33rd International Symposium on Algorithms and Computation

ISAAC 2022, December 19–21, 2022, Seoul, Korea

Edited by

Sang Won Bae

Heejin Park

Editors

Sang Won Bae
Kyonggi University, Suwon, Korea
swbae@kgu.ac.kr

Heejin Park
Hanyang University, Seoul, Korea
hjpark@hanyang.ac.kr

ACM Classification 2012
Theory of computation; Mathematics of computing

ISBN 978-3-95977-258-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-258-7.

Publication date
December, 2022

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ISAAC.2022.0

ISBN 978-3-95977-258-7 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at the 33rd International Symposium on Algorithms and Computation (ISAAC 2022). ISAAC 2022 was held in a hybrid manner on December 19–21, 2022 and was organized by Hanyang University, Korea. ISAAC 2022 provided a forum for researchers working in the areas of algorithms, theory of computation, and computational complexity. The technical program of the conference included 65 contributed papers. We received 177 submissions in response to the call for papers. Each submission received at least three reviews. The program committee held electronic meetings using EasyChair. In the end, the Program Committee selected 65 of the submissions for presentation at the symposium.

The program committee selected the following paper as the winner of the ISAAC 2022 Best Paper Award:

- On Maximizing Sums of Non-Monotone Submodular and Linear Functions by Benjamin Qi.

Since Benjamin Qi is a student, there is no winner of the Best Student Paper Award. The conference included two invited presentations, delivered by Jeff Erickson (University of Illinois Urbana-Champaign) and Kunihiko Sadakane (The University of Tokyo). Abstracts of their talks are included in this volume. We wish to thank all the authors who submitted extended abstracts for consideration, the program committee members for their scholarly efforts, and all external reviewers who assisted in the evaluation process.

We are grateful to Hanyang University for financial support and the local organizers of ISAAC 2022. Finally, we acknowledge the endorsement from Special Interest Group on Theoretical Computer Science of KIISE.

December 2022

Sang Won Bae and Heejin Park

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Sang Won Bae (co-chair, Kyonggi University)
Hideo Bannai (Tokyo Medical and Dental University)
Ho-Lin Chen (National Taiwan University)
Siu-Wing Cheng (Hong Kong University of Science and Technology)
Jinhee Chun (Tohoku University)
Adrian Dumitrescu (Algoresearch L.L.C.)
Shuichi Hirahara (National Institute of Informatics)
Wing-Kai Hon (National Tsing Hua University)
Seok-Hee Hong (University of Sydney)
Sungjin Im (University of California, Merced)
Seungbum Jo (Chungnam National University)
Christian Knauer (Universität Bayreuth)
Inbok Lee (Korea Aerospace University)
Anil Maheshwari (Carleton University)
Subhas Nandy (Indian Statistical Institute)
Yoshio Okamoto (University of Electro-Communications)
Hirotaka Ono (Nagoya University)
Sang-il Oum (Institute for Basic Science and KAIST)
Heejin Park (co-chair, Hanyang University)
Solon Pissis (CWI)
Kunihiko Sadakane (University of Tokyo)
Francesco Silvestri (University of Padova)
Fabian Stehn (Universität Bayreuth)
Wing-Kin Sung (National University of Singapore)
Takeshi Tokuyama (Kwansei Gakuin University)
Ryuhei Uehara (Japan Advanced Institute of Science and Technology)
André van Renssen (University of Sydney)
Kevin Verbeek (Eindhoven University of Technology)
Tomasz Walen (University of Warsaw)
Alexander Wolff (Universität Würzburg)
Deshi Ye (Zhejiang University)
Guochuan Zhang (Zhejiang University)

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of External Reviewers

Mikkel Abrahamsen
Hee-Kap Ahn
Jungho Ahn
Harry Altman
Hyung-Chan An
Antonios Antoniadis
Tetsuya Araki
Alma Arevalo Loyola
Artem Baklanov
Sayan Bandyapadhyay
Matthias Bentert
Arindam Biswas
Silvia Bonomi
Valentin Bouquet
Milutin Brankovic
Marco Bressan
Frederik Brüning
Yixin Cao
Matteo Ceccarello
Sankardeep Chakraborty
Sourav Chakraborty
Ching-Lueh Chang
Panagiotis Charalampopoulos
Kushagra Chatterjee
Ke Chen
Qingyun Chen
Yong Chen
Gecheng Cheng
Otfried Cheong
Nai-Hui Chia
Man Kwun Chiu
Christopher Chubb
Jacobus Conradi
Linda Cook
Bireswar Das
Gautam K Das
Samir Datta
Sami Davies
Jean-Lou De Carufel
Minati De
Hu Ding
Benjamin Doerr
Yichao Duan
Kunal Dutta
Thomas Dybdahl Ahle
György Dósa
Eduard Eiben
Nicolas El Maalouly
Christian Engels
Sándor Fekete
Qilong Feng
Guillaume Fertin
Oksana Firman
Zhiguo Fu
Jose Fuentes
Kaito Fujii
Hiroshi Fujiwara
Takuro Fukunaga
Martin Fürer
Pawel Gawrychowski
Tzvika Geft
Jakob Geiger
Panos Giannopoulos
Daniel Gibney
Tatsuya Gima
Paweł Gora
Joachim Gudmundsson
Yo-Sub Han
Tesshu Hanaka
Jan-Henrik Haunert
Tim Hegemann
Diptarama Hendrian
Kiya Hironori
Duc A. Hoang
Felix Hommelsheim
Takashi Horiyama
Ling-Ju Hung
Tomohiro I
Shunsuke Inenaga
Dalu Jacob
Klaus Jansen
Jesper Jansson
Varunkumar Jayapaul
Vincent Jugé
Andrzej Kaczmarczyk
Naonori Kakimura
Sayaka Kamei
Frank Kammer
Adam Karczmarz
Matti Karppa

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Eun Jung Kim
Jae-Hoon Kim
Jin Wook Kim
Yonghwan Kim
Sándor Kisfaludi-Bak
Masashi Kiyomi
Jonathan Klawitter
Boris Klemz
Felix Klesen
Dušan Knop
Yasuaki Kobayashi
Yusuke Kobayashi
Tomasz Kociumaka
Stefanos Kourtis
Martin S. Krejca
R. Krithika
Erik Krohn
Myroslav Kryven
Ariel Kulik
Mrinal Kumar
Ravi Kumar
Kazuhiro Kurita
O-Joung Kwon
Dominik Köppl
Elmar Langetepe
Jason Li
Xianyue Li
Peihai Liu
Thomas Locher
Dong Lu
Xuanlong Ma
Subhamoy Maitra
Diptapriyo Majumdar
Yury Makarychev
Yuchen Mao
Rogers Mathew
Lili Mei
Laura Merker
Adrian Miclăuș
Takuya Mieno
Gopinath Mishra
Joseph Mitchell
Eiji Miyano
Kaushik Mondal
Anish Mukherjee
Nikolas Mählmann
Tobias Mömke
Joong Chae Na
Samuel Nalin
Mikito Nanashima
Giacomo Nannicini
Daniel Neuen
Soeren Nickel
Bengt J. Nilsson
Saeed Odak
Kensuke Onishi
Michal Opler
Yota Otachi
Supantha Pandit
Leonard Papenmeier
Irene Parada
Matthew Patitz
Jakub Pawlewicz
Paolo Pellizzoni
Aditya Petety
Alexandru Popa
Nicola Prezza
Lianrong Pu
Dömötör Pálvölgyi
Luowen Qian
Jakub Radoszewski
Arash Rafiey
Amatur Rahman
Abhishek Rathod
Alexander Ravsky
Giuseppe Re
Marcel Roeloffzen
Bodhayan Roy
Sasanka Roy
Tobias Rupp
Toshiki Saitoh
Srinivasa Rao Satti
Saket Saurabh
Manfred Scheucher
Christiane Schmidt
Daniel Schoepflin
Yuan Sha
Shuai Shao
Akiyoshi Shioura
Marie Diana Sieper
Mark Siggers
Martin Skutella
Michiel Smid
Bettina Speckmann
Frits Spieksma
Joachim Spoerhase

Frank Staals
Georgios Stamoulis
Juliusz Straszyński
Yuichi Sudo
Akira Suzuki
Paolo Sylos Labini
Karthick T
Prafullkumar Tale
Suguru Tamaki
Yuma Tamura
Sharma V. Thankachan
Josef Tkadlec
Meng-Tsung Tsai
Kostas Tsichlas
Torsten Ueckerdt
Seeun William Umboh
Vincent Vajnovszki
Jasper van Doornmalen
Marc Vicuna
Giovanni Viglietta
Antoine Vigneron
Hung-Lung Wang
Kai Wang
Naqueeb Warsi
Kunihiro Wasa
Hao-Ting Wei
Max Willert
Michal Wlodarczyk
Sampson Wong
Chenchen Wu
Yicheng Xu
Takashi Yamakawa
Katsuhisa Yamanaka
Yu Yokoi
Sang Duk Yoon
Wei Yu
Or Zamir
An Zhang
Liwei Zhang
Peng Zhang
Yong Zhang
Yiming Zhao
Johannes Zink
Wiktor Zuba

Obituary

Takao Nishizeki 1947–2022

Takao Nishizeki was born in Fukushima Prefecture Japan on Feburuary 11, 1947. He received his B.E., M. E., and Dr. Eng. from Tohoku University in 1969, 1971, and 1974, respectively, and became an assistant professor of Department of Communication Engineering, Tohoku University, Japan. He was promoted to associate professor and full professor in 1976 and 1988, respectively. He was the dean of Graduate School of Information Sciences, Tohoku University for two years from 2008. After his retirement from Tohoku University, he worked as a professor of Kwansei Gakuin University from 2010 to 2015.

Takao was a distinguished and influential researcher of theoretical computer science(TCS), and his main research area was graph algorithms. Most of combinatorial graph problems are NP-hard, and hence intractable in general. However, he thought there should be a theoretical framework to design efficient algorithms for graphs used in practical applications such as electrical circuits and VLSI design. Those graphs are often systematically constructed, and hence have special structures. In his seminal paper "Linear-time computability of combinatorial problems on series-parallel graphs (J. ACM 29-3, 1982, coauthored with Takamizawa and Saito)", he developed a unified theory to design linear-time algorithms for several combinatorial problems on the series-parallel graph, which is a fundamental model for electrical circuits. He extended the theory to partial k-trees (i.e., graphs with bounded tree-width), which led to the fixed parameter complexity theory. He also gave pioneering works on graph drawing, and *Planar Graph Drawing* (Nishizeki and Rahman, 2004) is a popular textbook. He was honored with many awards including Commendation for Science and Technology by MEXT Japan, and named as Fellow of ACM, IEEE, IEICE, IPSJ, and Bangladesh Academy of Sciences.

Besides his research contributions, we appreciate his leadership to promote the research community of TCS. In particular, he contributed to the establishment of three major conference series: ISAAC, Graph Drawing and WALCOM.

Especially, we could consider him as the father of our ISAAC. In 1990, he organized with his colleagues the SIGAL International Symposium inviting many influential researchers. Takao proposed in that Symposium to continue it as an annual event held in Asia-Pacific region in order to promote word-wide collaboration of our TCS community, and it was the birth of the ISAAC series. He had taken responsibility to serve as the Chair of Advisory Committee for 17 years, and worked hard to make the long-term plan and selecting organizers, hosts and venues of future symposia. He attended every ISAAC to give an opening remark to thank organizers and participants. He also contributed as the symposium chair (1990 and 2007) and PC chair (1992) as well as coauthors of 24 research articles.

Takao passed away on January 30, 2022. We pray that his soul may rest in peace.

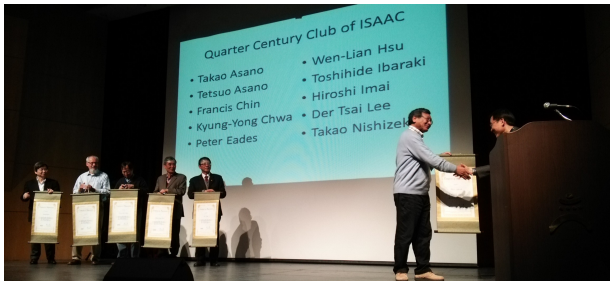


Figure 1 Takao Nishizeki receiving the certificate of appreciation at the ISAAC 25th anniversary.

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Kyung-Yong Chwa, 1946–2021

Professor Kyung-Yong Chwa was born in Seoul, Korea on February 4, 1946, and raised in Busan. He received his B.S. degree in electrical engineering from Seoul National University in 1971 and his M.S. and Ph.D. degrees in electrical engineering and computer science from Northwestern University in 1977 and 1980, respectively, under the supervision of Professor S. Louis Hakimi. He joined the Department of Computer Science at KAIST in 1980, and retired from KAIST in 2011 at his age of 65.

The main research area of Professor Chwa was graph theory and algorithms, while his interests covered many areas including parallel and distributed algorithms, online algorithms, approximation algorithms, computational geometry and computer graphics. In his seminal paper, "Schemes for fault-tolerant computing: a comparison of modularly redundant and t-diagnosable systems (Inform. Cont. 49(3), 1981)," he proposed a new comparison-based model of diagnosable systems for fault-tolerant computing and efficient algorithms under the model. His model is still being referenced for today as a theoretical base of fault-tolerant algorithms in distributed environments.

Professor Chwa is indeed the "father" of Theoretical Computer Science (TCS) in Korea. He was the frontier and a leader to establish the Algorithm community in Korea. He has successfully supervised 24 Ph.D.s and 74 masters at KAIST until his retirement, and they continue their career in academy and industry. In 1990, he founded Special Interest Group on TCS in Korea and served as the first chair. In 1997, he served as the President of the Korean Institute of Information Scientists and Engineers (KIISE). In particular, he was devoted to educating algorithmic thinking for young students, by hosting an annual Asia regional contest of ACM ICPC (Int. Collegiate Programming Contest) since 2000 and organizing the 14th IOI (Int. Olympiad on Informatics) in 2002. His dedicated leadership and achievements were recognized when he was honored the Order of Service Merit and a member of the Korean Academy of Science and Technology (KAST) in 2003.

Professor Chwa was one of the founding members who had initiated the conference ISAAC with Professor Takao Nishizeki and other colleagues. He greatly contributed to the success of ISAAC by hosting ISAAC in Daejeon, Korea (1998) and in Jeju, Korea (2010), publishing many papers in ISAAC, and serving as a Advisory Committee member of ISAAC.

Professor Chwa passed away on November 5, 2021. We wish he may rest in peace.

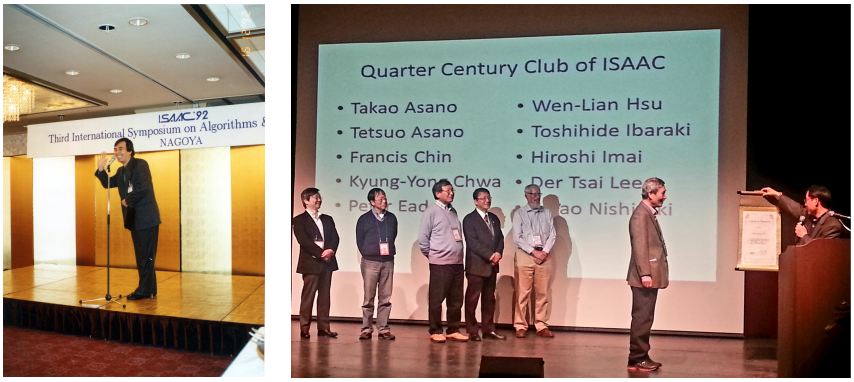


Figure 2 Professor Chwa in his speech at the 3rd ISAAC in Nagoya and receiving the certificate of appreciation at the 25th anniversary of ISAAC.

Succinct Representations of Graphs

Kunihiko Sadakane ✉
Graduate School of Information Science and Technology, The University of Tokyo, Japan

Abstract

We consider the problem of finding succinct representations of graphs, that is, encodings using asymptotically the minimum number of bits which support queries on the graphs efficiently. For a special class of graphs, there exist many theoretical results and practical implementations on ordered trees. On the other hand, for wider classes of graphs, though there are many results on counting the number of non-isomorphic graphs belonging to a graph class, there were few number of results on their succinct representations until recently.

In this talk, we review some recent results on succinct representations of graphs such as interval, permutation, circle, circular-arc, trapezoid, circle-trapezoid, k-polygon, circle-polygon, cograph, separable, ptolemaic, distance hereditary, clique width k, block, cactus, series-parallel, planar, tree width k, path, boxicity k, chordal bipartite, strongly chordal, chordal graphs, etc.

2012 ACM Subject Classification Theory of computation → Data compression; Mathematics of computing → Graph enumeration

Keywords and phrases Graph Enumeration, Succinct Data Structure, Compression

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.1

Category Invited Talk

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park; Article No. 1; pp. 1:1–1:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Tragedy of Being Almost but Not Quite Planar

Jeff Erickson
University of Illinois Urbana-Champaign, IL, USA

Abstract

Planar graphs have been fertile grounds for algorithms research for decades, both because they model several types of real-world networks, and because they admit simpler and and faster algorithms than arbitrary graphs. Many important structural properties of planar graphs extend naturally to graphs that embed on more complex surfaces. As a result, efficient algorithms for planar graphs often extend naturally to higher-genus surface graphs with little or no modification.

I will describe a few of my favorite exceptions to this rule – classical problems that admit simple, efficient, and practical algorithms for planar graphs, but where algorithms for graphs on other surfaces are significantly slower and/or more complex.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Computational geometry

Keywords and phrases planar graphs, surface graphs, algorithms, open problems

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.2

Category Invited Talk

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Local Search Algorithm for the Min-Sum Submodular Cover Problem

Lisa Hellerstein
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA

Thomas Lidbetter
Department of Engineering Systems and Environment, University of Virginia, Charlottesville, VA, USA
Department of Management Science and Information Systems, Rutgers Business School, Newark, NJ, USA

R. Teal Witter
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA

Abstract

We consider the problem of solving the Min-Sum Submodular Cover problem using local search. The Min-Sum Submodular Cover problem generalizes the NP-complete Min-Sum Set Cover problem, replacing the input set cover instance with a monotone submodular set function. A simple greedy algorithm achieves an approximation factor of 4, which is tight unless P=NP [Streeter and Golovin, NeurIPS, 2008]. We complement the greedy algorithm with analysis of a local search algorithm. Building on work of Munagala et al. [ICDT, 2005], we show that, using simple initialization, a straightforward local search algorithm achieves a $(4+\epsilon)$-approximate solution in time $O(n^3 \log(n/\epsilon))$, provided that the monotone submodular set function is also second-order supermodular. Second-order supermodularity has been shown to hold for a number of submodular functions of practical interest, including functions associated with set cover, matching, and facility location. We present experiments on two special cases of Min-Sum Submodular Cover and find that the local search algorithm can outperform the greedy algorithm on small data sets.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Local search, submodularity, second-order supermodularity, min-sum set cover

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.3

Related Version *Full Version*: `https://arxiv.org/abs/2209.03054`

Supplementary Material *Software*: `https://github.com/rtealwitter/Local-Search` archived at `swh:1:dir:a757879418c8fba172d787428be9b3480f008821`

Funding *Lisa Hellerstein*: NSF Award IIS-1909335
Thomas Lidbetter: NSF Award IIS-1909446

Acknowledgements We thank Christopher Musco for pointing out that the set function induced by entropy on continuous domains is not submodular.

1 Introduction

We consider the Min-Sum Submodular Cover problem, defined as follows. The input to the problem consists of an oracle for a monotone submodular function $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$, and positive costs $c_1, \ldots, c_n \in \mathbb{R}_{>0}$, where $[n] = \{1, \ldots, n\}$. Let $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be such that for all $S \subseteq [n]$, $c(S) = \sum_{i \in S} c_i$. We refer to u as the "utility" and c as the "cost" function. The problem is to find the permutation of the elements of $[n]$ that minimizes

$$\sum_{i=1}^{n} c(S_i)\left(u(S_i) - u(S_{i-1})\right) \tag{1}$$

where S_i is the set containing the first i elements of the permutation. The Min-Sum Submodular Cover problem generalizes the NP-Complete Min-Sum Set Cover problem introduced by Feige et al. [5]. It has a simple greedy algorithm that achieves a 4-approximation [9, 18].[1] The 4-approximation is tight, assuming $P \neq NP$ [5].

In this work, we analyze a local search algorithm for Min-Sum Submodular Cover. Local search algorithms have been extensively applied to discrete optimization problems [3, 14, 15] and offer several benefits over other types of algorithms [2]. One advantage of local search algorithms compared to greedy methods in practice is their ability to explore a diverse set of solutions. In Section 4, we present the results of preliminary experiments which demonstrate that this ability can yield improved solutions to Min-Sum Submodular Cover instances.

The local search algorithm we consider works in iterations, starting with an initial solution. In an iteration, the algorithm updates the current solution to its best "neighbor". Once a solution is locally optimal (or after a fixed number of iterations), the algorithm returns the current solution.

Our analysis builds on previous work of Munagala et al. [16] for the Pipelined Set Cover problem (defined below). Munagala et al. exploit the observation that utility can be attributed to elements of the ground set for the set cover instance, and use these elements as variables in their linear program. The main challenge of generalizing their analysis is that the utility in general submodular set functions is more abstract and cannot be attributed to particular objects. In order to apply the linear program used in Munagala et al., our analysis relies on an additional property of the utility function called second-order supermodularity. We leave as an open question whether the algorithm gives a $(4 + \epsilon)$-approximation even without second-order supermodularity.

