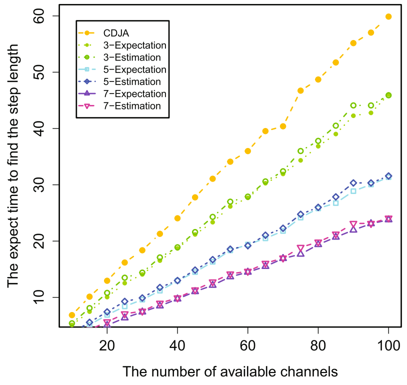
When the jammer gets the step length r, another important problem is the start time and we tackle it similarly as the symmetric situation, where the jammer allocates one radio on channel r to find the time of entering the stay pattern, and then it can know the start time of the user.

Remark 3. When the jammer doesn't know the available channel set of the user, it can still use Alg. 3 to make the attack, by just choosing n random channels to listen. However, this makes the algorithm slower, and we leave more study of this situation to the future work.

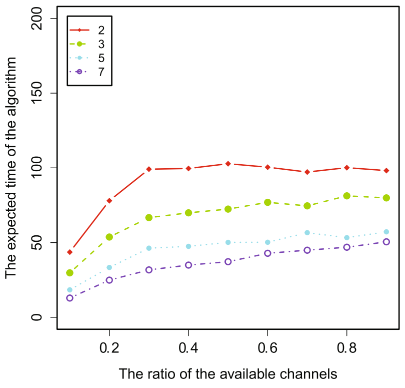
4.5 Analysis of Alg. 3

In this section we discuss why Alg. 3 works well even though sometime "good channels" may not be "good". Suppose the number of all channels is M and the number of available channels for the user is λM, P is the smallest prime no less than M, the jammer has n radios and starts detecting earlier than the user starts. In the jump patterns of the first round which last for $3P$ time slots, there are about $3P\lambda$ channels which are not chosen randomly. For each of these channels, the probability of being detected by the jammer is about $n/(P\lambda)$, so on average, the jammer can detect about $3P\lambda \times n/(P\lambda) = 3n$ channels which are not random channels in the jump patterns of the first round. Thus we can expect that Alg. 3 ends in the first round with great probability, which means there is great chance that the jammer can find really good "good channels" in the first round. Our simulation results in the full version [3] shows that, the probability of successful attack in the first round is more than 80% when the ratio of available channels is more than 40%.

A good way to make the calculation of the step length more accurate is to run Alg. 3 multiple times. In each time, the jammer can get a step length r and



(a) Expectation and estimation of the time of Alg. 2



(b) The expected time of running Alg. 3 once

Fig. 4. Simulation Results

it is easy to see that the real step length appears more times than the other false r. Notice that under the asymmetric setting, the rendezvous time is no longer within $4P$ time slots, so the jammer can try to run Alg. 3 several times to get an accurate r. Moreover, our simulation result in Section 5 shows that the average time for Alg. 3 is always within P time slots.

5 Performance Evaluation

In this section we conduct extensive simulations to evaluate our algorithms. We use R language to implement the simulation, in each experiment we get the result as the average of more than 20000 separate runs. Due to page limits, we only show part of the simulations results, you can refer to the full version [3] for all the simulation results.

Fig. 4(a). shows the expected time of Alg. 2. The x-bar represents different numbers of available channels, while the y-bar represents the expected time. We do experiment in case the jammer can access 3, 5 or 7 channels. We also use the CDJA[14] to attack the EJS users. From the figure, we can see that our result is better than CDJA. In each case our estimation which is based on Theorem. 2 is also shown in the figure, they are almost the same as the experiment result. The gap is no more than 2.

Fig. 4(b) shows the expected time of Alg. 3. Here we assume there are totally 100 channels and the jammer starts detecting earlier than both users, the x-bar represents the ratio of available channels of the user. The y-bar represents the expected time. We can see that there is no much difference when the ratio changes, but the number of radios the jammer has is very important. However, even if the jammer can detect only 2 channels each time slot, the expected time is around 100, this means in expectation Alg. 3 can end in the 1*st* pattern of the 1*st* round. It coincides with our analysis in Section 4.5.

6 Conclusion

In this paper, we propose the Multi-Radio Channel Detecting Jamming Attack (MRCDJA) problem where the jammer can listen or block $n \geq 2$ channels simultaneously to prevent rendezvous between the users. We assume the users adopting the Enhanced Jump Stay (EJS)[8] algorithm for rendezvous attempt and we design efficient algorithms to determine the users' channel hopping sequence. For the symmetric users that can use all channels in the channel set, our algorithm is n times faster than the current methods. For asymmetric users, our algorithm can also work well. Finally we conduct extensive simulations for evaluation. In the future, we are to explore efficient attack algorithms when the users are also equipped with multiple radios for rendezvous attempt.