Second-order supermodularity was first studied by Korula et al. [12]. It can be viewed as a natural extension of submodularity: If one considers the multilinear extension $F : [0, 1]^n \to \mathbb{R}_{\geq 0}$ of a set function $f : \{0, 1\}^n \to \mathbb{R}_{\geq 0}$ (which is a way of interpolating the values of f from the vertices of the Boolean hypercube to points in its interior), the submodularity of f is equivalent to the property that the second partial derivatives of F are non-positive. As mentioned by Korula et al. [12] and Iyer et al. [11], the second-order supermodularity of f is equivalent to the property that the third partial derivatives of F are non-negative.

The second-order supermodularity property is not overly restrictive; there are several classes of submodular functions that have this property including weighted coverage functions, weighted matching functions, and facility location [12, 10, 11]. Since this property was first defined, improved bounds have been obtained for optimization problems by assuming the property [12, 10]. The related properties of second-order modularity and second-order submodularity have also been used in analyzing local search algorithms for constrained submodular maximization problems [6, 7].

We now define two special cases of Min-Sum Submodular Cover: the Pipelined Set Cover problem and the Min-Sum Facility Location problem.

Pipelined Set Cover

The inputs to the problem consist of (i) m "ground" elements $\{1, \ldots, m\} = [m]$, (ii) $D_1, \ldots, D_n$, a family of n subsets of the ground elements $[m]$ such that $\bigcup_{i \in [n]} D_i = [m]$, and (iii) positive costs $c_1, \ldots, c_n$ associated with each D_i. Let $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be such that for all $S \subseteq [n]$, $u(S)$ is the number of ground elements in $\bigcup_{i \in S} D_i$. We call u a "coverage" function.

[1] See the Appendix A.1 for comments on Streeter and Golovin [18].

Let $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be such that for all $S \subseteq [n]$, $c(S) = \sum_{i \in S} c_i$. The problem is to find the permutation of $[n]$ that minimizes the objective function in the Min-Sum Submodular Cover problem, $\sum_{i=1}^{n} c(S_i)\,(u(S_i) - u(S_{i-1}))$. Thus Pipelined Set Cover problem is equivalent to the special case of the Min-Sum Submodular Cover problem where the utility function u is a coverage function. The Min-Sum Set Cover problem is the special case of Pipelined Set Cover with unit costs. (We note that Munagala et al. also present results for Weighted Pipelined Set Cover, where the ground elements have weights.)

Min-Sum Facility Location

Consider the following problem facility location problem, studied by Krause and Golovin [13]. There is a set $[n]$ of possible locations where facilities could be opened, to serve a collection of m customers. Opening a facility at location a provides a service of value $M_{a,b}$ to customer b, where $M \in \mathbb{R}_{\geq 0}^{n \times m}$. The utility of opening facilities in a subset S of the locations is $u(S)$, where $u(S) = \sum_{b=1}^{m} \max_{a \in S} M_{a,b}$. This corresponds to the total value obtained by all the customers, assuming each customer chooses the open facility with highest service value. The problem of Krause and Golovin is to maximize the utility function u subject to a constraint on the number of facilities that can be opened.

We introduce a min-sum version of this facility location problem by considering the Min-Sum Submodular Cover problem with the utility function u just described, and with c_i representing the time to open a facility i. This problem corresponds to a situation where facilities will be opened in all n locations, but they can only be opened one at a time. $M_{a,b}$ represents the estimated value facility a will provide to customer b per unit of time, once facility a is opened. Minimizing the objective value $\sum_{i=1}^{n} c(S_i)\,(u(S_i) - u(S_{i-1}))$ corresponds to finding the order to build facilities so as to minimize lost value as facilities are built.

Our Contributions

We introduce the study of solving Min-Sum Submodular Cover using local search. Building on work of Munagala et al. [16], who presented a local-search algorithm for Pipelined Set Cover, we generalize their LP-based analysis by redefining a key quantity in their proof and using second-order supermodularity. We show that local search produces a $(4+\epsilon)$-approximate solution for Min-Sum Submodular Cover in time $O(n^3 \log(\frac{d}{\epsilon}))$, assuming second-order supermodularity of the utility function, when initialized with a d-approximate solution. We prove that a permutation listing the items in non-decreasing cost order is an n-approximate solution. Thus initializing local search with a non-decreasing cost permutation enables us to reach a $(4+\epsilon)$-approximate solution in time $O(n^3 \log(\frac{n}{\epsilon}))$. Applying this result to Pipelined Set Cover improves on the $O(n^3 \log(\frac{mn}{\epsilon}))$ time bound from Munagala et al., where m is the size of the ground set of the set cover instance, by eliminating the dependence on m. We also present results of experiments on two types of Min-Sum Submodular Cover problems: Pipelined Set Cover and Min-Sum Facility Location. Our empirical findings suggest that local search can reliably produce better solutions than the natural greedy algorithm on small data sets.

2 Preliminaries

Let $f(e|S) := f(S \cup \{e\}) - f(S)$ be the marginal utility of adding element e to set S. With this notation in hand, we define several useful properties of set functions.

▶ **Definition 1** (Set Function Properties). *Consider a positive integer n and set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$. We first define the following properties of f, which hold if the inequality given below for that property holds for all $S \subseteq [n]$ and all $i, j, k \in [n] \setminus S$,*

- *monotone:* $f(S \cup \{i\}) \geq f(S)$,
- *submodular (diminishing returns):* $f(i|S) \geq f(i|S \cup \{j\})$,
- *second-order supermodular:* $f(i|S) - f(i|S \cup \{j\}) \geq f(i|S \cup \{k\}) - f(i|S \cup \{k, j\})$.

Note that the way in which we have written the above properties illustrates the progression from monotonicity to submodularity and submodularity to second-order supermodularity: we arrive at the "next" property by subtracting the left-hand side from the right-hand side. Another related property is modularity: for all $S \subseteq [n]$, $f(S) = \sum_{i \in S} f(\{i\})$. In this paper, the functions we consider will be monotone set functions that are normalized, i.e., $f(\emptyset) = 0$ unless otherwise stated.

The Min-Sum Submodular Cover problem is a special case of the Min-Sum Permutation Problem, defined by Happach et al. [8]. That problem has the same objective function as Min-Sum Submodular Cover, and minimization may be over all permutations, or only over a subset of them. The only assumptions on u and c in [8] are that they are monotone and normalized.

3 A Local Search Algorithm for Min-Sum Submodular Cover

Munagala et al. [16] gave a local search algorithm for the special case of the Min-Sum Submodular Cover problem where u is a coverage function. Applying the same approach to the general Min-Sum Submodular Cover problem, we have the following local search algorithm: initialize the algorithm with a permutation π of $[n]$. Define a neighbor π' of π to be a permutation that can be be produced from π by removing the element in some position i of π and reinserting it in position j. Find the neighbor π' of π with lowest objective value (given by Equation 1). If that value is less than the objective value of π, then replace π by π' and repeat. Otherwise, output π. Pseudocode for this algorithm is given in Algorithm 1.

The analysis of Munagala et al. [16] shows that in the special case where u is a coverage function, Algorithm 1 achieves a $(4 + \epsilon)$-approximation to the optimal permutation. We generalize their analysis to all utility functions u that are submodular and second-order supermodular (in addition to being monotone and normalized, which we assume is the case for all utility functions in this paper).

Let π_c be a non-decreasing cost permutation, i.e., $c(\{\pi_c(i)\}) \leq c(\{\pi_c(i+1)\})$ for $i \in [n-1]$. We prove the following results.

▶ **Theorem 2.** *Fix a positive integer n. Let $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be a submodular and second-order supermodular set function and let $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be a modular set function. If Algorithm 1 converges before terminating, then the solution it returns is a 4-approximation to Min-Sum Submodular Cover on u and c.*

Unfortunately, we cannot guarantee that Algorithm 1 will converge before terminating. The next result guarantees a $(4 + \epsilon)$-approximation when the algorithm terminates.

▶ **Theorem 3.** *Consider the positive integer n, utility function u, and cost function c considered in Theorem 2. Fix $\epsilon > 0$. Let π be a d-approximate permutation. If Algorithm 1 does not converge before terminating, then the solution it returns (after $2n^3 \log(d/\epsilon)$ iterations), is a $(4 + \epsilon)$-approximation to Min-Sum Submodular Cover on u and c.*

Algorithm 1 Local search algorithm to produce a $(4+\epsilon)$-approximation.

```
Input: ε > 0, n > 0, utility function u : 2^[n] → ℝ_≥0,
cost function c : 2^[n] → ℝ_≥0, d-approximate permutation π
Output: permutation π
for iteration in {1, ..., 2n^3 log(d/ε)} do
    π* ← π
    for i, j ∈ [n] do
        # π' is π with π(i) moved to position j
        π' ← move(π, i, j)
        # objective(u, c, π) is Equation (1)
        if objective(u, c, π') < objective(u, c, π*) do
            π* ← π'
    if π* = π
        # Algorithm converged:
        # π is locally optimal with respect to moves
        return π # 4-approximation
    π ← π*
return π # (4+ε)-approximation
```

Assuming constant access query access to u and c, Algorithm 1 returns a $(4+\epsilon)$-approximation in $2n^3 \log(d/\epsilon)$ time.

As in Munagala et al. [16], in our analysis we consider a modified version of local search based on "insertions" rather than "moves." We find it easier to analyze local search with insertions and the approximation result immediately applies to local search with moves since a permutation that is locally optimal with respect to moves is also locally optimal with respect to insertions. In each iteration of local search with insertions, rather than considering the set of neighbors π' of π, the modified algorithm considers a set of what we will call pseudo-neighbors. Each is derived from π by taking an element appearing in some position i of π, and inserting a *second copy* of the element into some position $j < i$. Each pseudo-neighbor of π corresponds to a unique neighbor of π, produced from the pseudo-neighbor by removing the original copy of the repeated element (which appears closer to the end of the permutation).

Define the objective value of a pseudo-neighbor π' (which has length $n+1$) to be $\sum_{i=1}^{n+1} c'(S_i)[u(S_i) - u(S_{i-1})]$, where here S_i is the prefix of π' containing its first i elements, $u(S_i)$ is the value of u for the set of *distinct* items in S_i, and $c'(S_i) = \sum_{j=1}^{i} c(\{s_j\})$ where s_j is the element in position j of π'. That is, if both copies of the repeated element appear within the first i positions of π', then $c'(S_i)$ charges for both copies.

If the objective value of π is no greater than the value of its pseudo-neighbors, then the modified algorithm outputs π. Otherwise, the algorithm takes the pseudo-neighbor with lowest objective value, deletes the original copy of its repeated element, and uses the resulting permutation as the new value of π in the next iteration.

The objective value of a pseudo-neighbor of π is clearly greater than or equal to the objective value of the corresponding neighbor. Therefore, if π has no neighbor with lower objective value, then it has no pseudo-neighbor with lower objective value. It follows that the bounds we prove on the modified local search algorithm (with insertions) also apply to the original local search algorithm (with moves).

We prove Theorems 2 and 3 in the remainder of this section.

3.1 Proof of Theorem 2: 4-approximation

Say a permutation π is locally optimal if no pseudo-neighbor has lower objective value. We begin by proving that a locally optimal solution satisfies a certain inequality, expressed in terms of variables b_{ij}. This inequality is taken from the analysis in Munagala et al. [16], but we define the variables b_{ij} differently here. We will use the following technical Observation 4 to prove Lemma 5.

▶ **Observation 4.** *Consider three sequences of non-negative real numbers, $X_0, \ldots, X_n$, $Y_0, \ldots, Y_n$, and $C_0, \ldots, C_n$. Let $j \in [n]$. Suppose that the following hold: (i) $X_0 \geq Y_0$, (ii) for all $r \in \{j, \ldots, n\}$, $C_0 \leq C_r$ and $X_r \leq Y_r$, and (iii) $X_0 + \sum_{r=j}^{n} X_r = Y_0 + \sum_{r=j}^{n} Y_r$. Then*

$$C_0 X_0 + \sum_{r=j}^{n} C_r X_r \leq C_0 Y_0 + \sum_{r=j}^{n} C_r Y_r.$$

Proof. Rewriting the final assumption yields

$$\begin{aligned} X_0 - Y_0 = \sum_{r=j}^{n} (Y_r - X_r) \iff C_0(X_0 - Y_0) &= \sum_{r=j}^{n} C_0(Y_r - X_r) \\ \Rightarrow C_0(X_0 - Y_0) &\leq \sum_{r=j}^{n} C_r(Y_r - X_r) \end{aligned}$$

where the second implication follows from the non-negativity of $Y_r - X_r$ and the assumption that $C_0 \leq C_r$. Observation 4 follows immediately. ◀

▶ **Lemma 5.** *Suppose $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is a submodular and second-order supermodular set function and $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is a modular set function. Let L_j denote the first j elements of the locally optimal permutation and O_i denote the first i elements of the optimal permutation. Similarly, we use l_j to represent the jth element of the local permutation and o_i to represent the ith element of the optimal permutation. Then*

$$\sum_{r=j}^{n} c(L_r) \sum_{s=1}^{n} b_{sr} \leq [c(o_i) + c(L_{j-1})] \sum_{r=j}^{n} b_{ir} + \sum_{r=j}^{n} [c(o_i) + c(L_r)], \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} \tag{2}$$

where

$$\begin{aligned} b_{ij} &= u(o_i | O_{i-1} \cup L_{j-1}) - u(o_i | O_{i-1} \cup L_{j-1} \cup \{l_j\}) \\ &= u(l_j | O_{i-1} \cup L_{j-1}) - u(l_j | O_{i-1} \cup L_{j-1} \cup \{o_i\}). \end{aligned}$$

Proof. When u is a coverage function, as in the analysis in Munagala et al. [16], b_{ij} represents the number of ground elements covered in the optimal permutation by subset o_i (and not by $o_1, \ldots, o_{i-1}$) and in the local permutation by subset l_j (and not by $l_1, \ldots, l_{j-1}$). Our definitions of b_{ij} generalize this intuition to functions where the utility is more abstract. In particular, we can use telescoping sums to derive the following identities:

$$\sum_{i=1}^{n} b_{ij} = u(l_j | L_{j-1}) \qquad \sum_{r=j}^{n} b_{ir} = u(o_i | O_{i-1} \cup L_{j-1}) \qquad \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} = u(l_r | L_{r-1}) - b_{ir} \tag{3}$$

Then Equation (2) is equivalent to

$$\sum_{r=j}^{n} c(L_r)u(l_r|L_{r-1}) \leq [c(o_i) + c(L_{j-1})]u(o_i|O_{i-1} \cup L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_r)][u(l_r|L_{r-1}) - b_{ir}]. \tag{4}$$

Notice that Equation (4) is not *quite* equivalent to the property of local optimality with respect to insertions. Instead, we know the following (very similar) inequality holds by the property that L is locally optimal:

$$\sum_{r=j}^{n} c(L_r)u(l_r|L_{r-1}) \leq [c(o_i) + c(L_{j-1})]u(o_i|L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_r)]u(l_r|L_{r-1} \cup \{o_i\}). \tag{5}$$

We will now show that the right-hand side of Equation (5) lower bounds the right-hand side of Equation (4). Then Equation (4) follows from Equation (5). We do this through the following four conditions combined with Observation 4:

$$c(L_{j-1}) \leq c(L_{r-1}) \tag{6}$$

$$u(o_i|L_{j-1}) \geq u(o_i|O_{i-1} \cup L_{j-1}) \tag{7}$$

$$u(l_r|L_{r-1} \cup \{o_i\}) \leq u(l_r|L_{r-1}) - b_{ir} \tag{8}$$

$$u(o_i|L_{j-1}) + \sum_{r=j}^{n} u(l_r|L_{r-1} \cup \{o_i\}) = u(o_i|O_{i-1} \cup L_{j-1}) + \sum_{r=j}^{n}[u(l_r|L_{r-1}) - b_{ir}] \tag{9}$$

for all $i, j, r \in [n]$ with $j \leq r \leq n$. Equation (6) holds by the monotonicity of c and Equation (7) holds by the submodularity of u. Equation (8) holds because u is second-order supermodular. (This is where we use second-order supermodularity.)

Finally, Equation (9) holds by the following argument: Notice that the left-hand side is equal to $u([n]) - u(L_{j-1})$ since we sequentially sum the marginal utility of adding the next element to our current chain. Similarly, the right-hand side simplifies to $u([n]) - u(L_{j-1})$ after cancellation using Equation (3).

Using the above, we apply Observation 4 with $C_0 = c(o_i) + c(L_{j-1})$, $C_{r-j+1} = c(o_i) + c(L_{r-1})$, $X_0 = u(o_i|L_{j-1})$, $X_{r-j+1} = u(l_r|L_{r-1} \cup \{o_i\})$, $Y_0 = u(o_i|O_{i-1} \cup L_{j-1})$, and $Y_{r-j+1} = u(l_r|L_{r-1}) - b_{ir}$. This yields

$$[c(o_i) + c(L_{j-1})]u(o_i|L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_{r-1})][u(l_r|L_{r-1} \cup \{o_i\})]$$

$$\leq [c(o_i) + c(L_{j-1})]u(o_i|O_{i-1} \cup L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_{r-1})][u(l_r|L_{r-1}) - b_{ir}].$$

Then the right-hand side of Equation (4) must be greater than or equal to the right-hand side of Equation (5). Therefore Lemma 5 follows by Equation (5). ◀

Proof of Theorem 2. We will now prove that a locally optimal permutation with respect to insertions is (at worst) a 4-approximation of the optimal permutation. To do this, we will show that the same linear program used by Munagala et al. [16] to prove their 4-approximation for coverage functions also applies to all utility functions that are monotone, submodular and second-order supermodular. The linear program in Munagala et al. [16] is:

$$\text{maximize } \sum_{j=1}^{n} c(L_j) \sum_{i=1}^{n} b_{ij} \tag{10}$$

$$\text{subject to } \sum_{i=1}^{n} c(O_i) \sum_{j=1}^{n} b_{ij} \leq 1 \quad \text{and}$$

$$\sum_{r=j}^{n} c(L_r) \sum_{s=1}^{n} b_{sr} \leq (c(o_i) + c(L_{j-1})) \sum_{r=j}^{n} b_{ir} + \sum_{r=j}^{n} (c(o_i) + c(L_r)) \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} \quad \forall i, j \in [n] \tag{11}$$

where the variables b_{ij} are non-negative for all $i, j \in [n]$.

The first constraint scales the variables in the optimal permutation so that the objective is the ratio of the local permutation to the optimal permutation. By Lemma 5, the second constraint must hold for locally optimal L, with the given values for b_{ij}. As shown in Munagala et al. [16], there is a feasible solution to the dual of the LP with objective value 4 so, by strong duality, the locally optimal permutation achieves a 4-approximation to the optimal permutation. ◀

3.2 Proof of Theorem 3: $(4 + \epsilon)$-approximation

In this section, we show that local search reaches a $(4 + \epsilon)$-approximation in reasonable time. The proof is based on the following fact, which bounds the progress made in each "round" of the local search (i.e., in each iteration of the while loop in Algorithm 1).

▶ **Fact 6.** *Let π be a permutation of $[n]$ that is an M-approximation to Min-Sum Submodular Cover. Then applying one round of local search to this permutation will either establish that it is a local optimum or yield a new permutation that is an $(M - \frac{M-4}{2n})$-approximation.*

Proof. In Munagala et al. [16], they prove that Fact 6 holds for a Pipelined Filter Ordering instance, i.e., when u is a coverage function. Consider an M-approximate solution. Let A be a variable representing the reduction in approximation factor after making the local search step. As in Equation (11), the variables b_{ij} are non-negative real numbers and correspond to a concept of shared utility formalized in Lemma 5. Then the following linear program is what Munagala et al. [16] use to lower-bound the improvement in approximation ratio:

$$\text{minimize } A$$

$$\text{subject to } \sum_{i=1}^{n} c(O_i) \sum_{j=1}^{n} b_{ij} \leq 1 \text{ and}$$

$$\sum_{r=j}^{n} c(L_r) \sum_{s=1}^{n} b_{sr} \leq A + (c(o_i) + c(L_{j-1})) \sum_{r=j}^{n} b_{ir} + \sum_{r=j}^{n} (c(o_i) + c(L_r)) \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} \text{ and}$$

$$\sum_{j=1}^{n} c(L_j) \sum_{i=1}^{n} b_{ij} \geq M \quad \forall i, j \in [n].$$

For the more general case where u is an abstract submodular and second-order supermodular utility function, the first and third inequalities trivially hold for our generalized definition of b_{ij}. The second inequality follows from Lemma 5. By taking the dual, Munagala et al. [16] show that the objective of the primal and therefore the reduction in approximation

ratio is at least $\frac{M-4}{2n}$ which gives Fact 6. When u is an arbitrary submodular and second-order supermodular function, it follows from Lemma 5 that the same linear program still lower-bounds the improvement. ◀

To achieve a good bound on the time required for the local search for Min-Sum Submodular Cover, we want to begin the local search from a permutation that is not too far from optimal. The following theorem gives such a permutation.

▶ **Theorem 7** (Non-Decreasing Cost). *A permutation that orders the elements $i \in [n]$ in non-decreasing order of $c(\{i\})$ is an n-approximate solution to Min-Sum Submodular Cover.*

Proof. Suppose, without loss of generality, that $c(\{1\}) \leq \cdots \leq c(\{n\})$.