References

1. Draves, R., Pahye, J., Zill, B.: Routing in multi-radio, multi-hop wireless mesh networks. In: MobiCom (2004)
2. Gu, Z., Hua, Q.-S., Wang, Y., Lau, F.C.M.: Nearly Optimal Asynchronous Blind Rendezvous Algorithm for Cognitive Radio Networks. In: SECON (2013)
3. Gao, Y., Gu, Z., Hua, Q.-S., Jin, H.: Multi-Radio Channel Detecting Jamming Attack Against Enhanced Jump-Stay Based Rendezvous in Cognitive Radio Networks. http://grid.hust.edu.cn/qshua/cocoon15_full.pdf
4. Gu, Z., Hua, Q.-S., Dai, W.: Local Sequence Based Rendezvous Algorithms for Cognitive Radio Networks. In: SECON (2014)
5. Gu, Z., Hua, Q.-S., Wang, Y., Lau, F.C.M.: Oblivious Rendezvous in Cognitive Radio Networks. In: Halldórsson, M.M. (ed.) SIROCCO 2014. LNCS, vol. 8576, pp. 165–179. Springer, Heidelberg (2014)
6. Gu, Z., Hua, Q.-S., Dai, W.: Fully Distributed Algorithms for Blind Rendezvous in Cognitive Radio Networks. In: MOBIHOC (2014)
7. Li, G., Gu, Z., Lin, X., Pu, H., Hua, Q.-S.: Deterministic Distributed Rendezvous Algorithms for Multi-Radio Cognitive Radio Networks. In: MSWiM (2014)
8. Lin, Z., Liu, H., Chu, X., Leung, Y.-W.: Enhanced Jump-Stay Rendezvous Algorithm for Cognitive Radio Networks. IEEE Communications Letters (2013)
9. Liu, H., Lin, Z., Chu, X., Leung, Y.-W.: Jump-Stay Rendezvous Algorithm for Cognitive Radio Networks. IEEE Transactions on Parallel and Distributed Systems **23**(10), 1867–1881 (2012)
10. Liu, H., Lin, Z., Chu, X., Leung, Y.-W.: Taxonomy and Challenges of Rendezvous Algorithms in Cognitive Radio Networks. In: ICNC (2012)
11. Liu, H., Lin, Z., Chu, X., Leung, Y.-W.: Jump-Stay Based Channel-Hopping Algorithm with Guaranteed Rendezvous for Cognitive Radio Networks. In: INFOCOM (2011)
12. Paul, R., Jembre, Y.-Z., Choi, Y.-J.: Multi-interface rendezvous in self-organizing cognitive radio networks. In: DySPAN (2014)
13. Yang, D., Shin, J., Kim, C.: Deterministic Rendezvous Scheme in Multichannel Access Networks. Electronics Letters (2010)
14. Oh, Y.-H., Thuente, D.-J.: Channel Detecting Jamming Attacks Against Jump-Stay Based Channel Hopping Rendezvous Algorithm for Cognitive Radio Networks. In: ICCCN (2013)
15. Zhao, Q., Sadler, B.-M.: A Survey of Dynamic Spectrum Acess (signalprocessing, networking and regulatory policy). IEEE Signal Processing Magazine (2007)

Upper Bounds on Fourier Entropy

Sourav Chakraborty[1], Raghav Kulkarni[2], Satyanarayana V. Lokam[3], and Nitin Saurabh[4](✉)

[1] Chennai Mathematical Institute, Chennai, India
sourav@cmi.ac.in
[2] Centre for Quantum Technologies, Singapore, Singapore
kulraghav@gmail.com
[3] Microsoft Research India, Bangalore, India
satya@microsoft.com
[4] The Institute of Mathematical Sciences, Chennai, India
nitin@imsc.res.in

Abstract. Given a function $f : \{0,1\}^n \to \mathbb{R}$, its *Fourier Entropy* is defined to be $-\sum_S \widehat{f}^2(S) \log \widehat{f}^2(S)$, where $\hat{f}$ denotes the Fourier transform of f. This quantity arises in a number of applications, especially in the study of Boolean functions. An outstanding open question is a conjecture of Friedgut and Kalai (1996), called the Fourier Entropy Influence (FEI) Conjecture, asserting that the Fourier Entropy of any Boolean function f is bounded above, up to a constant factor, by the total influence (= average sensitivity) of f.

In this paper we give several upper bounds on the Fourier Entropy of Boolean as well as real valued functions. We first give upper bounds on the Fourier Entropy of Boolean functions in terms of several complexity measures that are known to be bigger than the influence. These complexity measures include, among others, the logarithm of the number of leaves and the average depth of a parity decision tree. We then show that for the class of Linear Threshold Functions (LTF), the Fourier Entropy is at most $O(\sqrt{n})$. It is known that the average sensitivity for the class of LTF is bounded by $\Theta(\sqrt{n})$. We also establish a bound of $O_d(n^{1-\frac{1}{4d+6}})$ for general degree-d polynomial threshold functions. Our proof is based on a new upper bound on the *derivative of noise sensitivity*. Next we proceed to show that the FEI Conjecture holds for read-once formulas that use AND, OR, XOR, and NOT gates. The last result is independent of a recent result due to O'Donnell and Tan [14] for read-once formulas with *arbitrary* gates of bounded fan-in, but our proof is completely elementary and very different from theirs. Finally, we give a general bound involving the first and second moments of sensitivities of a function (average sensitivity being the first moment), which holds for real valued functions as well.

1 Introduction

Fourier transforms are extensively used in a number of fields such as engineering, mathematics, and computer science. Within theoretical computer science,

D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 771–782, 2015.
DOI: 10.1007/978-3-319-21398-9_60

Fourier analysis of Boolean functions evolved into one of the most useful and versatile tools; see the book [13] for a comprehensive survey of this area and pointers to numerous other sources of literature on this subject. In particular, it plays an important role in several results in complexity theory, learning theory, social choice, inapproximability, metric spaces, etc. If $\hat{f}$ denotes the Fourier transform of a Boolean function f, then $\sum_{S\subseteq[n]} \widehat{f}^2(S) = 1$ and hence we can define an entropy of the distribution given by $\widehat{f}^2(S)$:

$$\mathbb{H}(f) := \sum_{S\subseteq[n]} \widehat{f}^2(S) \log \frac{1}{\widehat{f}^2(S)} \quad . \tag{1}$$

The Fourier Entropy-Influence (FEI) Conjecture, made by Friedgut and Kalai [5] in 1996, states that for every Boolean function, its Fourier entropy is bounded above by its total influence :

Fourier Entropy-Influence Conjecture. There exists a universal constant C such that for all $f : \{0,1\}^n \to \{+1,-1\}$,

$$\mathbb{H}(f) \leq C \cdot \mathsf{Inf}(f) \ , \tag{2}$$

where $\mathsf{Inf}(f)$ is the total influence of f which is the same as the average sensitivity $\mathsf{as}(f)$ of f. The latter quantity may be intuitively viewed as the expected number of coordinates of an input which, when flipped, will cause the value of f to be changed, where the expectation is w.r.t. the uniform distribution on the input assignments of f.

1.1 Motivation

Resolving the FEI conjecture is one of the most important open problems in the Fourier analysis of Boolean functions. The conjecture intuitively asserts that if the Fourier coefficients of a Boolean function are "smeared out," then its influence must be large, i.e., at a typical input, the value of f changes in several different directions. The original motivation for the conjecture in [5] stems from a study of threshold phenomena in random graphs.

The FEI conjecture has numerous applications. It implies a variant of *Mansour's Conjecture* [11] stating that for a Boolean function computable by a DNF formula with m terms, most of its Fourier mass is concentrated on $\mathsf{poly}(m)$-many coefficients. A proof of Mansour's conjecture would imply a polynomial time *agnostic* learning algorithm for DNF's [6] answering a major open question in computational learning theory.

The FEI conjecture also implies that for any n-vertex graph property, the influence is at least $c(\log n)^2$. The best known lower bound, by Bourgain and Kalai [1], is $\Omega((\log n)^{2-\epsilon})$, for any $\epsilon > 0$. See [9], [15] and [10] for a detailed explanation on these and other consequences of the conjecture.

1.2 Prior Work

The first progress on the FEI conjecture was made in 2010 in [10] showing that the conjecture holds for random DNFs. O'Donnell et al. [15] proved that the conjecture holds for symmetric functions and more generally for any d-part symmetric functions for constant d. They also established the conjecture for functions computable by read-once decision trees. Keller et al. [9] studied a generalization of the conjecture to biased product measures on the Boolean cube and proved a variant of the conjecture for function with extremely low Fourier weight on the high levels. O'Donnell and Tan [14] verified the conjecture for read-once formulas using a composition theorem for the FEI conjecture. Wan et al. [16] studies the conjecture from the point of view of existence of efficient prefix-free codes for the random variable, $\mathcal{X} \sim \hat{f}^2$, that is distributed according to $\hat{f}^2$. Using this interpretation they verify the conjecture for bounded read decision trees. It is also relatively easy to show that the FEI conjecture holds for a random Boolean function, e.g., see [3] for a proof. By direct calculation, one can verify the conjecture for simple functions like AND, OR, $\mathsf{Majority}$, Tribes etc.

1.3 Our Results

We report here various upper bounds on Fourier entropy that may be viewed as progress toward the FEI conjecture.

Upper bounds by Complexity Measures. The $\mathsf{Inf}(f)$ of a Boolean function f is used to derive lower bounds on a number of complexity parameters of f such as the number of leaves or the average depth of a decision tree computing f. Hence a natural weakening of the FEI conjecture is to prove upper bounds on the Fourier entropy in terms of such complexity measures of Boolean functions. By a relatively easy argument, we show that

$$\mathbb{H}(f) = O(\log \mathsf{L}(f)), \tag{3}$$

where $\mathsf{L}(f)$ denotes the minimum number of leaves in a decision tree that computes f. If $\mathsf{DNF}(f)$ denotes the minimum size of a DNF for the function f, note that $\mathsf{DNF}(f) \leq \mathsf{L}(f)$. Thus improving (3) with $O(\log \mathsf{DNF}(f))$ on the right hand side would resolve *Mansour's conjecture* – a long-standing open question about sparse Fourier approximations to DNF formulas motivated by applications to learning theory – and a special case of the FEI conjecture for DNF's. We note that (3) also holds when the queries made by the decision tree involve parities of subsets of variables, conjunctions of variables, etc. It also holds when $\mathsf{L}(f)$ is generalized to the number of subcubes in a *subcube partition* that represents f. Note that for a Boolean function

$$\mathsf{Inf}(f) \leq \log(L_c(f)) \leq \log(\mathsf{L}(f)) \leq D(f),$$

where $L_c(f)$ is number of subcubes in a *subcube partition* that represents f and $D(f)$ is the depth of the decision tree computing f.

We also prove the following strengthening of (3):

$$\mathbb{H}(f) = O(\bar{\mathsf{d}}(f)), \tag{4}$$

where $\bar{\mathsf{d}}(f)$ denotes the *average depth* of a decision tree computing f (observe that $\bar{\mathsf{d}}(f) \leq \log(\mathsf{L}(f))$). Note that the average depth of a decision tree is also a kind of entropy: it is given by the distribution induced on the leaves of a decision tree when an input is drawn uniformly at random. Thus (4) relates the two kinds of entropy, but only up to a constant factor. We further strengthen (4) by improving the right-hand side in (4) to *average depth* of a *parity* decision tree computing f, that is, queries made by the decision tree are parities of a subset of variables.

Upper bounds on the Fourier Entropy of Polynomial Threshold Functions. The Fourier Entropy-Influence conjecture is known to be true for unweighted threshold functions, i.e., when $f(x) = \mathsf{sign}(x_1 + \cdots + x_n - \theta)$ for some integer $\theta \in [0..n]$. This follows as a corollary of the result due to O'Donnell et al. [15] that the FEI conjecture holds for all symmetric Boolean functions. It is known that the influence for the class of linear threshold functions is bounded by $\Theta(\sqrt{n})$ (where the lower bound is witnessed by Majority [12]). Recently Harsha et al. [7] studied average sensitivity of polynomial threshold function (see also [4]). They proved that average sensitivity of degree-d polynomial threshold functions is bounded by $O_d(n^{1-(1/4d+6)})$, where $O_d(\cdot)$ denotes that the constant depends on degree d. This suggests a natural and important weakening of the FEI conjecture: *Is Fourier Entropy of polynomial threshold functions bounded by a similar function of n as their average sensitivity?* In this paper we answer this question in the positive. An important ingredient in our proof is a bound on the derivative of noise sensitivity in terms of the noise parameter.