Suppose an optimal solution is given by the sets $\emptyset = S_0, S_1 \ldots, S_n$, and for each $j \in [n]$, let $S'_j = [\max S_j]$, where $\max S_j$ is the maximum integer in S_j. For instance, for $n = 4$ and $S_1 = \{2\}$, $S_2 = \{12\}$, $S_3 = \{124\}$, $S_4 = \{1234\}$, we have $S'_1 = \{12\}$, $S'_2 = \{12\}$, $S'_3 = \{1234\}$, $S'_4 = \{1234\}$. Setting $[0]$ and S'_0 equal to $\emptyset$, observe that

$$\sum_{j=1}^{n} c([j])(u([j]) - u([j-1])) \leq \sum_{j=1}^{n} c(S'_j)(u([j]) - u([j-1])) = \sum_{j=1}^{n} c(S'_j)(u(S'_j) - u(S'_{j-1})),$$

where the inequality follows by the monotonicity of c and the equality follows (not term-wise but in total) by considering when utility is accrued by each side.

Now, for each $j \geq 0$, we have $c(S_j) \geq c(\{\max S_j\})$, by the monotonicity of c. Also, since $c(\{\max S_j\}) \geq c(\{i\})$ for every $i \in S_j$ by the indexing assumption,

$$n \cdot c(S_j) \geq n \cdot c(\{\max S_j\}) = n \cdot c(\{\max S'_j\}) \geq c(S'_j)$$

where the equality follows from the definition of S'_j and the second inequality follows since there are at most n elements in S'_j. Then the objective value of the optimal permutation is at least $\frac{1}{n}\sum_{j=1}^{n} c(S'_j)(u(S_j) - u(S_{j-1}))$, or equivalently by charging utility to each increase in cost,

$$\frac{1}{n}\sum_{j=1}^{n}(u([n]) - u(S_{j-1}))(c(S'_j) - c(S'_{j-1})).$$

By the monotonicity of u, this sum is at least

$$\frac{1}{n}\sum_{j=1}^{n}(u([n]) - u(S'_{j-1}))(c(S'_j) - c(S'_{j-1})) \Leftrightarrow \frac{1}{n}\sum_{j=1}^{n} c(S'_j)(u(S'_j) - u(S'_{j-1})).$$

This is at least $1/n$ times the objective value of the increasing cost permutation, by our earlier observation. ◀

We note that Theorem 7 also holds for Min-Sum Permutation Problems minimizing over all permutations where u only satisfies monotonicity and c only satisfies monotonicity and subadditivity (both are still normalized).

We can now prove that the output of Algorithm 1 is a $(4+\epsilon)$-approximation to Min-Sum Submodular Cover. The time bound assumes constant time oracle queries.

Proof of Theorem 3. The improvement in the quality of the solution in each round of local search, guaranteed by Fact 6, implies that a $(4+\epsilon)$-approximation is achieved within $O(n\log(\frac{d}{\epsilon}))$ rounds of Algorithm 1, when it is initialized with a d-approximate permutation. This implication was stated without proof by Munagala et al. [16], in proving the same

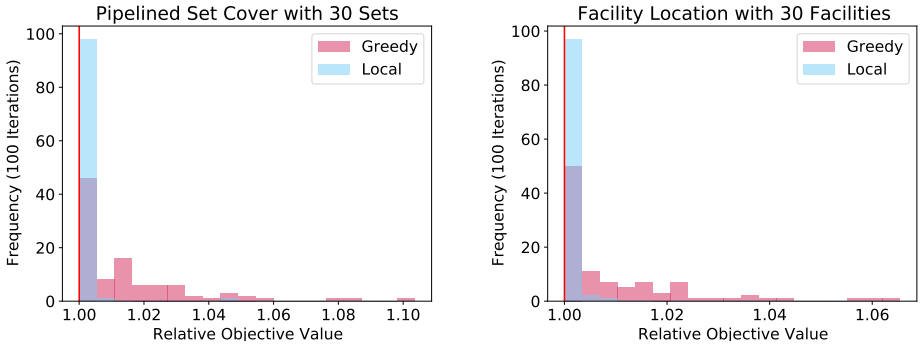


Figure 1 Histograms of greedy and local search performance with $n = 30$ in Pipelined Set Cover and Min-Sum Facility Location. The relative objective values are normalized with respect to the best of the 6 generated solutions (1 greedy and 5 local search). Frequency is reported from the 100 random instances generated for each dataset.

bounds for Pipelined Set Cover. For completeness, we present a proof of the implication in Appendix A.2. The $O(n^3 \log(\frac{d}{\epsilon}))$ time bound is achieved by spending $O(n^2)$ per round. To accomplish this, we do not recompute all n terms of the objective function for each of the $\theta(n^2)$ neighbors of the current solution. Instead, by considering "adjacent" neighbors sequentially, we can compute the objective function value for the next neighbor from the value obtained for the previous neighbor in constant time, by recomputing only two terms of the objective function. The time bound for the non-decreasing cost permutation follows from Theorem 7. ◀

4 Experiments

Assuming constant-time oracle access to the utility functions, the greedy algorithm runs in $O(n^2)$ total time, while our local search algorithm spends time $O(n^2)$ in each round. In our experiments, with no oracle, we had to compute the utility function.

The greedy algorithm is certainly faster than the local search, but it only explores one type of solution, where cost effective elements appear earlier in the permutation. Local search initialized with random permutations can sample from the entire solution space. Our experiments compare the greedy solution to local search solutions from four random initial permutations, and from a non-decreasing cost permutation. We run each local search for n steps (rather than running it to convergence, or until it is guaranteed to find a $(4+\epsilon)$-approximate solution). We see empirically that the best of the 5 local search solutions tends to be better than the worst, and also better than the greedy solution. In applications where computing is cheap, n is not large, and the quality of the solution is crucial, using local search may be preferable to using greedy.

Our experiments compare greedy and local search on 100 random instances of two problems: Pipelined Set Cover and Min-Sum Facility Location. For our random instances, we set $n = 30$ and each cost c_i to be a uniform random value between 0 and 1. Figure 1 shows the results of our experiments, with objective values given relative to the best of the 6 solutions (1 greedy and 5 local search). Local search finds the best solution in almost 100% of the 100 instances whereas greedy finds the best in roughly 50%.

Pipelined Set Cover

We perform experiments on synthetic, randomly generated instances of the (unweighted) Pipelined Set Cover problem with correlated subsets, following an approach of Babu et al. [4]. Recall that an instance of (unweighted) Pipelined Set Cover consists of a finite ground set $[m]$ and a collection of subsets $D_i \subseteq V$ for $i \in [n]$. The utility of a set $S \subseteq [n]$ is $u(S) = |\bigcup_{i \in S} D_i|$. The n subsets D_i in our random instance are divided into $\lceil n/\Gamma \rceil$ groups, where Γ is a "correlation factor." The instance has the following properties, for each element $j \in [m]$. For all $i \in [n]$, $\Pr[j \in D_i]$ is a fixed value p. For two subsets D_i and $D_{i'}$ in different groups, membership of j in D_i is independent of its membership in $D_{i'}$. For two subsets D_i and $D_{i'}$ in the same group, the probability that j has the same membership status in D_i and $D_{i'}$ (i.e., is either in both subsets, or in neither), is a fixed value ρ. In our experiments, $m = 2n$, $\Gamma = 4$, $p = 0.3$, and $\rho = 0.7858$. In Appendix A.3, we describe in detail how we generated the instance.

Min-Sum Facility Location

We use the locations of n Citi Bike stations [1] in New York City as the facilities for our facility location data set. For calculating the utility to customers, we uniformly generate customer locations within the range of latitude and longitude of the stations. The value $M_{a,b}$ for customer b and station a is the inverse of the Euclidean distance between them.

References

1 Citi bike system data. `https://ride.citibikenyc.com/system-data`, 2021. Accessed: 2021-12-1.

2 Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

3 Daniel Antunes, Claire Mathieu, and Nabil H. Mustafa. Combinatorics of Local Search: An Optimal 4-Local Hall's Theorem for Planar Graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2017.8`.

4 Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 407–418, 2004.

5 Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

6 Mehrdad Ghadiri, Richard Santiago, and Bruce Shepherd. Beyond submodular maximization. *arXiv preprint*, 2019. `arXiv:1904.09216`.

7 Mehrdad Ghadiri, Richard Santiago, and Bruce Shepherd. A parameterized family of meta-submodular functions. *arXiv preprint*, 2020. `arXiv:2006.13754`.

8 Felix Happach, Lisa Hellerstein, and Thomas Lidbetter. A general framework for approximating min sum ordering problems. *INFORMS Journal on Computing*, 34(3):1437–1452, 2022.

9 Satoru Iwata, Prasad Tetali, and Pushkar Tripathi. Approximating minimum linear ordering problems. In *Proceedings of Approx-Random*, 2012.

10 Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*, pages 722–754. PMLR, 2021.

11 Rishabh Iyer, Ninad Khargonkar, Jeff Bilmes, and Himanshu Asnani. Generalized submodular information measures: Theoretical properties, examples, optimization algorithms, and applications. *IEEE Transactions on Information Theory*, 68(2):752–781, 2021.

12 Nitish Korula, Vahab Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018.

13 Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2014.

14 Gilad Kutiel and Dror Rawitz. Local Search Algorithms for Maximum Carpool Matching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2017.55.

15 Wenjun Li, Jianer Chen, and Jianxin Wang. Deeper local search for better approximation on maximum internal spanning trees. In *European Symposium on Algorithms*, pages 642–653. Springer, 2014.

16 Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In *International Conference on Database Theory*, pages 83–98. Springer, 2005.

17 Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. Technical report, Carnegie-Mellon University School of Computer Science, 2007.

18 Matthew J. Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Neural Information Processing Systems (NeurIPS)*, pages 1577–1584, 2008.

A Appendix

A.1 Prior Use of the Term "Min-Sum Submodular Cover"

The first proof that a greedy algorithm for Min-Sum Submodular Cover yields a 4-approximation was given by Streeter and Golovin. The proof appears both in a Technical Report [17] and an associated conference paper [18]. They actually gave their proof for a more general problem than the one we considered in this paper, in which $u : 2^{[n]\times\mathbb{R}} \to \mathbb{R}$, and the output is a sequence of pairs of the form $(i, \tau) \in [n] \times \mathbb{R}$. In their Technical Report, Streeter and Golovin used the name Min-Sum Submodular Cover to refer to the more general problem, but they did not use this name (nor any other) to refer to the problem in their conference paper. We opted to use the name Min-Sum Submodular Cover to refer to the problem we defined in this paper, as we believe this usage of the name is natural given the connection to Min-Sum Submodular Cover.

We note that the definition Streeter and Golovin gave for the more general problem is problematic as written. The greedy algorithm may not be well-defined for functions u that are non-zero for subsets of $[n] \times \mathbb{R}$ that include pairs (i, τ), where τ is infinitesimally small. However, the results in the paper are not dependent on allowing such u, and the problem with the definition can be fixed by restricting the domain of u.

A.2 Bounding the number of rounds in the proof of Theorem 3

We show using Fact 6 that Algorithm 1 yields a $(4+\epsilon)$-approximation in at most $O(n \log(\frac{d}{\epsilon}))$ rounds from a d-approximate permutation.

We introduce a recurrence relation $T(\ell) = aT(\ell-1) - b$ where $a = 2n/(2n-1)$ and $b = 4/(2n-1)$. We can derive this recurrence by setting $T(\ell) = M$ and $T(\ell-1) = M-(M-4)/2n$. Intuitively, ℓ is the number of iterations until we reach $(4+\epsilon)$ and so we set $T(0) = 4+\epsilon$. By repeatedly expanding $T(\ell)$, we get

$$T(\ell) = -b\sum_{i=0}^{\ell-1} a^i + a^\ell(4+\epsilon) = -b\frac{a^\ell - 1}{a-1} + a^\ell(4+\epsilon) = \epsilon\left(\frac{2n}{2n-1}\right)^\ell + 4$$

where the last equality follows by plugging in the values of a and b. We claim that a d-approximate permutation is at most $2n\log(\frac{d}{\epsilon})$ rounds from a $(4+\epsilon)$-approximation. We can verify this by evaluating

$$T\left(2n\log\left(\frac{d}{\epsilon}\right)\right) = \epsilon\left(\frac{2n}{2n-1}\right)^{2n\log(\frac{d}{\epsilon})} + 4 = d^{2n\log\left(\frac{2n}{2n-1}\right)} + 4.$$

Notice that

$$x\log\left(\frac{x}{x-1}\right) \geq 1 \iff e^x\left(\frac{x}{x-1}\right) \geq e$$

which is certainly true for $x > 2$. It follows that $T(2n\log(\frac{d}{\epsilon})) \geq d$ so local search converges in at most $2n\log(\frac{d}{\epsilon})$ rounds.

A.3 Generation of the Pipelined Set Cover Data Set

We generate the n subsets $D_1, \ldots, D_n$ of $[m]$ randomly as follows. Initially, for each group G of subsets, we generate an advice bit $a_{G,j}$ for each element $j \in [m]$, which is True with probability p, and False with probability $1-p$. Then for each element j, and for each D_i in group G, we do the following: with probability p' we use advice bit $a_{G,j}$ to determine whether or not to include j in D_i (if $a_{G,j}$ is True, we include j in D_i, else we do not). With probability $(1-p')$, we ignore the advice bit, and instead, we include j in D_i with probability p, and exclude it with probability $(1-p)$. The probability that j is in a given set D_j is clearly p.

For $i \neq i'$, if subsets D_i and $D_{i'}$ are in different groups, then membership of an element j in D_i is clearly independent of its membership of $D_{i'}$. If D_i and $D_{i'}$ are subsets in the same group G, then the probability that j has the same membership status in both subsets can be calculated by noting that this can happen in two ways: either the advice bit $a_{G,j}$ was used to determine membership in both subsets, or $a_{G,j}$ was ignored for one or both of the two subsets and membership ended up being the same in both subsets. The first event happens with probability $p' \cdot p'$. The second happens with probability $(1-p' \cdot p')(p \cdot p + (1-p) \cdot (1-p))$. Because $p = 0.3$ and $p' = 0.7$, the probability that the membership status of j is the same for both subsets is .7858.

We note that it is possible that this process results in subsets D_i and $D_{i'}$ where $i \neq i'$ and $D_i = D_{i'}$. We do not eliminate such duplicates.

Algorithms for Coloring Reconfiguration Under Recolorability Digraphs

Soichiro Fujii
Research Institute for Mathematical Sciences, Kyoto University, Japan
School of Mathematical and Physical Sciences, Macquarie University, Sydney, Australia

Yuni Iwamasa
Graduate School of Informatics, Kyoto University, Japan

Kei Kimura
Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

Akira Suzuki
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Abstract

In the k-Recoloring problem, we are given two (vertex-)colorings of a graph using k colors, and asked to transform one into the other by recoloring only one vertex at a time, while at all times maintaining a proper k-coloring. This problem is known to be solvable in polynomial time if $k \leq 3$, and is PSPACE-complete if $k \geq 4$. In this paper, we consider a (directed) recolorability constraint on the k colors, which forbids some pairs of colors to be recolored directly. The recolorability constraint is given in terms of a digraph $\overrightarrow{R}$, whose vertices correspond to the colors and whose arcs represent the pairs of colors that can be recolored directly. We provide algorithms for the problem based on the structure of recolorability constraints $\overrightarrow{R}$, showing that the problem is solvable in linear time when $\overrightarrow{R}$ is a directed cycle or is in a class of multitrees.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases combinatorial reconfiguration, graph coloring, recolorability, recoloring

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.4

Funding *Soichiro Fujii*: Partially supported by JST, ERATO Grant Number JPMJER1603 (HASUO Metamathematics for Systems Design Project) and JSPS Overseas Research Fellowships.
Yuni Iwamasa: Partially supported by JSPS KAKENHI Grant Numbers JP20K23323, JP20H05795, 22K17854, Japan.
Kei Kimura: Partially supported by JSPS KAKENHI Grant Numbers JP19K22841, JP21K17700, Japan.
Akira Suzuki: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091, JP20K11666, JP20H05794, Japan.

1 Introduction

In a *combinatorial reconfiguration problem*, we are given two feasible solutions of a search problem, and asked whether there exists a step-by-step transformation (satisfying a given *reconfiguration rule*) between them, such that all intermediate solutions are also feasible. In other words, our task is to check the reachability between two given feasible solutions in a *reconfiguration (di)graph*, a (di)graph whose vertices correspond to the feasible solutions and whose arcs to the one-step transformations. (From this perspective, the traditional search problem is about deciding whether the reconfiguration (di)graph is empty or not.) Since the framework of combinatorial reconfiguration was developed by Ito et al. in 2008 [11, 12], combinatorial reconfiguration problems have been actively studied in theoretical computer science. We refer the reader to the surveys by van den Heuvel [10] and Nishimura [17] for more information on combinatorial reconfiguration in general.

One of the most fundamental reconfiguration problems is k-Recoloring. In this problem, we are given two k-colorings α and β of a graph G, and asked to determine whether there exists a sequence $(\alpha = \gamma_0, \gamma_1, \ldots, \gamma_\ell = \beta)$ of k-colorings of G such that for each $i \in \{1, 2, \ldots, \ell\}$, γ_i is obtained from γ_{i-1} by recoloring a single vertex in G.

The computational complexity of k-Recoloring has been studied with respect to various parameters such as the number k of colors [3, 4, 14] and the input graph classes [1, 9, 23]. In particular, it is known that k-Recoloring is polynomial-time solvable if $k \leq 3$ [4, 14] and is PSPACE-complete if $k \geq 4$ [3]. (On the other hand, k-Coloring, which asks to determine if a given graph admits a k-coloring, is polynomial-time solvable if $k \leq 2$ and is NP-complete if $k \geq 3$ [8].) While some of these parameters also appear in traditional search problems, others do not. In order to analyze the problem from a viewpoint specific to reconfiguration, Osawa et al. [19] introduced an undirected recolorability constraint for k-Recoloring, and Osawa [18] further considered its directed variant. More precisely, let $C = \{0, 1, \ldots, k-1\}$ be the color set for k-Recoloring. We define a **recolorability digraph** on C as a digraph $\overrightarrow{R} = (V(\overrightarrow{R}), A(\overrightarrow{R}))$ with $V(\overrightarrow{R}) = C$; the idea is that each arc in $A(\overrightarrow{R})$ represents a pair of colors allowed to be recolored directly. Then k-Recoloring under the recolorability constraint $\overrightarrow{R}$ is formulated as follows.

k-Recoloring Under $\overrightarrow{R}$-Recolorability

Given A graph G and k-colorings α and β of G.

Problem Determine if there exists a sequence $\mathcal{W} = (\alpha = \gamma_0, \gamma_1, \ldots, \gamma_\ell = \beta)$ of k-colorings of G such that for each $i \in \{1, 2, \ldots, \ell\}$, there exists $v \in V(G)$ with $(\gamma_{i-1}(v), \gamma_i(v)) \in A(\overrightarrow{R})$ and for all $w \in V(G) \setminus \{v\}$, $\gamma_{i-1}(w) = \gamma_i(w)$ (such a sequence $\mathcal{W}$ is called an (α, β)-**reconfiguration sequence**).

Throughout this paper, we assume for simplicity that the input graph G is connected; otherwise, we can solve the problem in each connected component of G separately. For an undirected graph $R = (V(R), E(R))$ with $V(R) = C$ (which we call a **recolorability graph**), we similarly define k-Recoloring Under R-Recolorability by replacing "$(\gamma_{i-1}(v), \gamma_i(v)) \in A(\overrightarrow{R})$" in the above definition by "$\{\gamma_{i-1}(v), \gamma_i(v)\} \in E(R)$." Notice that k-Recoloring is equivalent to k-Recoloring Under K_k-Recolorability, where K_k denotes the undirected complete graph with k vertices.

Several (in)tractability results in terms of the recolorability constraint have been obtained in [19, 20, 18]. In [19], Osawa et al. showed that k-Recoloring Under R-Recolorability is PSPACE-complete if the undirected recolorability graph R contains a connected component having more than one cycle or a vertex of degree at least four. The same authors showed in [20] that this problem can be solved in polynomial time if the maximum degree of R is at most two. Since k-Recoloring is equivalent to k-Recoloring Under K_k-Recolorability, these results generalize the classification of the computational complexity of k-Recoloring in terms of the number k of colors; PSPACE-complete if $k \geq 4$ (in this case K_k has more than one cycle) and solvable in polynomial time if $k \leq 3$ (in this case the maximum degree of K_k is at most two).

For a directed recolorability constraint, Osawa [18] showed that k-Recoloring Under $\overrightarrow{R}$-Recolorability can be NP-hard even when $\overrightarrow{R}$ is a polytree (i.e., a directed graph whose underlying undirected graph is a tree), and solvable in polynomial time when $\overrightarrow{R}$ is an arborescence (i.e., a directed rooted tree). However, most of the previous results are for undirected recolorability R, and it is fair to say that the corresponding problems for directed recolorability $\overrightarrow{R}$ are not well-investigated.

Our contribution

We provide two linear-time algorithms for k-Recoloring Under $\overrightarrow{R}$-Recolorability in the cases where

1. $\overrightarrow{R}$ is a directed cycle; and
2. $\overrightarrow{R}$ is a multitree (i.e., a directed acyclic graph (DAG) in which there exists at most one directed path between any two vertices) satisfying the condition (S) stated in Section 4.

The former result is the directed analogue of the tractability of k-Recoloring Under R-Recolorability when R is an undirected cycle [20], and the latter is a generalization of the tractability of k-Recoloring Under $\overrightarrow{R}$-Recolorability when $\overrightarrow{R}$ is an arborescence [18].