FEI inequality for Read-Once Formulas. We also prove that the FEI conjecture holds for a special class of Boolean functions: Read-Once Formulas over {AND, OR and XOR}, i.e., functions computable by a tree with AND, OR and XOR gates at internal nodes and each variable (or its negation) occurring *at most once* at the leaves. Our result is independent of a very recent result by O'Donnell and Tan [14] that proves the FEI conjecture holds for read-once formulas that allow *arbitrary* gates of bounded fan-in. However, our proof is completely elementary and very different from theirs. Prior to these results, O'Donnell et al. [15] proved that the FEI conjecture holds for read-once *decision trees*. Our result for read-once formulas is a strict generalization of their result. For instance, the tribes function is computable by read-once formulas but not by read-once decision trees. Our proof for read-once formulas is a consequence of a kind of tensorizability for $\{0, 1\}$-valued Boolean functions. In particular, we show that an inequality similar to the FEI inequality is preserved when functions depending on *disjoint* sets of variables are combined by AND, OR and XOR operators.

A Bound for Real valued Functions via Second Moment. Recall [8] that total influence $\mathsf{Inf}(f)$ or average sensitivity $\mathsf{as}(f)$ is related to $\hat{f}$ by the well-known identity: $\mathsf{as}(f) = \mathsf{Inf}(f) = \sum_S |S| \, \widehat{f}^2(S)$. Hence, an equivalent way to state the FEI conjecture is that there is an absolute constant C such that for all Boolean f,

$$\mathbb{H}(f) \leq C \cdot \sum_S |S| \, \widehat{f}^2(S) \ . \tag{5}$$

Here, we prove that for all δ, $0 \leq \delta \leq 1$, and for all f with $\sum_S \widehat{f}^2(S) = 1$, and hence for Boolean f in particular,

$$\mathbb{H}(f) \leq \sum_S |S|^{1+\delta} \widehat{f}^2(S) \ + \ (\log n)^{O(1/\delta)} \ . \tag{6}$$

An alternative interpretation of the above theorem states

$$\mathbb{H}(f) \leq \mathsf{as}(f)^{1-\delta} \cdot \mathsf{as}_2(f)^{\delta} \ + \ (\log n)^{O(1/\delta)} \ , \tag{7}$$

where $\mathsf{as}_2(f) := \sum_S |S|^2 \, \widehat{f}^2(S)$. We also mention that $\mathsf{as}_2(f) \leq \mathsf{s}(f)^2$ (see [2]), where $\mathsf{s}(f)$ is the *maximum* sensitivity of f.

It is important to note that (6) holds for *arbitrary*, i.e., even non-Boolean, f such that (without loss of generality) $\sum_S \widehat{f}^2(S) = 1$. On the other hand, there are examples of non-Boolean f for which the FEI conjecture (5) is *false*. Combining (7) with a "tensorizability" property [15] of $\mathbb{H}(f)$ and $\mathsf{as}(f)$, it is possible to show that for *all* f, $\mathbb{H}(f) = O(\mathsf{as}(f) \log n)$. Hence proving the FEI conjecture should involve removing the "extra" log factor while exploiting the Boolean nature of f.

Remainder of the paper. We give basic definitions in Section 2. Section 3 contains upper bounds in terms of complexity measures. In Section 4 and Section 5 we consider special classes of Boolean functions namely, the *polynomial threshold functions* and *Read-Once formulas.* We then provide bounds for real valued functions in Section 6. Due to space limitations, proofs had to be omitted from this extended abstract. For a more complete version, please see [2].

2 Preliminaries

We recall here some basic facts of Fourier analysis. For a detailed treatment please refer to [13,17]. Consider the space of all functions from $\{0,1\}^n$ to $\mathbb{R}$, endowed with the inner product $\langle f, g \rangle = 2^{-n} \sum_{x \in \{0,1\}^n} f(x) g(x)$. The character functions $\chi_S(x) := (-1)^{\sum_{i \in S} x_i}$ for $S \subseteq [n]$ form an orthonormal basis for this space of functions w.r.t. the above inner product. Thus, every function $f : \{0,1\}^n \longrightarrow \mathbb{C}$ of n Boolean variables has the *unique Fourier* expansion: $f(x) = \sum_{S \subseteq [n]} \hat{f}(S) \chi_S(x)$. The vector $\hat{f} = (\hat{f}(S))_{S \subseteq [n]}$ is called the Fourier transform of the function f. The Fourier coefficient $\hat{f}(S)$ of f at S is then given by, $\hat{f}(S) = 2^{-n} \sum_{x \in \{0,1\}^n} f(x)\chi_S(x)$. The norm of a function f is defined to be $\|f\| = \sqrt{\langle f, f \rangle}$. Orthonormality of $\{\chi_S\}$ implies *Parseval's identity*: $\|f\|^2 = \sum_S \widehat{f}^2(S)$.

We only consider *finite probability distributions* in this paper. The *entropy* of a distribution $\mathcal{D}$ is given by, $\mathbb{H}(\mathcal{D}) := \sum_{i \in \mathcal{D}} p_i \log \frac{1}{p_i}$. In particular, the *binary* entropy function, denoted by $\mathrm{H}(p)$, equals $-p \log p - (1-p)\log(1-p)$. All logarithms in the paper are base 2, unless otherwise stated.

We consider Boolean functions with range $\{-1,+1\}$. For an $f : \{0,1\}^n \to \{-1,+1\}$, $\|f\|$ is clearly 1 and hence Parseval's identity shows that for Boolean functions $\sum_S \widehat{f}^2(S) = 1$. This implies that the squared Fourier coefficients can be thought of as a probability distribution and the notion of Fourier Entropy (1) is well-defined.

The *influence of f in the i-th direction*, denoted $\mathsf{Inf}_i(f)$, is the fraction of inputs at which the value of f gets flipped if we flip the i-th bit:

$$\mathsf{Inf}_i(f) = 2^{-n}|\{x \in \{0,1\}^n : f(x) \neq f(x \oplus e_i)\}| \ ,$$

where $x \oplus e_i$ is obtained from x by flipping the i-th bit of x.

The (total) *influence* of f, denoted by $\mathsf{Inf}(f)$, is $\sum_{i=1}^n \mathsf{Inf}_i(f)$. The influence of i on f can be shown, e.g., [8], to be $\mathsf{Inf}_i(f) = \sum_{S \ni i} \hat{f}(S)^2$ and hence it follows that $\mathsf{Inf}(f) = \sum_{S \subseteq [n]} |S| \hat{f}(S)^2$.

For $x \in \{0,1\}^n$, the *sensitivity of f at x*, denoted $\mathsf{s}_f(x)$, is defined to be $\mathsf{s}_f(x) := |\{i : f(x) \neq f(x \oplus e_i), 1 \leq i \leq n\}|$, i.e., the number of coordinates of x, which when flipped, will flip the value of f. The (maximum) *sensitivity of the function f*, denoted $\mathsf{s}(f)$, is defined to be the largest sensitivity of f at x over all $x \in \{0,1\}^n$: $\mathsf{s}(f) := \max\{\mathsf{s}_f(x) : x \in \{0,1\}^n\}$. The *average sensitivity of f*, denoted $\mathsf{as}(f)$, is defined to be $\mathsf{as}(f) := 2^{-n} \sum_{x \in \{0,1\}^n} s_f(x)$. It is easy to see that $\mathsf{Inf}(f) = \mathsf{as}(f)$ and hence we also have $\mathsf{as}(f) = \sum_{S \subseteq [n]} |S| \hat{f}(S)^2$.

The noise sensitivity of f at ϵ, $0 \leq \epsilon \leq 1$, denoted $\mathsf{NS}_\epsilon(f)$, is given by $\Pr_{x, y \sim_\epsilon x}[f(x) \neq f(y)]$ where $y \sim_\epsilon x$ denotes that y is obtained by flipping each bit of x independently with probability ϵ. It is easy to see that $\mathsf{NS}_\epsilon(f) = \frac{1}{2} - \frac{1}{2}\sum_S (1-2\epsilon)^{|S|} \hat{f}(S)^2$. Hence the derivative of $\mathsf{NS}_\epsilon(f)$ with respect to ϵ, denoted $\mathsf{NS}_\epsilon{}'(f)$, equals $\sum_{S \neq \emptyset} |S|(1-2\epsilon)^{|S|-1} \hat{f}(S)^2$.

3 Bounding Entropy Using Complexity Measures

In this section, we prove upper bounds on Fourier entropy in terms of some complexity parameters associated to decision trees and subcube partitions.

3.1 *via* Leaf Entropy : Average Decision Tree Depth

Let T be a decision tree computing $f : \{0,1\}^n \to \{+1,-1\}$ on variable set $X = \{x_1, \ldots, x_n\}$. If $A_1, \ldots, A_L$ are the sets (with repetitions) of variables queried along the root-to-leaf paths in the tree T, then the average depth (w.r.t. the uniform distribution on inputs) of T is defined to be $\bar{d} := \sum_{i=1}^{L} |A_i| 2^{-|A_i|}$. Note that the average depth of a decision tree is also a kind of entropy: if each leaf λ_i is chosen with the probability $p_i = 2^{-|A_i|}$ that a uniformly chosen random input reaches it, then the entropy of the distribution induced on the λ_i is $\mathbb{H}(\lambda_i) = -\sum_i p_i \log p_i = \sum_i |A_i| 2^{-|A_i|}$. Here, we will show that *the Fourier entropy is at most twice the leaf entropy of a decision tree.*

W.l.o.g., let x_1 be the variable queried by the root node of T and let T_1 and T_2 be the subtrees reached by the branches $x_1 = +1$ and $x_1 = -1$ respectively and let g_1 and g_2 be the corresponding functions computed on variable set $Y = X \setminus \{x_1\}$. Let $\bar{d}$ be the average depth of T and $\bar{d}_1$ and $\bar{d}_2$ be the average depths of T_1 and T_2 respectively. We first observe a fairly straightforward lemma relating Fourier coefficients of f to the Fourier coefficients of restrictions of f.

Lemma 1. *Let $S \subseteq \{2, \ldots, n\}$.*

(i) $\widehat{f}(S) = (\widehat{g_1}(S) + \widehat{g_2}(S))/2.$
(ii) $\widehat{f}(S \cup \{1\}) = (\widehat{g_1}(S) - \widehat{g_2}(S))/2.$
(iii) $\bar{d} = (\bar{d}_1 + \bar{d}_2)/2 + 1.$

Using Lemma 1 and concavity of entropy we establish the following technical lemma, which relates the entropy of f to entropies of restrictions of f.

Lemma 2. *Let g_1 and g_2 be defined as before in Lemma 1. Then,*

$$\mathbb{H}(f) \leq \frac{1}{2}\mathbb{H}(g_1) + \frac{1}{2}\mathbb{H}(g_2) + 2 \ . \tag{8}$$

Let $\bar{\mathsf{d}}(f)$ denote the minimum *average* depth of a decision tree computing f. As a consequence of Lemma 2 we obtain the following bound.

Theorem 1. *For every Boolean function f, $\mathbb{H}(f) \leq 2 \cdot \bar{\mathsf{d}}(f)$.*

Remark 1. The constant 2 in the bound of Theorem 1 cannot be replaced by 1. Indeed, let $f(x, y) = x_1 y_1 + \cdots + x_{n/2} y_{n/2} \bmod 2$ be the inner product mod 2 function. Then because $\widehat{f}^2(S) = 2^{-n}$ for all $S \subseteq [n]$, $\mathbb{H}(f) = n$. On the other hand, it can be shown that $\bar{\mathsf{d}}(f) = \frac{3}{4}n - o(n)$. Hence, the constant must be at least 4/3.