When $\overrightarrow{R}$ is a directed cycle, the key to our algorithm is the notion of *potential function* on the set of vertices induced by a reconfiguration sequence. We rephrase the existence of an (α, β)-reconfiguration sequence as the existence of a suitable potential function (Theorem 3), which in turn is characterized as a nonnegative integer solution of a system of linear equations, whose coefficient matrix is the incidence matrix of the input graph G (endowed with edge orientation). This implies that k-Recoloring Under $\overrightarrow{R}$-Recolorability is solvable in linear time. We also introduce a similar potential function in the case where $\overrightarrow{R}$ is an *undirected* cycle, and characterize the existence of an (α, β)-reconfiguration sequence as the existence of an integer solution to the same system of linear equations as in the directed case (Theorem 8). These characterizations clarify the relationship of the problems when $\overrightarrow{R}$ is a directed cycle and those when $\overrightarrow{R}$ is an undirected cycle. Any yes-instance (G, α, β) for the former is also a yes instance for the latter, and the difference is precisely the additional requirement of nonnegativity (for the corresponding potential function) in the former. We also shed new light on a criterion of the existence of an (α, β)-reconfiguration sequence given in [20, Section 4.2] in terms of the system of linear equations. We can compute a shortest reconfiguration sequence in linear time when $\overrightarrow{R}$ is a directed cycle, as in the case of undirected cycles.

We then show that k-Recoloring Under $\overrightarrow{R}$-Recolorability is also solvable in linear time when $\overrightarrow{R}$ is a multitree satisfying condition (S). This generalizes the tractability of k-Recoloring Under $\overrightarrow{R}$-Recolorability when $\overrightarrow{R}$ is an arborescence shown in [18]. It might be worth remarking that a certain aspect of the algorithm in [18] is preserved in this more general setting: for any yes-instance, we can always find a reconfiguration sequence in which once we change the color of some vertex in G, we keep changing the color of that vertex until we reach the target color (see Remark 14). In this case, any reconfiguration sequence has the same length and a shortest reconfiguration sequence can be obtained trivially.

Related work

One of the features of this study is that the reconfiguration rule is asymmetric, that is, even when a solution can be reached from another solution in one step, the reverse transformation is not available in general. The computational complexity of reconfiguration problems with asymmetric reconfiguration rules (that is, with a directed reconfiguration graph) has been studied since at least late 1990s. For example, Sokoban [7] is a well-known example of such a reconfiguration problem. In recent years, research on this kind of reconfiguration problems has grown rapidly; in particular, Demaine et al. [5, 6] introduced a useful tool to analyze the computational complexity of such problems in 2022. Ito et al. [13] considered an asymmetric version of the independent set reconfiguration under the token sliding rule. We note that an auxiliary digraph used in Section 4 is similar to the one introduced in [13].

k-Recoloring Under $\overrightarrow{R}$-Recolorability is related to the multi-agent path finding problem studied in motion planning and multi-agent systems (see, e.g., [22] for a survey on the problem). In a variant of the multi-agent path finding problem, we are given a digraph $\overrightarrow{R}$ and k agents, where each agent has a start vertex and a goal vertex in $\overrightarrow{R}$. Then we are asked to assign a directed walk in $\overrightarrow{R}$ to each agent that does not cause collisions on the vertices among the agents, in such a way that the sum of the time steps required for every agent to reach its goal vertex is minimized. (It is assumed that passing through an arc of $\overrightarrow{R}$ takes a unit time, and that agents can stay at vertices to avoid collision.) This problem is similar to k-Recoloring Under $\overrightarrow{R}$-Recolorability with the input graph K_k, where each vertex of K_k corresponds to an agent and a coloring $\alpha\colon V(K_k) \to V(\overrightarrow{R})$ corresponds to a configuration of the agents on $\overrightarrow{R}$. The coloring constraint (i.e., adjacent vertices must have different colors) corresponds to the requirement that the agents do not collide on the vertices. While each agent can move simultaneously in the multi-agent path finding problem, agents can only move one by one in k-Recoloring Under $\overrightarrow{R}$-Recolorability. Therefore, any reconfiguration sequence in k-Recoloring Under $\overrightarrow{R}$-Recolorability gives an upper bound on the above variant of the multi-agent path finding problem.

Preliminaries

For a graph G, we denote its vertex set by $V(G)$ and its edge set by $E(G)$. An edge incident to vertices v and w is written as vw or $\{v, w\}$. For a digraph $\overrightarrow{H}$, we denote its vertex set by $V(\overrightarrow{H})$ and its arc set by $A(\overrightarrow{H})$. An arc from v to w is written as (v, w). All graphs and digraphs considered in this paper are finite and simple.

Let C be a finite set of colors, and $\overrightarrow{R}$ a recolorability digraph on C (defined above). For $\overrightarrow{R}$ and a graph G, we define the **$\overrightarrow{R}$-reconfiguration digraph** on G, denoted by $\mathcal{C}_{\overrightarrow{R}}(G)$, as follows. A vertex of $\mathcal{C}_{\overrightarrow{R}}(G)$ is a coloring of G with respect to the color set C, i.e., a function $\alpha\colon V(G) \to C$ such that $\alpha(v) \neq \alpha(w)$ holds whenever $vw \in E(G)$. Given two colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$, there is an arc (α, β) in $\mathcal{C}_{\overrightarrow{R}}(G)$ if and only if there exists $v \in V(G)$ such that $(\alpha(v), \beta(v)) \in A(\overrightarrow{R})$ and $\alpha(w) = \beta(w)$ for each $w \in V(G) \setminus \{v\}$. See Figure 1 for an example of $\mathcal{C}_{\overrightarrow{R}}(G)$. A directed walk in $\mathcal{C}_{\overrightarrow{R}}(G)$ from α to β is called an **(α, β)-reconfiguration sequence**. We denote by $\mathsf{dist}(\alpha, \beta)$ the length of a shortest (α, β)-reconfiguration sequence; when there is no (α, β)-reconfiguration sequence, we define $\mathsf{dist}(\alpha, \beta) = +\infty$.

Organization

The remainder of this paper is organized as follows. Section 2 considers the case where $\overrightarrow{R}$ is a directed cycle. Section 3 provides a novel view to the case where $\overrightarrow{R}$ is an undirected cycle based on the results in Section 2. Section 4 deals with the case where $\overrightarrow{R}$ is a multitree satisfying condition (S).

Due to space limitation, the proofs of the statements marked with ($\star$) are deferred to the appendix.

2 Directed cycle

In this section, we consider k-Recoloring Under $\overrightarrow{R}$-Recolorability when $\overrightarrow{R}$ is a directed cycle, and show that it is solvable in linear time. Throughout this section, we fix a natural number $k \geq 3$, the k-element set $C = \{0, 1, \ldots, k-1\}$ and the recolorability digraph

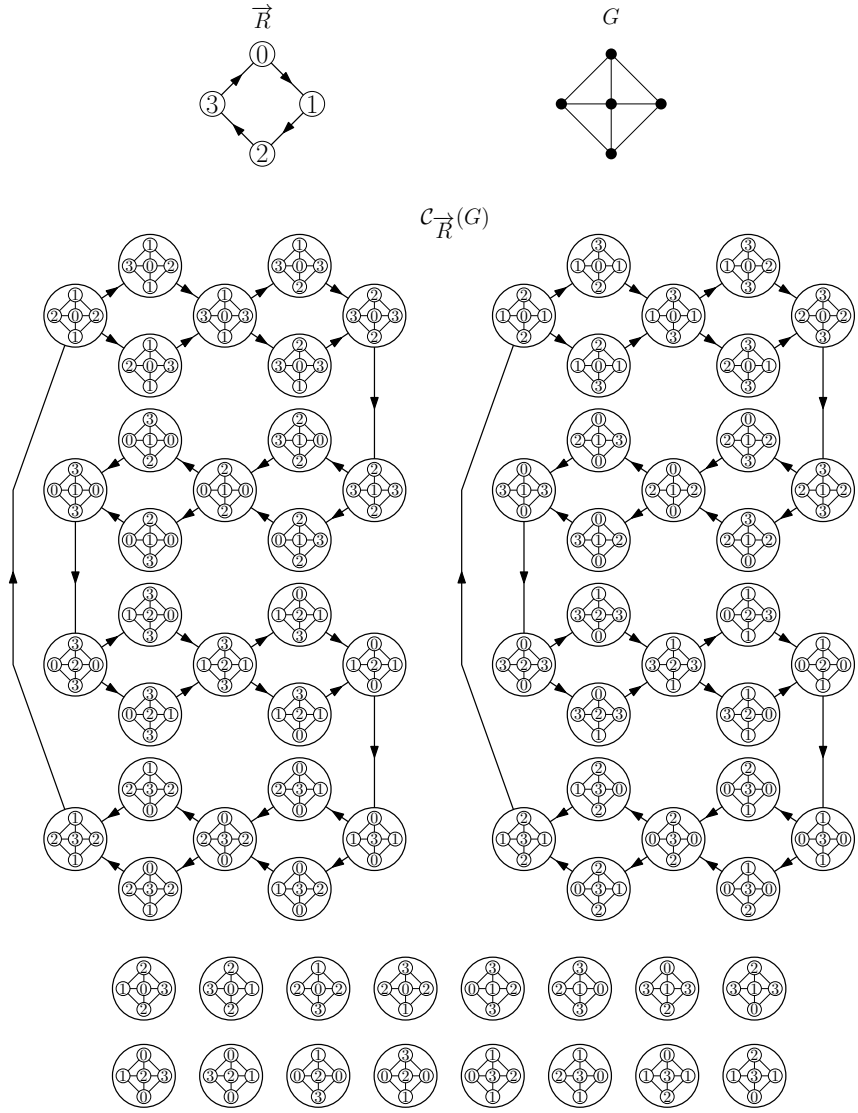


Figure 1 Examples of $\overrightarrow{R}$, G, and $\mathcal{C}_{\overrightarrow{R}}(G)$.

$\overrightarrow{R}$ on C whose arc set is given by $A(\overrightarrow{R}) = \{(0,1),(1,2),\dots,(k-1,0)\}$. For each $c,d \in C$, we denote by $\Delta(c,d)$ the length of the shortest (directed) (c,d)-path in $\overrightarrow{R}$; in other words, $\Delta(c,d)$ is the unique natural number in $\{0,1,\dots,k-1\}$ satisfying

$$c + \Delta(c,d) = d \pmod{k}. \tag{1}$$

Notice that whenever $c \neq d$, we have $\Delta(c,d) = k - \Delta(d,c)$. For each $c \in C$, define $c^+ \in C$ to be the (unique) color satisfying $\Delta(c,c^+) = 1$.

Given a coloring $\alpha \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ of a graph G, we say that a vertex $v \in V(G)$ is

- **blocked** with respect to α if for any $\beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ with $(\alpha,\beta) \in A(\mathcal{C}_{\overrightarrow{R}}(G))$, we have $\alpha(v) = \beta(v)$; and
- **frozen** with respect to α if for any $\beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ with $\mathsf{dist}(\alpha,\beta) < +\infty$, we have $\alpha(v) = \beta(v)$.

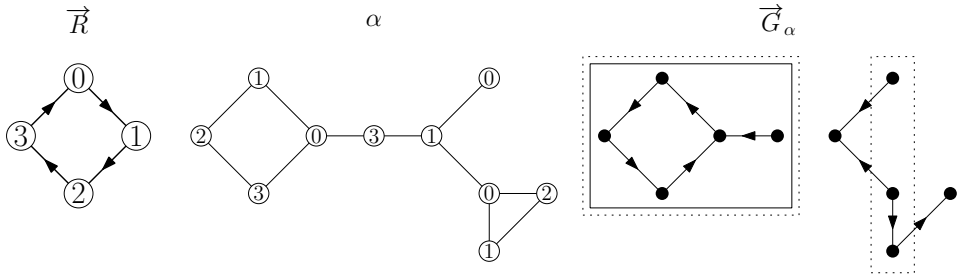


Figure 2 Examples of $\overrightarrow{R}$, α, and $\overrightarrow{G}_\alpha$. The vertices in the dotted rectangles are the blocked vertices, and those in the solid rectangle are the frozen ones.

Intuitively, v is blocked with respect to α if v cannot be recolored *directly* from α, whereas v is frozen with respect to α if v cannot be recolored by any successive recoloring starting from α.

Let G be a graph and $\alpha \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be a coloring, and suppose that a vertex $v \in V(G)$ is blocked with respect to α. Then, since the only color to which v can be recolored is $\alpha(v)^+$, it follows that there exists $w \in V(G)$ such that $vw \in E(G)$ and $\alpha(w) = \alpha(v)^+$. In order to represent this blocking relation, we define the **blocking digraph** $\overrightarrow{G}_\alpha$ on α as $V(\overrightarrow{G}_\alpha) = V(G)$ and for each $v, w \in V(G)$, $(v, w) \in A(\overrightarrow{G}_\alpha)$ if and only if $vw \in E(G)$ and $\alpha(w) = \alpha(v)^+$.[1] Clearly, a vertex $v \in V(G)$ is blocked with respect to α if and only if the outdegree of v in $\overrightarrow{G}_\alpha$ is positive. Notice that any vertex of G which is in a directed cycle in $\overrightarrow{G}_\alpha$, is frozen with respect to α. (More precisely, a vertex $v \in V(G)$ is frozen with respect to α if and only if there exists a directed path from v to a directed cycle in $\overrightarrow{G}_\alpha$.) See Figure 2 for an example of $\overrightarrow{G}_\alpha$.

Any directed walk $\mathcal{W} = (\gamma_0, \gamma_1, \ldots, \gamma_\ell)$ in $\mathcal{C}_{\overrightarrow{R}}(G)$ induces the **potential function**[2] $p^{\mathcal{W}} \colon V(G) \to \mathbb{N}$ associated with $\mathcal{W}$, mapping each vertex $v \in V(G)$ to

$$p^{\mathcal{W}}(v) = |\{\, i \in \{1, 2, \ldots, \ell\} \mid \gamma_{i-1}(v) \neq \gamma_i(v) \,\}|.$$

So $p^{\mathcal{W}}(v)$ is the number of arcs in $\mathcal{W}$ where the vertex v is recolored. Note that we have $\sum_{v \in V(G)} p^{\mathcal{W}}(v) = \ell$.

The following lemma establishes a crucial property of potential functions; cf. [24, Lemma 4.1].

▶ **Lemma 1.** *Let G be a graph and $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings. Given any (α, β)-reconfiguration sequence $\mathcal{W} = (\alpha = \gamma_0, \gamma_1, \ldots, \gamma_\ell = \beta)$ and any pair of vertices $v, w \in V(G)$ such that $vw \in E(G)$, we have*

$$\Delta(\alpha(v), \alpha(w)) + p^{\mathcal{W}}(w) = p^{\mathcal{W}}(v) + \Delta(\beta(v), \beta(w)).$$

Proof. For each $i \in \{0, 1, \ldots, \ell\}$, let $\mathcal{W}_i = (\gamma_0, \gamma_1, \ldots, \gamma_i)$. We show that

$$\Delta(\gamma_0(v), \gamma_0(w)) + p^{\mathcal{W}_i}(w) = p^{\mathcal{W}_i}(v) + \Delta(\gamma_i(v), \gamma_i(w)) \quad (\forall vw \in E(G)) \tag{2}$$

holds for each $i \in \{0, 1, \ldots, \ell\}$, by induction on i.

[1] The digraph $\overrightarrow{G}_\alpha$ is the same as $\overrightarrow{H}_\alpha$ in [20, Section 4.1], although $\overrightarrow{H}_\alpha$ is defined for the case where the recolorability graph is an undirected cycle.

[2] We remark that this notion is different from the notion of *potential of an (oriented) edge in G with respect to a coloring* introduced in [20, Section 4.2]. We have decided to adopt the current terminology since it seems to be more consistent with the usage of the term "potential" in the literature (see, e.g., [2, Chapter 20] and [16, Chapter 9]).

When $i = 0$, $p^{\mathcal{W}_0}(v) = p^{\mathcal{W}_0}(w) = 0$ and hence both sides of (2) are equal.

Let $j \in \{1, 2, \ldots, \ell\}$ and assume that (2) holds for $i = j - 1$. Let $u \in V(G)$ be the vertex such that $\gamma_{j-1}(u) \neq \gamma_j(u)$. If $u \notin \{v, w\}$, then clearly (2) continues to hold for $i = j$. If $u = v$, then $p^{\mathcal{W}_j}(v) = p^{\mathcal{W}_{j-1}}(v) + 1$ and $\Delta(\gamma_j(v), \gamma_j(w)) = \Delta(\gamma_{j-1}(v), \gamma_{j-1}(w)) - 1$, while $p^{\mathcal{W}_j}(w) = p^{\mathcal{W}_{j-1}}(w)$; hence (2) holds for $i = j$. The remaining case $u = w$ is similar. ◀

▶ Remark 2. The equation in Lemma 1 can be rewritten as

$$p^{\mathcal{W}}(v) - p^{\mathcal{W}}(w) = \Delta(\alpha(v), \alpha(w)) - \Delta(\beta(v), \beta(w)) \quad (\forall vw \in E(G)). \tag{3}$$

We can interpret this equation using the notion of incidence matrix of a digraph. Let us arbitrarily choose an orientation of each edge of G and denote the resulting digraph by $\overrightarrow{G}$. Let $\mathbf{M} \in \mathbb{R}^{V(\overrightarrow{G}) \times A(\overrightarrow{G})}$ be the incidence matrix of $\overrightarrow{G}$, i.e.,

$$\mathbf{M}(v, e) = \begin{cases} 1 & \text{if } e = (v, w) \text{ for some } w \in V(G); \\ -1 & \text{if } e = (w, v) \text{ for some } w \in V(G); \\ 0 & \text{otherwise.} \end{cases}$$

If we regard the potential function $p^{\mathcal{W}}$ as the row vector $\mathbf{p}^{\mathcal{W}} \in \mathbb{R}^{V(\overrightarrow{G})}$ and define another row vector $\mathbf{\Delta}^{\alpha,\beta} \in \mathbb{R}^{A(\overrightarrow{G})}$ by $\mathbf{\Delta}^{\alpha,\beta}((v, w)) = \Delta(\alpha(v), \alpha(w)) - \Delta(\beta(v), \beta(w))$, then (3) can be written as $\mathbf{p}^{\mathcal{W}}\mathbf{M} = \mathbf{\Delta}^{\alpha,\beta}$. This implies that for the existence of an (α, β)-reconfiguration sequence, it is necessary that the vector $\mathbf{\Delta}^{\alpha,\beta}$ be in the row space of $\mathbf{M}$, also known as the **tension space** of $\overrightarrow{G}$ (see, e.g., [2, Chapter 20]). It turns out that if there are no frozen vertices with respect to α, then the latter condition is equivalent to the existence of an (α, β)-reconfiguration sequence; see Remark 6. ⌟

The following theorem is the key to our algorithm. It allows us to reduce the task of deciding the existence of an (α, β)-reconfiguration sequence to that of deciding the existence of a suitable function $p\colon V(G) \to \mathbb{N}$, which is straightforward.

▶ **Theorem 3.** *Let G be a graph. For colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ and a function $p\colon V(G) \to \mathbb{N}$, there exists an (α, β)-reconfiguration sequence $\mathcal{W}$ such that $p = p^{\mathcal{W}}$ if and only if the following conditions hold:*

(C1) *for each $v \in V(G)$ in a directed cycle in $\overrightarrow{G}_\alpha$, we have $p(v) = 0$;*
(C2) *for each $v \in V(G)$, we have $\alpha(v) + p(v) = \beta(v) \pmod k$; and*
(C3) *for each $vw \in E(G)$, we have $\Delta(\alpha(v), \alpha(w)) + p(w) = p(v) + \Delta(\beta(v), \beta(w))$.*

Proof. Condition (C1) is necessary since a vertex in a directed cycle in $\overrightarrow{G}_\alpha$ is frozen with respect to α, (C2) is clearly necessary, and (C3) is necessary by Lemma 1.

The sufficiency of conditions (C1)–(C3) is established by induction on $\sum_{v \in V(G)} p(v)$. The base case (i.e., when $p(v) = 0$ for all $v \in V(G)$) is clear by (C2). Assume that a function $p\colon V(G) \to \mathbb{N}$ with $\sum_{v \in V(G)} p(v) > 0$ and colorings α, β satisfying conditions (C1)–(C3) are given. We claim that there exists a vertex $w \in V(G)$ such that $p(w) > 0$ and that w is not blocked with respect to α, i.e., w has outdegree 0 in the blocking digraph $\overrightarrow{G}_\alpha$. To show this, let $V' \subseteq V(G)$ be the set consisting of all $v \in V(G)$ with $p(v) > 0$. Then there is no arc in $\overrightarrow{G}_\alpha$ from V' to $V(G) \setminus V'$. Indeed, $(v, w) \in A(\overrightarrow{G}_\alpha)$ implies $p(v) \leq p(w)$: by (C3) we have $\Delta(\alpha(v), \alpha(w)) + p(w) = p(v) + \Delta(\beta(v), \beta(w))$, whereas $\Delta(\alpha(v), \alpha(w)) = 1 \leq \Delta(\beta(v), \beta(w))$. So if there is no $w \in V(G)$ with $p(w) > 0$ such that w is not blocked with respect to α, then it follows that each vertex in the induced subgraph $\overrightarrow{G}_\alpha|_{V'}$ has positive outdegree. Therefore the digraph $\overrightarrow{G}_\alpha|_{V'}$ (which is nonempty by $\sum_{v \in V(G)} p(v) > 0$) contains a directed cycle, contradicting (C1).