Average Parity Decision Tree Depth. Let L be a linear transformation. Applying the linear transformation on a Boolean function f we obtain another Boolean function Lf which is defined as $Lf(x) := f(Lx)$, for all $x \in \{0,1\}^n$. Before proceeding further, we note down a useful observation.

Proposition 2. *Let $f : \{0,1\}^n \to \{+1,-1\}$ be a Boolean function. For an invertible linear transformation $L \in \mathsf{GL}_n(\mathbb{F}_2)$, $\mathbb{H}(f) = \mathbb{H}(Lf)$.*

Let T be a parity decision tree computing $f : \{0,1\}^n \to \{+1,-1\}$ on variable set $X = \{x_1, \dots, x_n\}$. Note that a parity decision tree computing f also computes Lf and vice versa. This implies that we can always ensure that a variable is queried at the root node of T via a linear transformation. Let us denote the new variable set, after applying the linear transformation, by $Y = \{y_1, \dots, y_n\}$. W.l.o.g, let y_1 be the variable queried at the root. Let T_1 and T_2 be the subtrees reached by the branches $y_1 = 0$ and $y_1 = 1$ respectively and let g_1 and g_2 be the corresponding functions computed on variable set $Y \setminus \{y_1\}$. Using Proposition 2 we see that the proof of Lemma 1 and Lemma 2 goes through in the setting of parity decision trees too. Hence, we get the following strengthening of Theorem 1.

Theorem 3. *For every Boolean function f, $\mathbb{H}(f) \le 2 \cdot \oplus\text{-}\bar{\mathsf{d}}(f)$, where $\oplus\text{-}\bar{\mathsf{d}}(f)$ denotes the minimum average depth of a parity decision tree computing f.*

3.2 *via* L_1-norm (or Concentration) : Decision Trees and Subcube Partitions

Note that a decision tree computing a Boolean function f induces a partition of the cube $\{0,1\}^n$ into monochromatic subcubes, i.e., f has the same value on all points in a given subcube, with one subcube corresponding to each leaf. But there exist monochromatic subcube partitions that are not induced by any decision tree. Consider any subcube partition $\mathcal{C}$ computing f (see [2]). There is a natural way to associate a probability distribution with $\mathcal{C}$: C_i has probability mass $2^{-(\text{number of co-ordinates fixed by } C_i)}$. Let us call the entropy associated with this probability distribution *partition entropy.* Based on the results of the previous subsection, a natural direction would be to prove that the Fourier entropy is bounded by the partition entropy. Unfortunately we were not quite able to show that but, interestingly, there is a very simple proof to see that the Fourier entropy is bounded by the logarithm of the number of partitions in $\mathcal{C}$. In fact, the proof gives a slightly better upper bound of the logarithm of the spectral-norm of f. For completeness sake, we note this observation [2] but we remark that it should be considered folklore. Our goal in presenting the generalization to subcube partitions is also to illustrate a different approach. The approach uses the concentration property of the Fourier transform and uses a general, potentially powerful, technique. One way to do this is to use a result due to Bourgain and Kalai (Theorem 3.2 in [9]). However, we give a more direct proof (see [2]) for the special case of subcube partitions.

4 Upper Bound on Fourier Entropy of Threshold Functions

In this section, we establish a better upper bound on the Fourier entropy of polynomial threshold functions. We show that the Fourier entropy of a linear threshold function is bounded by $O(\sqrt{n})$, and for a degree-d threshold function it is bounded by $O_d(n^{1-\frac{1}{4d+6}})$. We remark that the bound is significant because the average sensitivity of a linear threshold function on n variables is bounded by $O(\sqrt{n})$, and this is tight. Also the bound on the Fourier entropy of degree-d threshold functions is the best known bound on their average sensitivity [4,7].

For $f : \{0,1\}^n \rightarrow \{+1,-1\}$, let $W^k[f] := \sum_{|S|=k} \hat{f}(S)^2$ and $W^{\geq k}[f] := \sum_{|S|\geq k} \hat{f}(S)^2$. We now state our main technical lemma which translates a bound on noise sensitivity to a bound on the derivative of noise sensitivity.

Lemma 3. *Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be such that $\mathsf{NS}_\epsilon(f) \leq \alpha \cdot \epsilon^\beta$, where α is independent of ϵ and $\beta < 1$. Then, $\mathsf{NS}_\epsilon{}'(f) \leq \frac{3}{1-e^{-2}} \cdot \frac{\alpha}{1-\beta} \cdot (1/\epsilon)^{1-\beta}$.*

From [15] we have the following bound on entropy.

Lemma 4. *[15] Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be a Boolean function. Then, $\mathbb{H}(f) \leq \frac{1}{\ln 2}\mathsf{Inf}[f] + \frac{1}{\ln 2}\sum_{k=1}^n W^k[f] k \ln \frac{n}{k} + 3 \cdot \mathsf{Inf}[f]$.*

Using Lemma 3 we prove a technical lemma that provides a bound on $\sum_{k=1}^n W^k[f] k \ln \frac{n}{k}$.

Lemma 5. *Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be a Boolean function. Then, $\sum_{k=1}^n W^k[f] k \ln \frac{n}{k} \leq exp(1/2) \cdot \frac{3}{1-e^{-2}} \cdot \frac{4^{1-\beta}}{(1-\beta)^2} \cdot \alpha \cdot n^{1-\beta}$.*

Using Lemma 5 and Lemma 4 we obtain the following theorem which bounds the Fourier entropy of a Boolean function.

Theorem 4. *Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be a Boolean function such that $\mathsf{NS}_\epsilon(f) \leq \alpha \cdot \epsilon^\beta$. Then*

$$\mathbb{H}(f) \leq C \cdot \left(\mathsf{Inf}[f] + \frac{4^{1-\beta}}{(1-\beta)^2} \cdot \alpha \cdot n^{1-\beta} \right) ,$$

where C is a universal constant.

In particular, for polynomial threshold functions there exist non-trivial bounds on their noise sensitivity.

Theorem 5 (Peres's Theorem). *[12] Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be a linear threshold function. Then $\mathsf{NS}_\epsilon(f) \leq O(\sqrt{\epsilon})$.*

Theorem 6. *[7] For any degree-d polynomial threshold function $f : \{0,1\}^n \rightarrow \{+1,-1\}$ and $0 < \epsilon < 1$, $\mathsf{NS}_\epsilon(f) \leq 2^{O(d)} \cdot \epsilon^{1/(4d+6)}$.*

As corollaries of Theorem 4, using Theorem 5 and Theorem 6, we obtain the following bounds on the Fourier entropy of polynomial threshold functions.

Corollary 1. *Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be a linear threshold function. Then, $\mathbb{H}(f) \leq C \cdot \sqrt{n}$, where C is a universal constant.*

Corollary 2. *Let $f : \{0,1\}^n \rightarrow \{+1,-1\}$ be a degree-d polynomial threshold function. Then, $\mathbb{H}(f) \leq C \cdot 2^{O(d)} \cdot n^{1-\frac{1}{4d+6}}$, where C is a universal constant.*

5 Entropy-Influence Inequality for Read-Once Formulas

In this section, we will prove the Fourier Entropy-Influence conjecture for read-once formulas using AND, OR, XOR, and NOT gates. We note that a recent (and independent) result of O'Donnell and Tan [14] proves the conjecture for read-once formulas with *arbitrary* gates of bounded fan-in. But since our proof is completely elementary and very different from theirs, we choose to present it here.

It is well-known that both Fourier entropy and average sensitivity add up when two functions on disjoint sets of variables are added modulo 2. Our main result here is to show that somewhat analogous "tensorizability" properties hold when composing functions on disjoint sets of variables using AND and OR operations.

For $f : \{0,1\}^n \to \{+1,-1\}$, let $f_{\mathbb{B}}$ denote its 0-1 counterpart: $f_{\mathbb{B}} \equiv \frac{1-f}{2}$.

$$\text{Let's define:} \qquad \mathbf{H}(f_{\mathbb{B}}) := \sum_S \widehat{f_{\mathbb{B}}}^2(S) \log \frac{1}{\widehat{f_{\mathbb{B}}}^2(S)}. \tag{9}$$

An easy relation enables translation between $\mathbb{H}(f)$ and $\mathbf{H}(f_{\mathbb{B}})$:

Lemma 6. *Let* $p = \Pr[f_{\mathbb{B}} = 1] = \widehat{f_{\mathbb{B}}}(\emptyset) = \sum_S \widehat{f_{\mathbb{B}}}^2(S)$ *and* $q := 1 - p$. *Then,*

$$\mathbb{H}(f) = 4 \cdot \mathbf{H}(f_{\mathbb{B}}) + \varphi(p), \quad \textit{where} \tag{10}$$

$$\varphi(p) := \mathrm{H}(4pq) - 4p(\mathrm{H}(p) - \log p). \tag{11}$$

$$\text{For } 0 \le p \le 1, \text{ let's also define:} \quad \psi(p) := p^2 \log \frac{1}{p^2} - 2\,\mathrm{H}(p). \tag{12}$$

▶ **Intuition:** Before going on, we pause to give some intuition about the choice of the function ψ and the function κ below (15). In the FEI conjecture (2), the right hand side, $\mathsf{Inf}(f)$, does not depend on whether we take the range of f to be $\{-1,+1\}$ or $\{0,1\}$. In contrast, the left hand side, $\mathbb{H}(f)$, depends on the range being $\{-1,+1\}$. Just as the usual entropy-influence inequality composes w.r.t. the parity operation (over disjoint variables) with $\{-1,+1\}$ range, we expect a corresponding inequality with $\{0,1\}$ range to hold for the AND operation (and by symmetry for the OR operation). However, Lemma 6 shows the translation to $\{0,1\}$-valued functions results in the annoying additive "error" term $\varphi(p)$. Such additive terms that depend on p create technical difficulties in the inductive proofs below and we need to choose the appropriate functions of p carefully.

For example, we know $4\,\mathbf{H}(f_{\mathbb{B}}) + \varphi(p) = \mathbb{H}(f) = 4\,\mathbf{H}(1 - f_{\mathbb{B}}) + \varphi(q)$ from Lemma 6. If the conjectured inequality for the $\{0,1\}$-valued entropy-influence inequality has an additive error term $\psi(p)$ (see (13) below), then we must have $\mathbf{H}(f_{\mathbb{B}}) - \mathbf{H}(1 - f_{\mathbb{B}}) = \psi(p) - \psi(q) = (\varphi(q) - \varphi(p))/4 = p^2 \log \frac{1}{p^2} - q^2 \log \frac{1}{q^2}$, using (11). Hence, we may conjecture that $\psi(p) = p^2 \log \frac{1}{p^2} +$ (an additive term symmetric w.r.t. p and q). Given this and the other required properties, e.g., Lemma 7 below, for the composition to go through, lead us to the definition of

ψ in (12). Similar considerations w.r.t. composition by parity operation (in addition to those by AND, OR, and NOT) leads us to the definition of κ in (15). ◀

Let us define the **FEI01 Inequality** (the 0-1 version of FEI) as follows:

$$\mathbf{H}(f_{\mathbb{B}}) \leq c \cdot \mathsf{as}(f) + \psi(p), \tag{13}$$

where $p = \widehat{f_{\mathbb{B}}}(\emptyset) = \Pr_x[f_{\mathbb{B}}(x) = 1]$ and c is a constant to be fixed later.