Now that we know the existence of $w \in V(G)$ such that $p(w) > 0$ and that w is not blocked with respect to α, let us define $\alpha' \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ to be the coloring obtained from α by recoloring w (necessarily to $\alpha(w)^+$), and $p' \colon V(G) \to \mathbb{N}$ to be the function given by

$$p'(v) = \begin{cases} p(v) - 1 & \text{if } v = w; \\ p(v) & \text{otherwise.} \end{cases}$$

Since we have $(\alpha, \alpha') \in A(\mathcal{C}_{\overrightarrow{R}}(G))$, it now suffices to show that there exists an (α', β)-reconfiguration sequence $\mathcal{W}'$ such that $p' = p^{\mathcal{W}'}$. For this we can use the induction hypothesis, since the triple (α', β, p') again satisfies (C1)–(C3). (To see that (α', β, p') satisfies (C1), observe that the directed cycles in $\overrightarrow{G}_\alpha$ and $\overrightarrow{G}_{\alpha'}$ remain unchanged.) ◀

▶ Remark 4. Let $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings of a (nonempty and connected) graph G and $p \colon V(G) \to \mathbb{N}$ be a function satisfying (C3). Then the triple (α, β, p) satisfies (C2) if and only if there exists *some* $v \in V(G)$ satisfying $\alpha(v) + p(v) = \beta(v) \pmod k$. Indeed, using (C3) and (1), it follows that the set of all vertices $v \in V(G)$ for which $\alpha(v) + p(v) = \beta(v) \pmod k$ holds is a union of connected components of G, i.e., is either $\emptyset$ or $V(G)$. ⌟

Theorem 3 suggests Algorithm 1 for k-Recoloring Under $\overrightarrow{R}$-Recolorability. This algorithm checks the existence of a function $p \colon V(G) \to \mathbb{N}$ satisfying (C1)–(C3). If the output of Algorithm 1 is "yes", then the length $\mathsf{dist}(\alpha, \beta)$ of a shortest (α, β)-reconfiguration sequence is given by $\sum_{v \in V(G)} \overline{p}(v)$, and such a sequence can be obtained by repeatedly recoloring any non-blocked vertex $w \in V(G)$ with $\overline{p}(w) > 0$ (and then updating $\overline{p}$ as in the proof of Theorem 3) in a greedy manner.

Algorithm 1 Algorithm for k-Recoloring Under $\overrightarrow{R}$-Recolorability with $\overrightarrow{R}$ a directed cycle.

Input : A (connected) graph G and two colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$.
Output : "Yes" if there exists an (α, β)-reconfiguration sequence, and "no" otherwise.

Step 1. Choose a vertex $v_0 \in V(G)$ and define $\tilde{p}(v_0) = \Delta(\alpha(v_0), \beta(v_0))$.
Step 2. Extend $\tilde{p}$ to a function $\tilde{p} \colon V(G) \to \mathbb{Z}$ so that for each edge $vw \in E(G)$, $\Delta(\alpha(v), \alpha(w)) + \tilde{p}(w) = \tilde{p}(v) + \Delta(\beta(v), \beta(w))$ holds. (For example, one can first choose a spanning tree T of G and then extend the value of $\tilde{p}$ along edges in T using the equation. Then one checks if the equation holds for each edge in $E(G) \setminus E(T)$.) Output "no" if this is impossible.
Step 3. Let $a = \min\{j \in \mathbb{N} \mid \forall v \in V(G).\ 0 \leq \tilde{p}(v) + jk\}$ and define $\overline{p} \colon V(G) \to \mathbb{N}$ by $\overline{p}(v) = \tilde{p}(v) + ak$ for each $v \in V(G)$.
Step 4. Compute $\overrightarrow{G}_\alpha$. Output "yes" if for each $v \in V(G)$ which is in a directed cycle in $\overrightarrow{G}_\alpha$, we have $\overline{p}(v) = 0$; output "no" otherwise.

▶ **Theorem 5.** *For any directed cycle $\overrightarrow{R}$, k-Recoloring Under $\overrightarrow{R}$-Recolorability is solvable in $O(|V(G)| + |E(G)|)$ time.*

Proof. Since the correctness of Algorithm 1 follows from Theorem 3, we only estimate its running time. It is clear that Steps 1, 2, and 3 in Algorithm 1 can be done in $O(1)$, $O(|E(G)|)$, and $O(|V(G)|)$ time, respectively. In Step 4, $\overrightarrow{G}_\alpha$ can be computed in $O(|E(G)|)$ time by definition and the subsequent procedure can be done in $O(|V(\overrightarrow{G}_\alpha)| + |A(\overrightarrow{G}_\alpha)|) = O(|V(G)| + |E(G)|)$ time by computing the strongly connected components of $\overrightarrow{G}_\alpha$. Hence, the running time of Algorithm 1 is $O(|V(G)| + |E(G)|)$. ◀

▶ Remark 6. This remark is a continuation of Remark 2, and we use the notations introduced there. Given colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$, we shall give a necessary and sufficient condition for the existence of an (α, β)-reconfiguration sequence in terms of the vector $\mathbf{\Delta}^{\alpha,\beta}$.

First we consider the case where there is no directed cycle in $\overrightarrow{G}_\alpha$, i.e., no vertex is frozen with respect to α. In this case, *there exists an (α, β)-reconfiguration sequence if and only if $\mathbf{\Delta}^{\alpha,\beta}$ belongs to the tension space of $\overrightarrow{G}$*. We have already seen the necessity of the latter condition. To show its sufficiency, suppose that $\mathbf{\Delta}^{\alpha,\beta}$ is in the tension space of $\overrightarrow{G}$, i.e., the row space of the incidence matrix $\mathbf{M}$ of $\overrightarrow{G}$. Since $\mathbf{M}$ is totally unimodular (see, e.g., [15, Theorem 5.27]), it follows that there exists an integer vector $\tilde{\mathbf{p}} \in \mathbb{Z}^{V(\overrightarrow{G})}$ satisfying $\tilde{\mathbf{p}}\mathbf{M} = \mathbf{\Delta}^{\alpha,\beta}$ (see, e.g., [21, Section 19.1]). Now notice that the all-one (row) vector $\mathbf{1} = (1, 1, \ldots, 1) \in \mathbb{R}^{V(\overrightarrow{G})}$ is in the left kernel of $\mathbf{M}$ (i.e., $\mathbf{1}\mathbf{M} = \mathbf{0}$), and in fact spans the left kernel of $\mathbf{M}$ since G is assumed to be connected. Therefore, we can add a suitable natural number multiple of $\mathbf{1}$ to $\tilde{\mathbf{p}}$ to obtain $\mathbf{p} \in \mathbb{N}^{V(\overrightarrow{G})}$ which moreover satisfies (C2) (cf. Remark 4). Since (C1) is vacuous by our assumption on α, Theorem 3 guarantees the existence of an (α, β)-reconfiguration sequence.

When there is a directed cycle in $\overrightarrow{G}_\alpha$, choose a vertex $v \in V(G)$ that is in such a directed cycle. If $\mathbf{\Delta}^{\alpha,\beta}$ is in the tension space of $\overrightarrow{G}$, then it follows from the above discussion that there exists a unique (integer) vector $\mathbf{p} \in \mathbb{Z}^{V(\overrightarrow{G})}$ satisfying $\mathbf{p}\mathbf{M} = \mathbf{\Delta}^{\alpha,\beta}$ and $\mathbf{p}(v) = 0$. Then an (α, β)-reconfiguration sequence exists if and only if all components of $\mathbf{p}$ are nonnegative and $\mathbf{p}$ satisfies (C1) and (C2).

The tension space of $\overrightarrow{G}$ is the orthogonal complement of the **circulation space** of $\overrightarrow{G}$, the vector subspace of $\mathbb{R}^{A(\overrightarrow{R})}$ spanned by the (signed) characteristic vectors of the cycles in G (see, e.g., [2, Proposition 20.1]). A basis of the circulation space of $\overrightarrow{G}$ can be obtained by choosing a spanning tree T of G and taking the set of all characteristic vectors of the fundamental cycles with respect to T (see, e.g., [15, Theorem 2.11]).[3] The role of Step 2 of Algorithm 1 can be understood from this perspective; it decides whether $\mathbf{\Delta}^{\alpha,\beta}$ is in the tension space or not, by checking its orthogonality to the circulation space. ⌟

▶ Remark 7. Here, we estimate the diameter of $V(\mathcal{C}_{\overrightarrow{R}}(G))$, i.e., the value

$$\max\{\, \mathsf{dist}(\alpha, \beta) \mid \alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G)), \mathsf{dist}(\alpha, \beta) < +\infty \,\}.$$

Given colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$, the length of a shortest (α, β)-reconfiguration sequence, if it exists, is given by $\sum_{v \in V(G)} \overline{p}(v)$ from Algorithm 1. We have

$$\begin{aligned}\sum_{v \in V(G)} \overline{p}(v) &\leq (k-1)n + \sum_{i=1}^{n-1} (k-2)i \\ &= \frac{((k-2)n + k)\, n}{2},\end{aligned}$$

since the minimum value of $\{\overline{p}(v) \mid v \in V(G)\}$ is at most $k-1$ and $|\overline{p}(v) - \overline{p}(w)| \leq k-2$ for any adjacent vertices v and w. The value $((k-2)n + k)\, n/2$ is attained by the n-vertex path $P = (v_1, v_2, \ldots, v_n)$ with $\alpha = (0, 1, 2, \ldots, k-1, 0, 1, \ldots, r(n,k) - 1)$ and $\beta = (k-1, k-2, \ldots, 1, 0, k-1, \ldots, k - r(n,k))$, where $r(n,k)$ is the remainder of n divided by k. Hence, the diameter of $V(\mathcal{C}_{\overrightarrow{R}}(P))$ is $\Theta(kn^2)$ which matches that in the case where the recolorability graph is an undirected cycle.[4] ⌟

[3] The tension space and the circulation space of $\overrightarrow{G}$ are called the cocycle space and the cycle space of $\overrightarrow{G}$ in [15], respectively.

[4] The construction of graphs with the diameter of the reconfiguration graph $\Omega(n^2)$ when R is the undirected cycle with three vertices in [4] can be easily generalized to that with $\Omega(kn^2)$ when R is the undirected cycle with k vertices.

3 Undirected cycle

A linear-time algorithm for k-Recoloring Under R-Recolorability where R is an undirected cycle has been given in [20]. Here we shall establish an analogue of Theorem 3 and shed new light on results of [20]. Throughout this section, we fix a natural number $k \geq 3$, the k-element set $C = \{0, 1, \ldots, k-1\}$ and the (undirected) recolorability graph R on C whose set of edges is given by $E(R) = \{01, 12, \ldots, (k-1)0\}$. Notions for reconfiguration problems with undirected recolorability graph are explained in [20]. We may also regard R as a digraph with $A(R) = \{(0,1), (1,2), \ldots, (k-1,0)\} \cup \{(0,k-1), (k-1,k-2), \ldots, (1,0)\}$. Note that since R is undirected, the R-reconfiguration "digraph" $\mathcal{C}_R(G)$ is also undirected for any graph G: for $\alpha, \beta \in V(\mathcal{C}_R(G))$, we have $(\alpha, \beta) \in A(\mathcal{C}_R(G))$ if and only if $(\beta, \alpha) \in A(\mathcal{C}_R(G))$.

For each $c, d \in C$, we denote by $\Delta(c,d)$ the unique natural number $\Delta(c,d) \in \{0, 1, \ldots, k-1\}$ satisfying $c + \Delta(c,d) = d \pmod{k}$. Given $c \in C$, define $c^+, c^- \in C$ to be the (unique) colors satisfying $\Delta(c, c^+) = 1$ and $\Delta(c^-, c) = 1$. Notice that given a coloring $\alpha \in V(\mathcal{C}_R(G))$, a vertex $v \in V(G)$ can only be recolored (directly) to $\alpha(v)^+$ or $\alpha(v)^-$. It is convenient to refine the notion of blocked vertices as follows. We say that a vertex $v \in V(G)$ is

- **forward-blocked** with respect to α if for any $\beta \in V(\mathcal{C}_R(G))$ with $(\alpha, \beta) \in A(\mathcal{C}_R(G))$, we have $\alpha(v) = \beta(v)$ or $\alpha(v)^- = \beta(v)$; and
- **backward-blocked** with respect to α if for any $\beta \in V(\mathcal{C}_R(G))$ with $(\alpha, \beta) \in A(\mathcal{C}_R(G))$, we have $\alpha(v) = \beta(v)$ or $\alpha(v)^+ = \beta(v)$.

The **blocking digraph** $\overrightarrow{G}_\alpha$ on a coloring $\alpha \in V(\mathcal{C}_R(G))$ is defined as in Section 2: $V(\overrightarrow{G}_\alpha) = V(G)$ and for each $v, w \in V(G)$, $(v,w) \in A(\overrightarrow{G}_\alpha)$ if and only if $vw \in E(G)$ and $\alpha(w) = \alpha(v)^+$ (or equivalently $\alpha(v) = \alpha(w)^-$). Notice that $v \in V(G)$ is forward-blocked with respect to α if and only if the outdegree of v in $\overrightarrow{G}_\alpha$ is positive, whereas v is backward-blocked with respect to α if and only if the indegree of v in $\overrightarrow{G}_\alpha$ is positive.

Let G be a graph. A walk $\mathcal{W} = (\gamma_0, \gamma_1, \ldots, \gamma_\ell)$ in $\mathcal{C}_R(G)$ induces the **potential function** $p^{\mathcal{W}} \colon V(G) \to \mathbb{Z}$ associated with $\mathcal{W}$, defined for each vertex v as

$$p^{\mathcal{W}}(v) = |\{\, i \in \{1, 2, \ldots, \ell\} \mid \gamma_{i-1}(v)^+ = \gamma_i(v) \,\}| - |\{\, i \in \{1, 2, \ldots, \ell\} \mid \gamma_{i-1}(v)^- = \gamma_i(v) \,\}|.$$

Note that we have $\sum_{v \in V(G)} |p^{\mathcal{W}}(v)| \leq \ell$.

The following is the analogue of Theorem 3. The only difference is that now the function p is allowed to take negative integer values.

▶ **Theorem 8.** *Let G be a graph. For colorings $\alpha, \beta \in V(\mathcal{C}_R(G))$ and a function $p \colon V(G) \to \mathbb{Z}$, there exists an (α, β)-reconfiguration sequence $\mathcal{W}$ such that $p = p^{\mathcal{W}}$ if and only if conditions (C1)–(C3) of Theorem 3 hold. Moreover, if these conditions are satisfied, then there exists an (α, β)-reconfiguration sequence $\mathcal{W} = (\alpha = \gamma_0, \gamma_1, \ldots, \gamma_\ell = \beta)$ with $p = p^{\mathcal{W}}$ which moreover satisfies $\sum_{v \in V(G)} |p^{\mathcal{W}}(v)| = \ell$.*

Proof. This can be proved by an argument analogous to the proof of Theorem 3. One can show the sufficiency of conditions (C1)–(C3) by induction on $\sum_{v \in V(G)} |p(v)|$. In the induction step, we may assume that either

1. there exists $v \in V(G)$ with $p(v) > 0$; or
2. there exists $v \in V(G)$ with $p(v) < 0$.

Accordingly, we can show (using $(v,w) \in A(\overrightarrow{G}_\alpha) \implies p(v) \leq p(w)$) that either

1. there exists $w \in V(G)$ with $p(w) > 0$ and w is not forward-blocked with respect to α; or
2. there exists $w \in V(G)$ with $p(w) < 0$ and w is not backward-blocked with respect to α.

The rest of the proof is straightforward. ◀

We can modify Algorithm 1 to obtain Algorithm 2. If the output of Algorithm 2 is "yes" and there exists a directed cycle in $\overrightarrow{G}_\alpha$, then the length $\mathsf{dist}(\alpha, \beta)$ of a shortest (α, β)-reconfiguration path is given by $\sum_{v \in V(G)} |\tilde{p}(v)|$, and such a path can be obtained by repeatedly recoloring any non-blocked vertex $w \in V(G)$ with $\tilde{p}(w) \neq 0$. If the output of Algorithm 2 is "yes" but there is no directed cycle in $\overrightarrow{G}_\alpha$, then $\mathsf{dist}(\alpha, \beta)$ might be strictly smaller than $\sum_{v \in V(G)} |\tilde{p}(v)|$. To find the precise value of $\mathsf{dist}(\alpha, \beta)$, consider $\tilde{p}[j] \colon V(G) \to \mathbb{Z}$ defined by $\tilde{p}[j](v) = \tilde{p}(v) + jk$ for each $j \in \mathbb{Z}$. Then we have

$$\mathsf{dist}(\alpha, \beta) = \min_{j \in \mathbb{Z}} \Big(\sum_{v \in V} |\tilde{p}[j](v)| \Big).$$

This value can be easily computed using the convexity of the function $j \mapsto \sum_{v \in V} |\tilde{p}[j](v)|$.

Algorithm 2 Algorithm for k-Recoloring Under R-Recolorability with R an undirected cycle.

Input : A (connected) graph G and two colorings $\alpha, \beta \in V(\mathcal{C}_R(G))$.
Output : "Yes" if there exists an (α, β)-reconfiguration sequence, and "no" otherwise.

Step 1. Compute $\overrightarrow{G}_\alpha$. If $\overrightarrow{G}_\alpha$ contains a directed cycle, choose a vertex $v_0 \in V(G)$ in such a directed cycle; otherwise, choose $v_0 \in V(G)$ arbitrarily. Define $\tilde{p}(v_0) = \Delta(\alpha(v_0), \beta(v_0))$.
Step 2. Extend $\tilde{p}$ to a function $\tilde{p} \colon V(G) \to \mathbb{Z}$ so that for each edge $vw \in E(G)$, $\Delta(\alpha(v), \alpha(w)) + \tilde{p}(w) = \tilde{p}(v) + \Delta(\beta(v), \beta(w))$ holds. (For example, one can first choose a spanning tree T of G and then extend the value of $\tilde{p}$ along edges in T using the equation. Then one checks if the equation holds for each edge in $E(G) \setminus E(T)$.) Output "no" if this is impossible.
Step 3. Output "yes" if for each $v \in V(G)$ which is in a directed cycle in $\overrightarrow{G}_\alpha$, we have $\tilde{p}(v) = 0$; output "no" otherwise.

From Algorithm 2, we obtain the following theorem, which was first proved in [20].

▶ **Theorem 9.** *For any undirected cycle R, k-Recoloring Under R-Recolorability is solvable in $O(|V(G)| + |E(G)|)$ time.*

Proof. This can be proved in a similar way as Theorem 5. ◀

▶ Remark 10. As mentioned in Remarks 2 and 6, we can interpret condition (C3) via the incidence matrix $\mathbf{M} \in \mathbb{R}^{V(\overrightarrow{G}) \times A(\overrightarrow{G})}$ of a digraph $\overrightarrow{G}$ obtained from G by arbitrarily choosing an orientation of each edge. We can define the row vector $\mathbf{\Delta}^{\alpha,\beta} \in \mathbb{R}^{A(\overrightarrow{G})}$ from any pair of colorings $\alpha, \beta \in V(\mathcal{C}_R(G))$ as in Remark 2.

Let $\alpha, \beta \in V(\mathcal{C}_R(G))$ be colorings. If there is no directed cycle in $\overrightarrow{G}_\alpha$, then *there exists an (α, β)-reconfiguration sequence if and only if $\mathbf{\Delta}^{\alpha,\beta}$ belongs to the tension space of $\overrightarrow{G}$*. So in this case k-Recoloring Under R-Recolorability can be solved by checking whether $\mathbf{\Delta}^{\alpha,\beta}$ is orthogonal to the circulation space of $\overrightarrow{G}$. This explains the criterion for the existence of an (α, β)-reconfiguration sequence given in [20, Theorem 13] (or more precisely its special case where no frozen vertices are involved).[5] We can turn this observation into a linear-time

[5] Note that, as mentioned in footnote 2, the meaning of the term "potential" is different in [20] and in this paper.

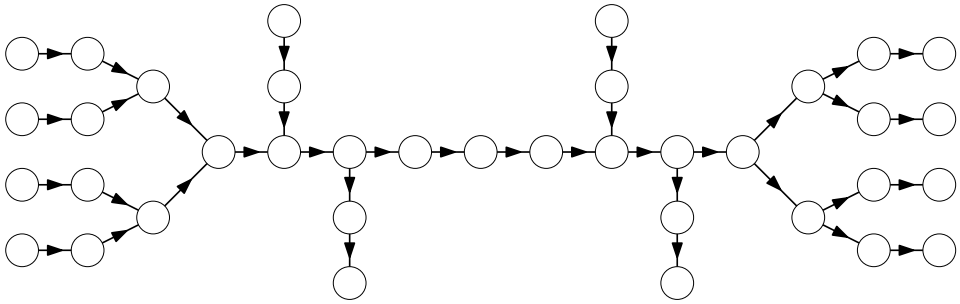


Figure 3 A polytree $\overrightarrow{R}$ such that k-Recoloring Under $\overrightarrow{R}$-Recolorability is NP-complete [18].

algorithm by using the fact that the basis of the circulation space is given by the set of all (signed) characteristic vectors of the fundamental cycles in G with respect to any spanning tree T of G; cf. [20, Lemma 17].

The case where $\overrightarrow{G}_\alpha$ contains directed cycles can be treated as in Remark 6. ⌟

▶ Remark 11. Let us examine how the length of a shortest reconfiguration sequence changes when the recolorability graph is changed from an undirected cycle R to a directed cycle $\overrightarrow{R}$ of the same size k. Namely, for any graph G and colorings $\alpha, \beta \in V(\mathcal{C}_R(G))$, we compare $d = \mathsf{dist}(\alpha, \beta)$ in $V(\mathcal{C}_R(G))$ with $d' = \mathsf{dist}(\alpha, \beta)$ in $V(\mathcal{C}_{\overrightarrow{R}}(G))$, provided that d' is finite. Assume that d satisfies $d = \sum_{v \in V} |\tilde{p}(v)|$ for some potential function $\tilde{p}$ of G. Let $M = -\min_{v \in V} \tilde{p}(v)$ and $j \in \mathbb{N}$ be such that $(j-1)k < M \leq jk$. Then from Algorithm 1 we have $d' = \sum_{v \in V} \tilde{p}'(v)$, where $\tilde{p}'(v) = \tilde{p}(v) + jk$ for each $v \in V$. By definition, we have $M \leq d$ and $jk < M + k$. Hence, we have

$$\begin{aligned} d' &\leq d + njk \\ &< d + n(M + k) \\ &\leq d + (d + k) \\ &= (n+1)d + nk. \end{aligned}$$

Using this inequality we can also obtain a multiplicative bound $d' < knd$, which is tight as seen from the following instance. Consider the n-vertex path $P = (v_1, v_2, \ldots, v_n)$, and its colorings $\alpha = (0, 1, 2, \ldots, k-1, 0, 1, \ldots, r(n,k)-1)$ and $\beta = (k-1, 1, 2, \ldots, k-1, 0, 1, \ldots, r(n,k)-1)$ (thus, β is obtained by changing only the color of v_1 from α), where $r(n,k)$ is the remainder of n divided by k. Then $d = 1$ and $d' = kn - 1$. Hence, the bound $d' < knd$ is tight. ⌟

4 A class of multitrees

In this section, we show that k-Recoloring Under $\overrightarrow{R}$-Recolorability is linear-time solvable whenever $\overrightarrow{R}$ is a multitree satisfying the condition (S) introduced below, where a **multitree** is a directed acyclic graph (DAG) $\overrightarrow{R}$ such that for each $c, d \in V(\overrightarrow{R})$, there exists at most one directed (c,d)-path in $\overrightarrow{R}$.

First observe that k-Recoloring Under $\overrightarrow{R}$-Recolorability is in NP whenever $\overrightarrow{R}$ is a DAG [18], since in this case the lengths of reconfiguration sequences in $\mathcal{C}_{\overrightarrow{R}}(G)$ are bounded by $|V(G)| \times |V(\overrightarrow{R})|$. For a DAG $\overrightarrow{R}$ and $c, d \in V(\overrightarrow{R})$, we write $c \leq d$ if there exists a (c,d)-path in $\overrightarrow{R}$. Then $\leq$ is a partial order on $V(\overrightarrow{R})$, and given any graph G and colorings $\alpha, \beta \in \mathcal{C}_{\overrightarrow{R}}(G)$, an obvious necessary condition for the existence of an (α, β)-reconfiguration sequence is *pointwise reachability*:

(PR) for each $v \in V(G)$, we have $\alpha(v) \leq \beta(v)$.

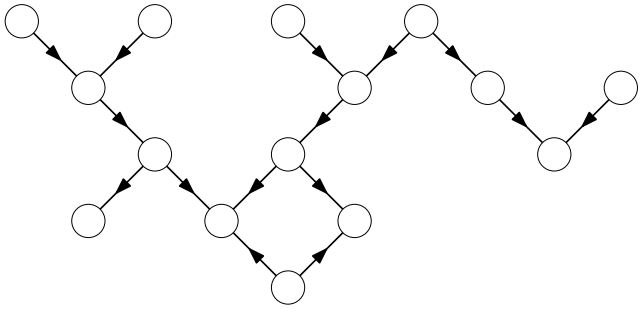


Figure 4 An example of a recolorability digraph satisfying (S) but not (S').

When $\overrightarrow{R}$ is moreover a multitree, for every $c, d \in V(\overrightarrow{R})$ such that $c \leq d$, the set of all vertices in the unique directed (c,d)-path in $\overrightarrow{R}$ is given by $[c,d] = \{e \in V(\overrightarrow{R}) \mid c \leq e \leq d\}$. To see its implication to the reconfiguration problem, let $\alpha, \beta \in \mathcal{C}_{\overrightarrow{R}}(G)$ be colorings satisfying (PR). Then we can tell in advance the sequence of colors on individual vertices $v \in V(G)$ in an (α,β)-reconfiguration sequence (if any): for each (α,β)-reconfiguration sequence $\mathcal{W} = (\alpha = \gamma_0, \gamma_1, \dots, \gamma_\ell = \beta)$ and $v \in V(G)$, we have

$$\{\, c \in V(\overrightarrow{R}) \mid \exists i \in \{0,1,\dots,\ell\}.\, \gamma_i(v) = c \,\} = [\alpha(v), \beta(v)].$$

In particular, the lengths of all (α,β)-reconfiguration sequences are the same, and is given by

$$\sum_{v \in V(G)} \big(\big|[\alpha(v), \beta(v)]\big| - 1\big).$$

Although the above discussion simplifies the situation greatly, Osawa [18] constructed a multitree $\overrightarrow{R}$ for which k-Recoloring Under $\overrightarrow{R}$-Recolorability is NP-complete; see Figure 3. (In fact this $\overrightarrow{R}$ is a *polytree*, i.e., a multitree whose underlying (undirected) graph is a tree.) Roughly speaking, the source of difficulty lies in the fact that it might be necessary to change the color of some vertex v of G a bit, but then "wait" in the middle of a reconfiguration sequence in order to "let colors on adjacent vertices pass" before reaching the final color of v; see the proof of [18, Theorem 14] for details.

Then a natural question is whether k-Recoloring Under $\overrightarrow{R}$-Recolorability is solvable in polynomial time if the multitree $\overrightarrow{R}$ prohibits such a phenomenon (cf. Remark 14). In our first attempt to find a sufficient condition on such multitrees $\overrightarrow{R}$, we were lead to the following:

(S') Let $(c_0, c_1, \dots, c_\ell)$ be a directed path in $\overrightarrow{R}$ and $i, j \in \{0,1,\dots,\ell\}$. If c_i has indegree > 1 and c_j has outdegree > 1 in $\overrightarrow{R}$, then $i \leq j$.

Intuitively, this says that no "merging" occurs after "splitting" in $\overrightarrow{R}$.

It then turned out that we can improve our result, and show (using condition (S') in the proof) that k-Recoloring Under $\overrightarrow{R}$-Recolorability is linear-time solvable whenever $\overrightarrow{R}$ is a multitree satisfying the following slightly more general condition:

(S) Let $(c_0, c_1, \dots, c_\ell)$ be any directed path in $\overrightarrow{R}$ and $i, j \in \{0,1,\dots,\ell\}$. If c_i has indegree > 1 and c_j has outdegree > 1, then either $i \leq j$, $i = \ell$, or $j = 0$.

See Figure 4 for an example of a recolorability digraph satisfying (S) but not (S'). (A multitree $\overrightarrow{R}$ satisfies (S) if and only if it does not contain a subgraph $\overrightarrow{R}'$ as in Figure 6 in the appendix.)

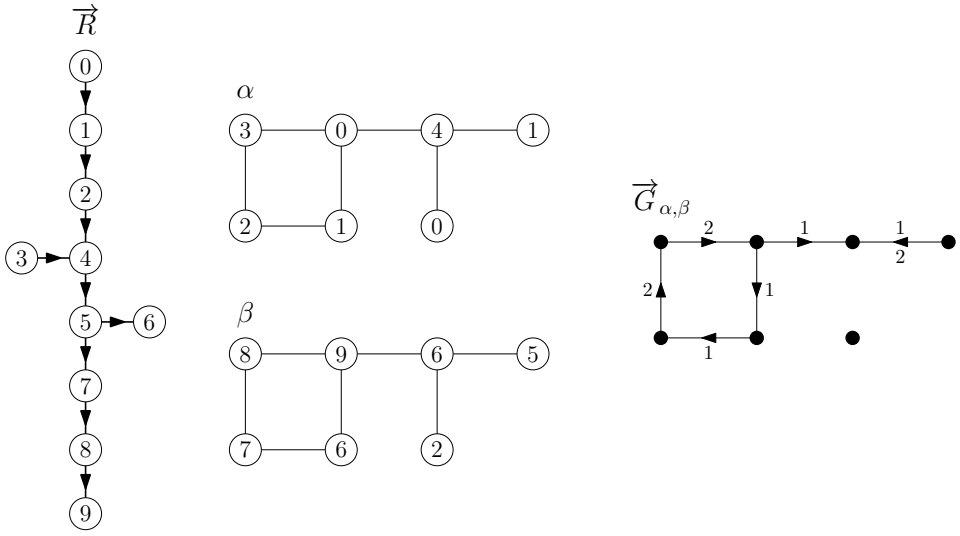


Figure 5 Examples of $\overrightarrow{R}$, α, β, and $\overrightarrow{G}_{\alpha,\beta}$. In $\overrightarrow{G}_{\alpha,\beta}$, the arcs from $A(\overrightarrow{G}^1_{\alpha,\beta})$ (resp. $A(\overrightarrow{G}^2_{\alpha,\beta})$) are labeled by 1 (resp. 2).

Throughout the rest of this section, let $\overrightarrow{R}$ be a multitree satisfying (S) unless otherwise stated. The outline of our linear-time algorithm for k-Recoloring Under $\overrightarrow{R}$-Recolorability is as follows. It first checks whether (PR) holds for the input data (G, α, β). If (G, α, β) satisfies (PR), then we construct an auxiliary digraph $\overrightarrow{G}_{\alpha,\beta}$. Now Theorem 12 below guarantees that we can obtain the answer for the instance (G, α, β) by checking whether $\overrightarrow{G}_{\alpha,\beta}$ contains a directed cycle or not.

We proceed to the definition of the auxiliary digraph $\overrightarrow{G}_{\alpha,\beta}$; see also Figure 5. Let G be a graph and $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings satisfying (PR). First define a digraph $\overrightarrow{G}^1_{\alpha,\beta}$ by $V(\overrightarrow{G}^1_{\alpha,\beta}) = V(G)$ and for each $v, w \in V(G)$, $(v, w) \in A(\overrightarrow{G}^1_{\alpha,\beta})$ if and only if $vw \in E(G)$ and $\alpha(w) \in [\alpha(v), \beta(v)]$. Similarly, define a digraph $\overrightarrow{G}^2_{\alpha,\beta}$ by $V(\overrightarrow{G}^2_{\alpha,\beta}) = V(G)$ and for each $v, w \in V(G)$, $(v, w) \in A(\overrightarrow{G}^2_{\alpha,\beta})$ if and only if $vw \in E(G)$ and $\beta(v) \in [\alpha(w), \beta(w)]$. Finally we define the digraph $\overrightarrow{G}_{\alpha,\beta}$ as the union of $\overrightarrow{G}^1_{\alpha,\beta}$ and $\overrightarrow{G}^2_{\alpha,\beta}$: $V(\overrightarrow{G}_{\alpha,\beta}) = V(G)$ and $(v, w) \in A(\overrightarrow{G}_{\alpha,\beta})$ if and only if $(v, w) \in A(\overrightarrow{G}^1_{\alpha,\beta})$ or $(v, w) \in A(\overrightarrow{G}^2_{\alpha,\beta})$. Intuitively, that (v, w) is an arc in $\overrightarrow{G}_{\alpha,\beta}$ means that w is an obstacle for recoloring v from $\alpha(v)$ to $\beta(v)$. We remark that similar auxiliary digraphs are also used in [13].

▶ **Theorem 12** (⋆). *Let G be a graph and $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings. There exists an (α, β)-reconfiguration sequence if and only if (PR) and the following condition hold:*
(DAG) *the digraph $\overrightarrow{G}_{\alpha,\beta}$ does not contain a directed cycle.*

We show the "if" part of Theorem 12; the proof of the "only if" part is deferred to the appendix.

▶ **Lemma 13.** *Let G be a graph and $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings. There exists an (α, β)-reconfiguration sequence if conditions (PR) and (DAG) hold.*

Proof. We prove this by induction on the sum of the lengths of the paths $[\alpha(v), \beta(v)]$ over $v \in V(G)$. Assume that $\alpha \neq \beta$. Every vertex $v \in V(G)$ incident to an arc of $\overrightarrow{G}_{\alpha,\beta}$ satisfies $\alpha(v) \neq \beta(v)$, because otherwise $\overrightarrow{G}_{\alpha,\beta}$ would contain a directed cycle of length 2, contradicting (DAG). It follows from (DAG) that there exists a vertex $w \in V(G)$ with $\alpha(w) \neq \beta(w)$ and whose outdegree is 0 in $\overrightarrow{G}_{\alpha,\beta}$.

Take any such $w \in V(G)$, and define $\alpha' \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ by

$$\alpha'(v) = \begin{cases} \beta(w) & \text{if } v = w; \\ \alpha(v) & \text{otherwise.} \end{cases}$$

There exists an (α, α')-reconfiguration sequence since w has outdegree 0 in $\overrightarrow{G}^1_{\alpha,\beta}$. It suffices to show that the pair (α', β) satisfies (DAG). By definition, $A(\overrightarrow{G}^2_{\alpha',\beta}) \subseteq A(\overrightarrow{G}^2_{\alpha,\beta})$ holds. We also have $A(\overrightarrow{G}^1_{\alpha',\beta}) \subseteq A(\overrightarrow{G}^1_{\alpha,\beta})$, since w has outdegree 0 in $\overrightarrow{G}^2_{\alpha,\beta}$. So $A(\overrightarrow{G}_{\alpha',\beta}) \subseteq A(\overrightarrow{G}_{\alpha,\beta})$ and (DAG) holds for (α', β). ◀

▶ Remark 14. Note that the above proof reveals the following striking property of k-Recoloring Under $\overrightarrow{R}$-Recolorability with $\overrightarrow{R}$ a multitree satisfying (S): for any graph G and colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$, if there exists some (α, β)-reconfiguration sequence, then there exists one in which the color of each vertex $v \in V(G)$ is changed in one go. ⌟

Algorithm 3 summarizes our algorithm for k-Recoloring Under $\overrightarrow{R}$-Recolorability when $\overrightarrow{R}$ is a multitree satisfying condition (S).

Algorithm 3 Algorithm for k-Recoloring Under $\overrightarrow{R}$-Recolorability with $\overrightarrow{R}$ a multitree satisfying (S).

Input : A graph G and two colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$.
Output : "Yes" if there exists an (α, β)-reconfiguration sequence, and "no" otherwise.

Step 1. Check if (PR) holds for G, α, and β. Output "no" if (PR) does not hold.
Step 2. Compute $\overrightarrow{G}_{\alpha,\beta}$.
Step 3. Output "no" if $\overrightarrow{G}_{\alpha,\beta}$ contains a directed cycle; output "yes" otherwise.

▶ **Theorem 15.** *For any multitree $\overrightarrow{R}$ satisfying condition (S), k-Recoloring Under $\overrightarrow{R}$-Recolorability is solvable in $O(|V(G)| + |E(G)|)$ time.*

Proof. As mentioned before Theorem 12, k-Recoloring Under $\overrightarrow{R}$-Recolorability can be solved by first checking whether (PR) holds for the input data (G, α, β) in $O(|V(G)|)$ time, constructing $\overrightarrow{G}_{\alpha,\beta}$ in $O(|E(G)|)$ time, and checking whether $\overrightarrow{G}_{\alpha,\beta}$ is a DAG or not in $O(|V(G)| + |E(G)|)$ time (e.g., by using the depth first search). The correctness of this procedure follows from Theorem 12. ◀

References

1 Marthe Bonamy, Matthew Johnson, Ioannis Lignos, Viresh Patel, and Daniël Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27(1):132–143, 2014. doi:10.1007/s10878-012-9490-y.

2 John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, 2008.

3 Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: Pspace-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009. doi:10.1016/j.tcs.2009.08.023.

4 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011. doi:10.1002/jgt.20514.

5 Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FUN.2018.18.

6 Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *LIPIcs*, pages 62:1–62:42. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.62.

7 Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(1):215–228, 1999. doi:10.1016/S0925-7721(99)00017-6.

8 Michael R. Garey and David S. Johnson. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

9 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98-A(6):1168–1178, 2015. doi:10.1587/transfun.E98.A.1168.

10 Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics 2013*, 409:127–160, 2013. doi:10.1017/CBO9781139506748.005.

11 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. In *Proceedings of the 19th Annual International Symposium on Algorithms and Computation (ISAAC 2008)*, volume 5369 of *Lecture Notes in Computer Science*, pages 28–39, 2008. doi:10.1007/978-3-540-92182-0_6.

12 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412:1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

13 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, Masahiro Takahashi, and Kunihiro Wasa. Independent set reconfiguration on directed graphs. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, to appear.

14 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016. doi:10.1007/s00453-015-0009-7.

15 Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, sixth edition, 2018. doi:10.1007/978-3-662-56039-6.

16 Kazuo Murota. *Discrete Convex Analysis*. Monographs on Discrete Mathematics and Applications. SIAM, 2003. doi:10.1137/1.9780898718508.

17 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

18 Hiroki Osawa. *Coloring Reconfiguration Problems and Their Generalizations*. PhD thesis, Tohoku University, 2020.

19 Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Complexity of Coloring Reconfiguration under Recolorability Constraints. In *Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ISAAC.2017.62.

20 Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Algorithms for coloring reconfiguration under recolorability constraints. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.37.

21 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

22 Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th International Symposium on Combinatorial Search (SOCS 2019)*, pages 151–159. AAAI Press, 2019.

23 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.

24 Marcin Wrochna. Homomorphism reconfiguration via homotopy. *SIAM Journal on Discrete Mathematics*, 34(1):328–350, 2020. doi:10.1137/17M1122578.

A Proofs for Section 4 (A class of multitrees)

▶ **Theorem 12** (⋆). *Let G be a graph and $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings. There exists an (α, β)-reconfiguration sequence if and only if (PR) and the following condition hold:*
(DAG) *the digraph $\overrightarrow{G}_{\alpha,\beta}$ does not contain a directed cycle.*

To prove the "only if" part of Theorem 12, we use the following lemma.

▶ **Lemma 16.** *Let G be a graph and $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ be colorings satisfying (PR) and*
($\not\subseteq$) *the digraph $\overrightarrow{G}_{\alpha,\beta}$ does not contain a directed cycle of length 2; in other words, for each $v, w \in V(G)$ with $vw \in E(G)$, we have $[\alpha(v), \beta(v)] \not\subseteq [\alpha(w), \beta(w)]$.*
Let $v, w \in V(G)$. If there exists a directed (v, w)-walk in $\overrightarrow{G}_{\alpha,\beta}$, then $\alpha(v) \leq \beta(w)$ holds.

Proof. Let $(v = u_0, u_1, \ldots, u_\ell = w)$ be a directed (v, w)-walk in $\overrightarrow{G}_{\alpha,\beta}$, and let $\overrightarrow{R}'$ be the subgraph $\bigcup_{k=0}^{\ell}[\alpha(u_k), \beta(u_k)]$ of $\overrightarrow{R}$, where we regard each directed path $[\alpha(u_k), \beta(u_k)]$ as the subgraph of $\overrightarrow{R}$ induced by the set $[\alpha(u_k), \beta(u_k)]$ of vertices. We shall see that the digraph $\overrightarrow{R}'$ has the following property:

(†) If $c \in V(\overrightarrow{R}')$ has indegree 0 in $\overrightarrow{R}'$, then c has outdegree ≤ 1 in $\overrightarrow{R}'$. If c has outdegree 0 in $\overrightarrow{R}'$, then c has indegree ≤ 1 in $\overrightarrow{R}'$.

To prove (†), we proceed as follows. We shall show below the statement of Lemma 16, *assuming* (†). Then, using this result, we can show (†) as follows. Suppose to the contrary that (†) does not hold. For each $\ell' \in \{0, 1, \ldots, \ell\}$, let $\overrightarrow{R}_{\ell'}$ be the subgraph $\bigcup_{k=0}^{\ell'}[\alpha(u_k), \beta(u_k)]$ of $\overrightarrow{R}$. Let $r \in \{0, 1, \ldots, \ell - 1\}$ be the largest number such that $\overrightarrow{R}_r$ satisfies (†). Assume without loss of generality that $\alpha(u_{r+1})$ has indegree 0 in $\overrightarrow{R}_{r+1}$, but has outdegree > 1 in $\overrightarrow{R}_{r+1}$. Take $s \in \{1, 2, \ldots, r\}$ such that $\alpha(u_s) = \alpha(u_{r+1})$. Since α is a coloring, we cannot have $(u_r, u_{r+1}) \in A(\overrightarrow{G}^1_{\alpha,\beta})$. Hence we have $(u_r, u_{r+1}) \in A(\overrightarrow{G}^2_{\alpha,\beta})$, i.e., $\beta(u_r) \in [\alpha(u_{r+1}), \beta(u_{r+1})]$. Using Lemma 16 for $\overrightarrow{R}_r$ (which satisfies (†)), we have $\alpha(u_s) \leq \beta(u_r)$ in $\overrightarrow{R}_r$. On the other hand, $[\alpha(u_{r+1}), \beta(u_{r+1})]$ is not contained in $\overrightarrow{R}_r$. Therefore, since $\alpha(u_{r+1}) \neq \beta(u_r)$, $[\alpha(u_{r+1}), \beta(u_r)]$ is not contained in $\overrightarrow{R}_r$ either. Since $\alpha(u_s) = \alpha(u_{r+1})$, it follows that there exist two directed $(\alpha(u_s), \beta(u_r))$-paths in $\overrightarrow{R}_{r+1}$ and hence in $\overrightarrow{R}$, contradicting the assumption that $\overrightarrow{R}$ is a multitree.