The following technical lemma gives us the crucial property of ψ:

Lemma 7. *For ψ as above and $p_1, p_2 \in [0,1]$, $p_1 \cdot \psi(p_2) + p_2 \cdot \psi(p_1) \leq \psi(p_1 p_2)$.*

Given this lemma, an inductive proof yields our theorem for read-once formulas over the complete basis of $\{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}\}$.

Theorem 7. *The FEI01 inequality (13) holds for all read-once Boolean formulas using* AND*,* OR*, and* NOT *gates, with constant $c = 5/2$.*

To switch to the usual FEI inequality (in the $\{-1, +1\}$ notation), we combine (13) and (10) to obtain

$$\mathbb{H}(f) \leq 10 \cdot \mathsf{as}(f) + \kappa(p), \quad \text{where} \tag{14}$$

$$\kappa(p) := 4\psi(p) + \varphi(p) = -8\,\mathrm{H}(p) - 8pq - (1 - 4pq)\log(1 - 4pq). \tag{15}$$

Since it uses the $\{-1, +1\}$ range, we expect that (14) should be preserved by parity composition of functions. The only technical detail is to show that the function κ also behaves well w.r.t. parity composition. We show that this indeed happens. This leads us to the main theorem of this section:

Theorem 8. *If f is computed by a read-once formula using* AND*,* OR*,* XOR*, and* NOT *gates, then $\mathbb{H}(f) \leq 10\,\mathsf{Inf}(f) + \kappa(p)$.*

Remark 2. The parity function on n variables shows that the bound in Theorem 8 is tight; it is not tight without the additive term $\kappa(p)$. It is easy to verify that $-10 \leq \kappa(p) \leq 0$ for $p \in [0, 1]$. Hence the theorem implies $\mathbb{H}(f) \leq 10\mathsf{Inf}(f)$ for all read-once formulas f using AND, OR, XOR, and NOT gates.

6 A Bound for Real Valued Functions via Second Moment

Due to space constraints we only state the theorem here, the full proof appears in [2].

Theorem 9. *If $f = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S$ is a real-valued function on the domain $\{0,1\}^n$ such that $\sum_S |\hat{f}(S)^2| = 1$ then for any $\delta > 0$,*

$$\sum_{S \subseteq [n]} \hat{f}(S)^2 \log\left(\frac{1}{\hat{f}(S)^2}\right) = \sum_S |S|^{1+\delta} \widehat{f}(S)^2 + 2\log_{1+\delta} n + 2(2\log n)^{1+\delta/\delta} (\log n)^2 \ .$$

As a corollary to Theorem 9, we also obtain the bound (7) in terms of the first and second moments of sensitivities of a function.

Acknowledgements. N. Saurabh thanks Ryan O'Donnell for helpful discussions on Lemma 3.

References

1. Bourgain, J., Kalai, G.: Influences of variables and threshold intervals under group symmetries. Geometric and Functional Analysis GAFA **7**(3), 438–461 (1997)
2. Chakraborty, S., Kulkarni, R., Lokam, S.V., Saurabh, N.: Upper bounds on fourier entropy. Tech. rep., Electronic Colloquium on Computational Complexity (ECCC), TR13-052 (2013). http://eccc.hpi-web.de/report/2013/052
3. Das, B., Pal, M., Visavaliya, V.: The entropy influence conjecture revisited. Tech. rep. (2011). arXiv:1110.4301
4. Diakonikolas, I., Raghavendra, P., Servedio, R., Tan, L.: Average sensitivity and noise sensitivity of polynomial threshold functions. SIAM Journal on Computing **43**(1), 231–253 (2014)
5. Friedgut, E., Kalai, G.: Every monotone graph property has a sharp threshold. Proceedings of the American Mathematical Society **124**(10), 2993–3002 (1996)
6. Gopalan, P., Kalai, A.T., Klivans, A.R.: Agnostically learning decision trees. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 527–536 (2008)
7. Harsha, P., Klivans, A., Meka, R.: Bounding the sensitivity of polynomial threshold functions. Theory of Computing **10**(1), 1–26 (2014)
8. Kahn, J., Kalai, G., Linial, N.: The influence of variables on boolean functions. In: Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, pp. 68–80 (1988)
9. Keller, N., Mossel, E., Schlank, T.: A note on the entropy/influence conjecture. Tech. rep. arXiv:1105.2651 (2011)
10. Klivans, A., Lee, H., Wan, A.: Mansour's conjecture is true for random dnf formulas. In: Proceedings of the 23rd Conference on Learning Theory, pp. 368–380 (2010)
11. Mansour, Y.: An $o(n^{\log\log n})$ learning algorithm for dnf under the uniform distribution. Journal of Computer and System Sciences **50**(3), 543–550 (1995)
12. O'Donnell, R.: Computational applications of noise sensitivity. Ph.D. thesis. MIT (2003)
13. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press (2014)
14. O'Donnell, R., Tan, L.Y.: A composition theorem for the fourier entropy-influence conjecture. In: Proceedings of Automata, Languages and Programming - 40th International Colloquium (2013)
15. O'Donnell, R., Wright, J., Zhou, Y.: The fourier entropy-influence conjecture for certain classes of boolean functions. In: Proceedings of Automata, Languages and Programming - 38th International Colloquium, pp. 330–341 (2011)
16. Wan, A., Wright, J., Wu, C.: Decision trees, protocols and the entropy-influence conjecture. In: Innovations in Theoretical Computer Science, ITCS 2014, pp. 67–80 (2014)
17. de Wolf, R.: A brief introduction to fourier analysis on the boolean cube. Theory of Computing, Graduate Surveys **1**, 1–20 (2008)

Author Index