Now we show the statement of Lemma 16 assuming (†). Since $\overrightarrow{R}'$ trivially satisfies (S), it follows from (†) that $\overrightarrow{R}'$ satisfies the stronger condition (S') mentioned above. (Indeed, to verify (S') for $\overrightarrow{R}'$, it suffices to show that (S') holds for all *maximal* directed paths in $\overrightarrow{R}'$. A directed path $(c_0, c_1, \ldots, c_\ell)$ in $\overrightarrow{R}$ is maximal if and only if c_0 has indegree 0 and c_ℓ has

outdegree 0 in $\overrightarrow{R}'$. Hence (†) implies that c_0 has outdegree ≤ 1 and c_ℓ has indegree ≤ 1 in $\overrightarrow{R}'$. Thus, among the three possible conclusions $i \leq j$, $i = \ell$ and $j = 0$ of (S), only the first can hold.) Since $c \leq d$ in $\overrightarrow{R}'$ implies $c \leq d$ in $\overrightarrow{R}$, it suffices to show $\alpha(v) \leq \beta(w)$ in $\overrightarrow{R}'$. We shall henceforth work within $\overrightarrow{R}'$; in particular, the terms "indegree" and "outdegree", as well as the relation $\leq$, are understood to be with respect to $\overrightarrow{R}'$.

Before proving the main statement, we show the following.

▷ Claim. If each $c \in V(\overrightarrow{R}')$ with $\alpha(v) \leq c$ has indegree ≤ 1, then $\alpha(v) \leq \alpha(w)$ holds.

Proof of Claim. First observe that if each $c \in V(\overrightarrow{R}')$ with $\alpha(v) \leq c$ has indegree ≤ 1, then we have:

(∗) for each $c, d, e \in V(\overrightarrow{R}')$ with $\alpha(v) \leq c \leq e$ and $d \leq e$, either $c \leq d$ or $d \leq c$ holds.

Indeed, if $\alpha(v) \leq e$, then any $d \in V(\overrightarrow{R}')$ with $d \leq e$ satisfies either $[\alpha(v), e] \subseteq [d, e]$ or $[d, e] \subseteq [\alpha(v), e]$, i.e., $d \leq \alpha(v)$ or $\alpha(v) \leq d$. In the former case we have $d \leq c$, and in the latter case we have either $c \leq d$ or $d \leq c$ since $c, d \in [\alpha(v), e]$.

We prove $\alpha(v) \leq \alpha(u_k)$ for all $k \in \{0, 1, \ldots, \ell\}$ by induction on k. Assume that we have shown $\alpha(v) \leq \alpha(u_k)$ for some $k \in \{0, 1, \ldots, \ell - 1\}$. We have $(u_k, u_{k+1}) \in A(\overrightarrow{G}_{\alpha,\beta})$. If $(u_k, u_{k+1}) \in A(\overrightarrow{G}^1_{\alpha,\beta})$, i.e., $\alpha(u_{k+1}) \in [\alpha(u_k), \beta(u_k)]$, then $\alpha(v) \leq \alpha(u_k) \leq \alpha(u_{k+1})$. Assume $(u_k, u_{k+1}) \in A(\overrightarrow{G}^2_{\alpha,\beta})$, i.e., $\beta(u_k) \in [\alpha(u_{k+1}), \beta(u_{k+1})]$. Since $[\alpha(u_k), \beta(u_k)] \nsubseteq [\alpha(u_{k+1}), \beta(u_{k+1})]$ by ($\nsubseteq$), we have $\alpha(u_{k+1}) \notin \alpha(u_k)$. But since $\alpha(v) \leq \alpha(u_k) \leq \beta(u_k)$ and $\alpha(u_{k+1}) \leq \beta(u_k)$ hold, we have $\alpha(u_k) \leq \alpha(u_{k+1})$ by (∗). Thus $\alpha(v) \leq \alpha(u_{k+1})$ as required. ◁

We now return to the proof of Lemma 16 (under the assumption (S')). We prove $\alpha(v) \leq \beta(u_k)$ for all $k \in \{0, 1, \ldots, \ell\}$ by induction on k. Assume that we have shown $\alpha(v) \leq \beta(u_k)$ for some $k \in \{0, 1, \ldots, \ell - 1\}$. We have $(u_k, u_{k+1}) \in A(\overrightarrow{G}_{\alpha,\beta})$. If $(u_k, u_{k+1}) \in A(\overrightarrow{G}^2_{\alpha,\beta})$, i.e., $\beta(u_k) \in [\alpha(u_{k+1}), \beta(u_{k+1})]$, then $\alpha(v) \leq \beta(u_k) \leq \beta(u_{k+1})$. Assume $(u_k, u_{k+1}) \in A(\overrightarrow{G}^1_{\alpha,\beta})$, i.e., $\alpha(u_{k+1}) \in [\alpha(u_k), \beta(u_k)]$. Let $[\alpha(u_k), \beta(u_k)] = (\alpha(u_k) = c_0, c_1, \ldots, c_p = \beta(u_k))$. Define

$$i = \min\{\, q \in \{0, 1, \ldots, p\} \mid \alpha(v) \leq c_q \,\} \quad \text{and} \quad j = \max\{\, q \in \{0, 1, \ldots, p\} \mid c_q \leq \beta(u_{k+1}) \,\}.$$

Note that i is well-defined since $\alpha(v) \leq \beta(v_k) = c_p$, and j is well-defined since $c_0 = \alpha(u_k) \leq \alpha(u_{k+1}) \leq \beta(u_{k+1})$. If $i \leq j$, then we have $\alpha(v) \leq c_i \leq c_j \leq \beta(u_{k+1})$ as required.

We finish the proof by showing that $i \leq j$ holds. Suppose to the contrary that $i > j$. First observe that c_j has outdegree > 1. Indeed, if c_j has outdegree ≤ 1, then (since $j < i \leq p$) $\beta(u_{k+1}) = c_j \in [\alpha(u_k), \beta(u_k)]$ and thus we have $[\alpha(u_{k+1}), \beta(u_{k+1})] \subseteq [\alpha(u_k), \beta(u_k)]$, contradicting ($\nsubseteq$). Hence, c_i has indegree ≤ 1 by (S'). Since $0 \leq j < i$, $\alpha(v) = c_i$. It follows from (S') that the assumption of Claim is satisfied. Thus, we have $\alpha(v) \leq \alpha(u_{k+1}) \leq \beta(u_{k+1})$, contradicting $\beta(u_{k+1}) = c_j < c_i = \alpha(v)$. ◀

Proof of Theorem 12. By Lemma 13, it suffices to show the "only if" part. We assume that the colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ admit an (α, β)-reconfiguration sequence and show that conditions (PR) and (DAG) hold. Condition (PR) is clearly necessary for the existence of an (α, β)-reconfiguration sequence. To prove (DAG), assume to the contrary that there exists a directed cycle $D = (v_0, v_1, \ldots, v_\ell = v_0)$ in $\overrightarrow{G}_{\alpha,\beta}$. It suffices to show that for each coloring α' such that $(\alpha, \alpha') \in A(\mathcal{C}_{\overrightarrow{R}}(G))$ and that there exists an (α', β)-reconfiguration sequence, the directed cycle D persists in $\overrightarrow{G}_{\alpha',\beta}$; this would contradict the fact that $\overrightarrow{G}_{\beta,\beta}$ has no arcs.

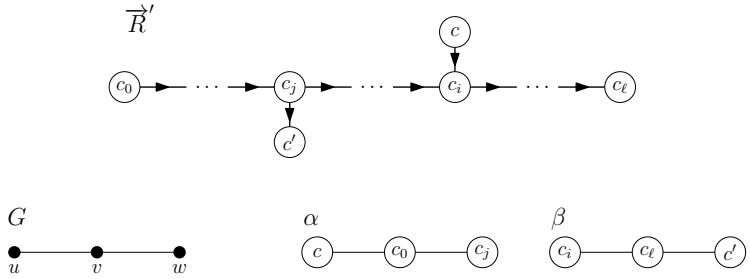


Figure 6 An example of a digraph $\overrightarrow{R}'$ that does not satisfy (S). We assume that $0 < j < i < \ell$.

We only have to treat the case where we recolor a vertex in D, say v_0. So we assume that $\alpha(v_0) \neq \beta(v_0)$ and, writing $[\alpha(v_0), \beta(v_0)] = (c_0, c_1, \dots, c_p)$, the coloring $\alpha' \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ is given by

$$\alpha'(v) = \begin{cases} c_1 & \text{if } v = v_0; \\ \alpha(v) & \text{otherwise.} \end{cases}$$

Note that the possible differences between $\overrightarrow{G}_{\alpha,\beta}$ and $\overrightarrow{G}_{\alpha',\beta}$ are only of the following form:

- some arcs of the form (v, v_0) in $\overrightarrow{G}_{\alpha,\beta}$ where $v \in V(G) \setminus \{v_0\}$ might disappear in $\overrightarrow{G}_{\alpha',\beta}$;
- some arcs of the form (v, v_0) where $v \in V(G) \setminus \{v_0\}$ which is not in $\overrightarrow{G}_{\alpha,\beta}$ might appear in $\overrightarrow{G}_{\alpha',\beta}$.

Thus we still have a path $(v_0, v_1, \dots, v_{\ell-1})$ in $\overrightarrow{G}_{\alpha',\beta}$. Since there exists an (α', β)-reconfiguration sequence, the pair (α', β) satisfies (PR) and ($\not\subseteq$). Therefore by Lemma 16 we have $\alpha'(v_0) \leq \beta(v_{\ell-1})$. If $(v_{\ell-1}, v_0) \in A(\overrightarrow{G}^1_{\alpha,\beta})$, i.e., $\alpha(v_0) \in [\alpha(v_{\ell-1}), \beta(v_{\ell-1})]$, then we have $\alpha'(v_0) \in [\alpha(v_{\ell-1}), \beta(v_{\ell-1})]$ and hence $(v_{\ell-1}, v_0) \in A(\overrightarrow{G}^1_{\alpha',\beta})$. If $(v_{\ell-1}, v_0) \in A(\overrightarrow{G}^2_{\alpha,\beta})$, i.e., $\beta(v_{\ell-1}) \in [\alpha(v_0), \beta(v_0)]$, then we have $\beta(v_{\ell-1}) \in [\alpha'(v_0), \beta(v_0)]$ and hence $(v_{\ell-1}, v_0) \in A(\overrightarrow{G}^2_{\alpha',\beta})$. So we still have the directed cycle D in $\overrightarrow{G}_{\alpha',\beta}$. ◀

▶ Remark 17. Let $\overrightarrow{R}$ be a multitree, not necessarily satisfying (S). We can still define the auxiliary digraph $\overrightarrow{G}_{\alpha,\beta}$ for each graph G and colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}}(G))$ satisfying (PR). We remark that (S) is *almost* a necessary and sufficient condition on $\overrightarrow{R}$ for Lemma 16 to hold. More precisely, $\overrightarrow{R}$ satisfies (S) if and only if all subgraphs of $\overrightarrow{R}$ satisfy Lemma 16. To show this, it suffices to exhibit a counterexample to Lemma 16 for a suitable subgraph of any multitree $\overrightarrow{R}$ not satisfying (S). If $\overrightarrow{R}$ does not satisfy (S), then it must contain a subgraph $\overrightarrow{R}'$ as in Figure 6. Now consider the graph G and colorings $\alpha, \beta \in V(\mathcal{C}_{\overrightarrow{R}'}(G))$ in Figure 6. It turns out that $(u, v) \in A(\overrightarrow{G}^2_{\alpha,\beta})$ and $(v, w) \in A(\overrightarrow{G}^1_{\alpha,\beta})$, and hence (u, v, w) is a directed path in $\overrightarrow{G}_{\alpha,\beta}$. However, we have $\alpha(u) = c \not\leq c' = \beta(w)$ in $\overrightarrow{R}'$. ⌟

Algorithms for Landmark Hub Labeling

Sabine Storandt
Universität Konstanz, Germany

Abstract

Landmark-based routing and Hub Labeling (HL) are shortest path planning techniques, both of which rely on storing shortest path distances between selected pairs of nodes in a preprocessing phase to accelerate query answering. In Landmark-based routing, stored distances to landmark nodes are used to obtain distance lower bounds that guide A* search from node s to node t. With HL, tight upper bounds for shortest path distances between any s-t-pair can be interfered from their stored node labels, making HL an efficient distance oracle. However, for shortest path retrieval, the oracle has to be called once per edge in said path. Furthermore, HL often suffers from a large space consumption as many node pair distances have to be stored in the labels to allow for correct query answering. In this paper, we propose a novel technique, called Landmark Hub Labeling (LHL), which integrates the landmark concept into HL. We prove better worst-case path retrieval times for LHL in case it is path-consistent (a new labeling property we introduce). Moreover, we design efficient (approximation) algorithms that produce path-consistent LHL with small label size and provide parametrized upper bounds, depending on the highway dimension h or the geodesic transversal number gt of the graph. Finally, we show that the space consumption of LHL is smaller than that of (hierarchical) HL, both in theory and in experiments on real-world road networks.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Approximation algorithms analysis

Keywords and phrases Hub Labeling, Landmark, Geodesic, Hitting Set, Highway Dimension

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.5

1 Introduction

There exists a plethora of shortest path planning algorithms, many of which follow the preprocessing paradigm to achieve better running times than Dijkstra's algorithm [8]. Combining orthogonal techniques often leads to further acceleration [11, 26]. However, such combinations usually escape rigorous formal analysis and are only justified empirically. In this paper, we propose and analyze the concept of Landmark Hub Labeling as a symbiosis of two preprocessing-based techniques, Hub Labeling and Landmark-based routing.

Hub Labeling (HL) is an elegant technique for shortest path computation in weighted graphs $G(V, E, c)$ which has been extensively studied both theoretically and empirically [14, 3, 4, 17, 5, 16, 7]. The core idea of HL is to construct labels $L : V \to 2^V$ for all nodes $v \in V$ which have to fulfill the so called *cover property*. The property demands that for any node pair $s, t \in V$, the intersection of their labels $L(s) \cap L(t)$ contains a node w on the shortest path from s to t. Now, if shortest path distances are stored along with the nodes in the label, the shortest path distance $c(s, t)$ from s to t can simply be computed as the sum of the distances from s to w and from w to t. As for all nodes $v \in L(s) \cap L(t)$ we have $c(s, v) + c(v, t) \geq c(s, t)$ and the inequality is tight for at least one node, the correct distance value can be computed purely based on label information. HL exhibits excellent distance query times on a large variety of graphs. For example, on road networks, running times in the microsecond range are achieved (a speed-up of more than six orders of magnitude over the Dijkstra baseline) [4], outperforming its state-of-the-art competitors. The price to pay is the rather high space consumption to store the node labels. On the road network of Western Europe, average label sizes of 69 were obtained (with an elaborate HL construction algorithm), increasing the space consumption from 0.4GB for the original graph to 18GB.

Landmark-based routing requires less space to store preprocessed data and is more flexible in that regard. Here, a small set of nodes is selected to be landmarks and distances from all nodes in the graph to the landmarks are precomputed. Shortest s-t-path queries are answered with ALT (A* + landmarks + triangle inequality) [23] which utilizes the precomputed distances to the landmark nodes to get lower bounds for the distance of a node to the target t. These lower bounds serve as an admissible heuristic for A* search and help to significantly reduce the search space size compared to running Dijkstra's algorithm. For example, on the road network of Western Europe, using 16 landmarks results in a reduction of the query time from about 5 seconds to 50 milliseconds, with 32 landmarks to 30 milliseconds [20].

There is seemingly a huge gap between the query times of ALT and HL but one has to note that ALT always returns the full shortest path while HL is a distance oracle. Shortest paths can also be computed based on HL but the respective algorithm needs to issue one HL distance query per edge in the shortest path. Thus, if the path contains thousands of edges, the gap is less pronounced.

We will show that our new Landmark Hub Labeling scheme exhibits interesting theoretical properties and allows for novel trade-offs between space consumption and query time.

1.1 Related Work

Hub Labeling (HL) was originally proposed by Cohen et al. [14] under the name 2-*hop labeling*. It was proven that minimizing the total label size is NP-hard in [7] and a $\mathcal{O}(\log n)$ approximation algorithm with a running time of $\mathcal{O}(n^5)$ was designed. The running time was later reduced to $\mathcal{O}(n^3 \log n)$ [16]. Furthermore, parameterized upper bounds for label sizes in HL were investigated. In [10], it was shown that search space sizes in a related preprocessing-based shortest path technique, called Contraction Hierarchies, can be upper bounded by $\mathcal{O}(tw \log n)$ where tw denotes the treewidth of the graph. The bounds transfers to label sizes in a HL [4]. In [2], it was proven that labels with sizes in $\mathcal{O}(h \log D)$ exist, where h denotes the highway dimension of the graph. Label sizes in $\mathcal{O}(h \log D \log n)$ can be computed in polytime. Furthermore, the skeleton dimension κ was introduced to analyze HL in [29]. Via a randomized construction algorithm, label sizes in $\mathcal{O}(\kappa \log n)$ were shown to be possible. For practical application, however, usually fast heuristics are used instead of algorithms with provable guarantees [15, 16, 32, 13]. These heuristics typically produce a special type of labeling, called hierarchical Hub Labeling (HHL). While hierarchical labelings might be larger by a factor of $\Omega(\sqrt{n})$ than general HL [7], the respective algorithms were shown to efficiently produce concise labels on real-world networks.

The ALT algorithm was proposed in [23] to improve the performance of A* via landmark-based lower bounds. Subsequent work has mainly focused on how many landmarks are needed and how they should be selected to get strong lower bounds while exhibiting low space consumption [19, 24, 21, 33]. It was proven in [9] that selecting a landmark node set of fixed size is NP-hard when the goal is to minimize the expected number of nodes settled with ALT. In [25], the concept of *active landmarks* was introduced and explored, where during query answering not the distances to all available landmarks are evaluated but only to a promising subset. This reduces the time for lower bound computation and hence accelerates query answering. Landmark-based routing was also studied for dynamic graphs with changing edge weights, either as a standalone method [20, 21, 27] or in combination with other (hierarchical) techniques [18, 30].

We note that also synergies between HL and landmarks as well as innovative labeling schemes were explored before. The Pruned Landmark Labeling (PLL) algorithm [5, 6] first subdivides the input network into shortest paths and then infers node labels that encode

distances towards these paths. The PLL approach utilizes landmarks to compute a concise labeling efficiently. Note that despite the similarity in name this is a vastly different approach to the concept we propose in this paper, as PLL uses landmark information only in the preprocessing step and the correctness of query answering relies on tight upper distance bounds just as for classical HL.

In [34], it was observed that stored distances to landmark nodes can not only be used to deduce lower bounds but also upper bounds in a similar fashion as in HL. However, in contrast to labeling approaches, their goal is not necessarily to be able to answer queries exactly purely based on label information. Instead, they either return only a distance estimate (e.g. the geometric mean of upper and lower bound) or fall back on A* if the gap between the bounds is too large. But they also prove that the Landmark Cover problem, in which one has to select the smallest number of landmarks to have tight bounds for all node pairs, is NP-hard. Interestingly, they define the tightness only via the upper bound and hence they rather prove that a variant of HL is NP-hard in which the labels of all nodes need to be the same. They also show that this problem can be approximated within a factor of $\mathcal{O}(\log n)$ in $\mathcal{O}(n^3)$. In our definition of Landmark Hub Labeling, we allow nodes to have individual label sets and the tightness criterion to be fulfilled either via an upper or a lower bound – again individually for each node pair – while still guaranteeing exact query answering.

1.2 Contribution

In this paper, we extend the concept of labeling beyond Hub Labeling (HL), and we design novel query algorithms as well construction schemes with provable guarantees regarding the label sizes. The following are our main contributions:

- We propose two novel labeling schemes, Landmark Labeling (LL) and Landmark Hub Labeling (LHL), which integrate distance lower bounds into the labeling framework. We show that there exist graphs on which the label sizes of LL and LHL are smaller than the label sizes of classical HL by a logarithmic factor.
- We also introduce a new property for general labelings called path-consistent (PC). We show that the PC property allows to answer shortest path queries significantly faster.
- We prove that the maximum label size in a PC-LHL is upper bounded by gt, where gt denotes the geodesic transversal number, a recently introduced graph parameter [31]. We discuss how a PC-LHL with label sizes in $\mathcal{O}(gt \log n)$ can be constructed in $\mathcal{O}(n^3)$.
- We investigate the relationship between gt and the highway dimension h, a graph parameter which was used in previous work to bound label sizes of classical HL [2]. While it turns out that gt and h are incomparable, we show that nevertheless the label sizes of PC-LHL can be also upper bounded in dependency of h.
- We design an approximation algorithm for PC-LHL (as well as PC-LL and PC-HL) that runs in $\mathcal{O}(n^6)$ and guarantees a total label size within a factor of $\mathcal{O}(\log n)$ of the optimum. The algorithm builds on the greedy set cover framework for HL computation introduced in [7] but needs to be significantly modified to be able to deal with the PC property.
- Inspired by hierarchical HL (HHL), we also study properties of hierarchical PC-LHL (HPC-LHL). In a proof-of-concept experimental study on medium sized road networks, we demonstrate that label sizes in HPC-LHL are indeed smaller than those in HHL.

2 Preliminaries

Throughout the paper, we assume to be given a weighted graph $G(V, E, c)$ with positive edge weights $c : E \to \mathbb{R}^+$ and a unique shortest path between any node pair $s, t \in V$. The latter is a common assumption (in particular when dealing with road networks) but it can also

be enforced in arbitrary graphs via (symbolic) edge weight perturbation. Furthermore, we assume the input graph to be undirected. Both, HL and ALT can also be applied to directed graphs by storing label information or landmark distances for each direction separately. The same is true for LHL. But for ease of exposition, we restrict our decriptions to the undirected case. We use $c(s,t) = c(t,s)$ to denote the shortest path distance between nodes $s,t \in V$.

3 Landmarks and Labelings

In this section, we first recall existing notions of landmarks and labelings and then introduce the new properties and schemes that will be studied in the remainder of the paper.

3.1 Notions of Landmarks

Landmark-based routing relies on selecting a (small) set of nodes as landmarks and precomputing the shortest path distances from the landmarks to all other nodes. For any node pair $s,t \in V$ and any landmark node l, it follows from the triangle inequality that $|c(s,l) - c(t,l)| \leq c(s,t)$. Hence we get a valid lower bound for the shortest path distance between s and t. Choosing the maximum lower bound over all landmark nodes is an admissible heuristic for A*. The tightness of the lower bounds depends on the number and distribution of the selected landmarks. Landmarks with a roughly equal distance to s and t (and hence in particular landmarks in the middle of the shortest path from s to t) provide little to no information. In contrast, some landmarks even allow to obtain tight lower bounds. Such landmarks are called *perfect*.

▶ **Definition 1** (Perfect Landmark). *A landmark $l \in V$ is called* perfect *for a node pair $s,t \in V$ if $|c(s,l) - c(t,l)| = c(s,t)$.*

To characterize perfect landmarks, we next introduce the notion of a *path landmark*.

▶ **Definition 2** (Path Landmark). *A landmark $l \in V$ is a* path landmark *for a node pair $s,t \in V$, if t lies on a shortest path from s to l or s lies on a shortest path from t to l.*

The relation between path landmarks and perfect landmarks is described in the following two simple lemmas.

▶ **Lemma 3.** *Path landmarks are perfect.*

Proof. If w.l.o.g. t lies on a shortest path from s to l, then we have $c(s,t) + c(t,l) = c(s,l)$. Rearranging the formula, we get $c(s,l) - c(t,l) = c(s,t)$. ◀

▶ **Lemma 4.** *All perfect landmarks are path landmarks.*

Proof. Let $l \in V$ be a perfect landmark for s,t. Then we know that w.l.o.g. $c(s,l) - c(t,l) = c(s,t)$ and hence $c(s,t) + c(t,l) = c(s,l)$. The latter implies that t lies on a shortest path from s to l and hence l is a path landmark for s,t. ◀

It follows that if shortest paths are unique, then for an s-t-perfect landmark l the shortest path from s to t is always a prefix of the shortest path from s to l.

The lemmas also provide some justification why landmark selection strategies as e.g. the *farthest strategy* [23] which prefer nodes in the periphery of the graph usually work very well.

3.2 Notions of Labelings

Next, we delve into labeling schemes. Based on the definition of classical Hub Labeling, we propose two new labeling schemes, namely Landmark Labeling and Landmark Hub Labeling, which build on the notion of path landmarks discussed above.

▶ **Definition 5** (Hub Labeling). *A* Hub Labeling *(HL) is a labeling $L : V \to 2^V$ which fulfills the* (hub) cover property*, i.e., for any two nodes $s, t \in V$ there is a node $w \in L(s) \cap L(t)$ which is a hub, i.e. w is on the shortest path from s to t.*

A HL is called a *hierarchical* Hub Labeling (HHL), if for some node ranking $r : V \to \mathbb{N}$ the nodes in $L(v)$ all have rank at least $r(v)$. It was proven in [7] that for fixed r the HHL with minimum total label size is the so called *canonical* HHL in which $w \in L(v)$ if and only if there is a shortest path emerging from v on which w (and only w) has maximum rank.

The idea behind (H)HL is that for each node $w \in L(s) \cap L(t)$ an upper bound for the shortest path distance from s to t can be derived (if distances to the nodes in the labels are stored along), and that this upper bound is tight for an s-t-hub. But we can also design a labeling scheme, where for each node $w \in L(s) \cap L(t)$ a lower bound for the shortest path distance from s to t can be inferred and then demand that the largest lower bound has to be tight. This leads us to the definition of a Landmark Labeling.

▶ **Definition 6** (Landmark Labeling). *A* Landmark Labeling *(LL) is a labeling $L : V \to 2^V$ which fulfills the* landmark cover property*, i.e., for any two nodes $s, t \in V$ there is a node $w \in L(s) \cap L(t)$ which is a path landmark for s, t.*

Consider any shortest path π that contains the nodes s and t. Then in a HL, only nodes on the shortest subpath from s to t contribute to fulfilling the *hub cover property* for this node pair. In contrast, in a LL, only nodes on π except for the nodes between s and t contribute to the *landmark cover property* for s, t. Accordingly, for any node w on a shortest path through s and t, the correct shortest s-t-path distance could be interferred if w is contained in both of their labels, but the two labeling schemes considered so far restrict the selection to specific subsets. We now lift this restriction by allowing any node w on any shortest path π which contains s and t to cover the pair s, t.

▶ **Definition 7** (Landmark Hub Labeling). *A* Landmark Hub labeling *(LHL) is a labeling $L : V \to 2^V$ which fulfills the* landmark hub cover property*, i.e., for any two nodes $s, t \in V$ there is a node $w \in L(s) \cap L(t)$ which either is a hub or a path landmark for s, t.*

An important property to be considered in our study of LHL will be *path-consistency*.

▶ **Definition 8** (Path-Consistent Labeling). *A* path-consistent *(PC) labeling is a labeling, where for each $w \in L(v)$, we also have $w \in L(u)$ for all u on the shortest path from v to w.*

To illustrate that path-consistency is a rather natural labeling property, we prove that a canonical HHL is always PC.

▶ **Lemma 9.** *Canonical hierarchical Hub Labelings are path-consistent.*

Proof. Let $L(v)$ be the label of v in a canonical HHL that respects ranking r. Let $w \in L(v)$ be a node contained in the label. Clearly, for any node u on the shortest path from v to w, we have $r(u) < r(w)$. Otherwise w would not be included in $L(v)$. Hence w also is the node of highest rank in any suffix of the shortest path from v to w. Accordingly, for all nodes u on the shortest path from v to w, it holds that $w \in L(u)$. ◀

Table 1 Optimal maximum (and average) label sizes for Hub Labeling (HL), Landmark Labeling (LL) and path-consistent Landmark Hub Labeling (PC-LHL) on example instances.

	star graph	path graph
HL	$\Theta(1)$	$\Theta(\log n)$
LL	$\Theta(n)$	$\Theta(1)$
PC-LHL	$\Theta(1)$	$\Theta(1)$

We will argue that path-consistency is beneficial for all three types of labelings (HL, LL, LHL) when it comes to shortest path retrieval. However, for LHL we will show that the property is crucial for efficient query answering. So from now on, we focus on path-consistent Landmark Hub Labelings (PC-LHL). Note that by Lemma 9, the optimal label sizes in a PC-LHL are upper bounded by the optimal label sizes in any HHL, as we have shown that any canonical HHL is a PC-HL and any PC-HL is a PC-LHL by definition.

To further see why PC-LHL is interesting, Table 1 shows instances on which the label sizes in a PC-LHL are smaller than those in an HL or LL. This is a result of the greater freedom to choose cover nodes in LHL. For LL, the endpoints of any maximal shortest path have to contain at least one of them in both of their labels to fulfill the *landmark cover property*. Hence, LL requires large maximum label sizes on star graphs (and in general on connected graphs with many dead-ends). For HL and PC-LHL, it suffices to include the middle node of the star in all labels. On path graphs, however, the *hub cover property* of HL results in label sizes that are logarithmic in the number of nodes on the path [9]. For LL and LHL, choosing one endpoint as label for all nodes is sufficient (and also trivially PC).

4 Query Algorithms

We distinguish two different types of queries: (i) distance queries and (ii) path queries. We first describe how these types of queries are answered with classical HL and argue that similar algorithms work for LL. For PC-LHL, however, we describe a novel query answering algorithm that takes care of the possible mix of hubs and path landmarks in the labels.

4.1 Distance and Path Queries with HL and LL

For HL, answering a distance query for nodes s and t is very simple and can be summarized with the following formula: $c(s,t) = \min_{w \in L(s) \cap L(t)} c(s,w) + c(t,w)$. If the labels are presorted by node ID, the intersection of two labels $L(s)$ and $L(t)$ can be computed in time linear in $\max\{|L(s)|, |L(t)|\}$ and the query time is hence in $\mathcal{O}(L_{max})$ with $L_{\max} := \max_{v \in V} |L(v)|$. Similarly, for LL, we have $c(s,t) = \max_{w \in L(s) \cap L(t)} |c(s,w) - c(t,w)|$ and can hence answer distance queries in $\mathcal{O}(L_{\max})$ as well.

For path queries, where not only the length but the sequence of edges on a shortest path should be returned, query answering becomes more involved. To make shortest path retrieval more efficient, not only the distances to the hub nodes are stored with the labels but also the first edge on the shortest path. Then, in a path query, we first issue a distance query to identify the hub node w and the shortest path distance $c(s,t)$. The shortest path from s to t is then a concatenation of the shortest path from s to w and the shortest path from w to t. The two subpaths can be extracted independently.

To extract any shortest a-b-path, first a distance query is issued and the respective hub w_{ab} is indentified. If $w_{ab} \neq a$, then the first edge on the shortest path from a to b is the one associated with the label $w \in L(a)$. Otherwise, we have $a \in L(b)$ and the edge associated

with this label is the last edge on the shortest path from a to b. Hence we either retrieve the first edge (a, a') or the last edge (b', b) of the shortest path and can then recursively repeat the procedure.

Overall, shortest path retrieval demands to issue k distance queries, where k denotes the number of edges on the shortest path. The running time for a path query is therefore in $\mathcal{O}(D \cdot L_{\max})$ where D denotes the graph diameter. A similar procedure works for LL, but we only need to return the subpath from s to t instead of the whole path to the landmark node.

4.2 Path Queries with Path-Consistent Labelings

In case we deal with a path-consistent (PC) labeling, path queries can be answered faster as follows. To each node $l \in L(s)$, we assign a pointer to the entry of $l \in L(s')$ where s' is the node after s on the shortest path from s to l. Note that $l \in L(s')$ follows directly from the PC property and that the space consumption of the labeling only increased linearly by the addition of these pointers. The shortest path from s to $l \in L(s)$ can then simply be retrieved by following the pointers towards l iteratively. Thus, path retrieval takes only $\mathcal{O}(k)$ where k denotes the number of shortest path edges. As a path query is composed of a distance query and subsequent path retrieval, we get the following corollary.

▶ **Corollary 10.** *Path query times for PC labelings are in* $\mathcal{O}(L_{\max} + D)$.

4.3 Query Answering with PC-LHL

For PC-LHL, we know that $L(s) \cap L(t)$ needs to contain either a hub or a path landmark for s, t but we don't know which of the two it is. And the role of a node in $L(s)$ might be different for different targets t. We hence apply both, the HL and the LL distance query answering routine to the label sets. This provides us with an upper bound $UB := \min_{w \in L(s) \cap L(t)} c(s, w) + c(t, w)$ and a lower bound $LB := \max_{w \in L(s) \cap L(t)} |c(s, w) - c(t, w)|$ on the shortest path distance, with at least one of them being tight.

If $UB > LB$, we have to figure out whether a shortest path of length LB exists. Let w_l be the label in $L(s) \cap L(t)$ from which LB resulted. If w_l is a path landmark for s, t, then w.l.o.g. t has to lie on the shortest path from s to w_l at distance LB from s. We hence need to extract the prefix of the shortest path from s to w_l up to a distance of LB and check whether t is found. For prefix extraction, we can use the path query algorithm for PC labelings, see Corollary 10. If t is found, the extracted prefix is the shortest path from s to t. Otherwise, UB is tight and the shortest path from s to t can be identified by extracting the paths from s and t to the respective hub node w_h (again using PC queries). The total running time of the query algorithm for PC-LHL is hence in $\mathcal{O}(L_{\max} + D)$ for both, distance and path queries.

Accordingly, PC-LHL has a worse distance query time than HL and LL, as path retrieval is necessary to decide whether UB or LB is tight. For path queries, however, the worst-case running time is better due to the enforced PC property. If we consider PC-HL and PC-LL then worst case path retrieval times match, but PC-LHL allows for smaller label sizes.

5 Parametrized Bounds for Landmark Hub Label Sizes

Next, we investigate how to construct PC-LHL with bounded label size. For that purpose, we exploit two parameters that measure the complexity of the shortest path structure of the input graph G, namely the *geodesic transversal number* [31] and the *highway dimension* [2].

5.1 Obtaining Labels from Geodesic Transversal Sets

The geodesic transversal number, abbreviated by $gt(G)$, denotes the size of a smallest hitting set for all maximal shortest paths in G. A shortest path (also called geodesic) is maximal if there is no superpath that is also a shortest path. A hitting set S for the set of maximal geodesics Π is also called a *gt*-set. Given a *gt*-set, we can easily construct a valid PC-LHL as shown in the following theorem.

▶ **Theorem 11.** *Given a graph $G(V, E)$, the maximum label size $L_{\max}$ of a PC-LHL can be bounded by the geodesic transversal number $gt(G)$.*

Proof. Let S be a *gt*-set in G. We claim that setting $L(v) = S$ for all $v \in V$ consitutes a valid PC-LHL. To see that, consider any pair of nodes s, t and the respective shortest path π. Then there is at least one maximal geodesic π' in G which contains π as a subpath. If $\pi \cap S \neq \emptyset$, then $L(s) \cap L(t) = S$ contains a valid hub for π. Otherwise, S has to contain at least one node from $\pi' \setminus \pi$ which constitutes a path landmark for s, t. Hence the landmark cover property is fulfilled. Path-consistency is obviously obeyed in case all labels are identical. ◀

Note, that the bound does not apply to HL or LL. This follows from the examples in Table 1. For path and star graphs, we have $gt = 1$ but HL and LL label sizes depend on n, respectively.

On trees, optimal *gt*-sets can be computed in linear time, but on general graphs deciding whether a graph has a *gt*-set of size $k \in \mathbb{N}$ is NP-hard [31].

However, one can find a *gt*-set of size $\mathcal{O}(gt \log n)$ in $\mathcal{O}(n^3)$ by extracting the set Π of all maximal geodesics from the input graph and computing a greedy hitting set thereupon. More precisely, after enumerating the $\mathcal{O}(n^2)$ maximal geodesics, we create a boolean array A which encodes whether a geodesic was already hit. Initially, the array is filled with *false* entries. Then we sweep over the geodesics and assign their IDs to each node contained therein. Additionally, we store the size of the resulting set at each node in form of a counter. This takes $\mathcal{O}(n^3)$ overall. Subsequently, in each of the $\mathcal{O}(n)$ rounds of the algorithm, the node with the largest counter value is identified in $\mathcal{O}(n)$ and added to the initially empty hitting set S. The counter of the node is set to zero. For all geodesics assigned to the selected node, we check in A whether they are newly hit. This takes constant time per geodesic and hence $\mathcal{O}(n^2)$ for all geodesics assigned to the node. For each newly hit geodetic, we set its entry in A from *false* to *true*, and then sweep over all nodes contained in that geodesic in order to decrement their counters. Note that we do not need to find or remove the respective ID of the geodesic from the stored sets at the nodes. Hence the update time of the data structures is in $\mathcal{O}(n)$ for each newly hit geodesic and in $\mathcal{O}(n^3)$ for all of them. The algorithm terminates if only nodes with a counter of zero are left. The total running time of the algorithm is hence in $\mathcal{O}(n^3)$.

We further remark that incorporating the fact that shortest paths have low VC-dimension, one can even compute *gt*-sets of size $\mathcal{O}(gt \log gt)$ in polytime [1].

5.2 Relation to Highway Dimension

The highway dimension h is a graph parameter that was specifically designed to study preprocessing-based shortest path planning techniques in road networks. For brevity, we don't include the rather involved definition here. For our considerations, it will only be important that the highway dimension is at least as large as the maximum node degree in the graph [2].

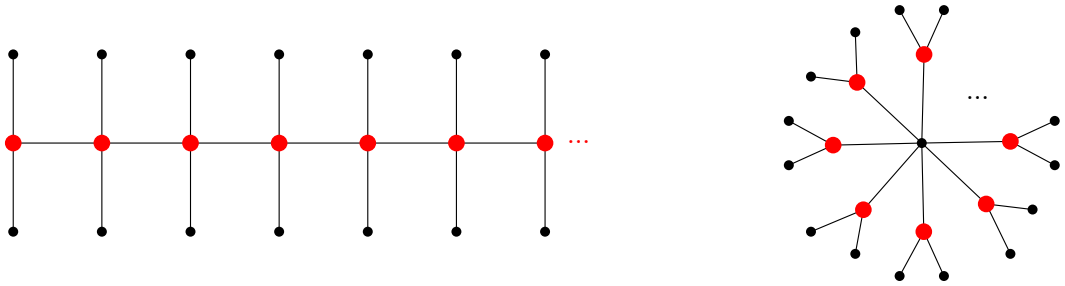


Figure 1 Example graphs with their optimal gt-sets in red. Left: special caterpillar. Right: augmented star graph.

For classical HL, labels of size $\mathcal{O}(h \log D \log n)$ can be computed in polytime, where D is the graph diameter. The construction also relies on hitting sets, more precisely on so called multi-level sparse shortest path hitting sets (SPHS) [2]. The number of levels is $\Theta(\log D)$. The top level is a hitting set for all shortest paths of length at least $\frac{D}{2}$ which can be greedily constructed in $\mathcal{O}(n^3)$. The lowest level simply includes all nodes. The other SPHS levels each require to solve up to n local shortest path hitting set instances (each considering paths of length 2^{i-1} at level i for $i = 1, \ldots, \lceil \log D \rceil - 1$). Hence (without further improvements) the running time per level is in $\mathcal{O}(n^4)$, resulting in a total running time of $\mathcal{O}(n^4 \log D)$. The node labels can then be interferred from the multi-level SPHS in $\mathcal{O}(n^3)$.

Again, exploiting the fact that shortest path sets have small VC-dimension, the bound can be improved to $\mathcal{O}(h \log D \log h)$ [1]. The question is whether this bound is tighter than the gt-bound. To answer this question, we study the relationship of gt and h.

▶ **Lemma 12.** *The geodesic transversal number $gt(G)$ and the highway dimension $h(G)$ are incomparable.*

Proof. Consider a star graph. The geodesic transversal number is 1, as the center node is a valid hitting set for all maximal geodesics. As the center node has a degree of $n-1$ and the highway dimension is lower bounded by the maximum node degree, we have $h \in \Omega(n)$.

For an example graph in which $h \ll gt$, we consider a specific tree graph, called a caterpillar. In a caterpillar, there exists a backbone path that contains all nodes of degree larger than two. We assume now that the backbone contains $n/3$ nodes, each of which is adjacent to two leaf nodes. Then, for each backbone node, the path between its two attached leaves is a maximal geodesic; each of which requires its own hitter in a gt-set. See Figure 1 (left) for an illustration. Therefore, we have $gt \in \Theta(n)$. In [12], it was shown that in caterpillars with constant node degree the highway dimension is a constant. ◀

Interestingly, we will show that the h-dependent upper bound nevertheless also holds for PC-LHL by proving that the SPHS-based labeling algorithm produces a path-consistent HL.

▶ **Lemma 13.** *The HL returned by the multi-level SPHS algorithm is path-consistent.*

Proof. Node labels are computed in the algorithm as follows: Based on the multi-level SPHS each node receives a value between 0 and $\lceil \log D \rceil$ indicating the highest level in which it appears as a hitter. For a node $v \in V$, the shortest path tree emerging from v is computed. Then along each shortest path starting at v, nodes are included in the label of v that have a higher level than all the ones previously seen on that shortest path. Hence the resulting labeling is a canonical HHL with respect to the level function. As proven in Lemma 9, such a labeling is always PC. ◀

The implications of the lemma are twofold: Firstly, the labels produced by the SPHS-based algorithm can benefit from the improved path queries times described in Section 4.2. Secondly, as every PC-HL is also a valid PC-LHL by definition, we get the following corollary.

▶ **Corollary 14.** *The label size in a PC-LHL can be bounded by* $\mathcal{O}(h \log D)$ *and labels of size* $\mathcal{O}(h \log D \log h)$ *can be computed in polytime.*

So now we have two valid upper bounds for the maximum label size in a PC-LHL, one depending on gt and one depending on h. The maximum label size $L_{\max}$ can be significantly smaller than those graph parameters, though, as shown in the next lemma.

▶ **Lemma 15.** *There exist graphs* $G(V, E)$ *in which the maximum label size of a PC-LHL is in* $\mathcal{O}(1)$ *but* $gt \in \Theta(n)$ *and* $h \in \Theta(n)$.

Proof. Consider an augmented star graph with $n/3$ arms where at the end of each arm is adjacent to the center node of the star but also to two other nodes of degree 1 each (as shown in Figure 1, right). Then there are $n/3$ maximal shortest paths that consist of two edges each, namely the ones that connect the degree-1 nodes via the intermediate star arm nodes. As none of those $n/3$ paths overlap, we have $gt \in \Theta(n)$. From the center node of the star having a degree of $\Theta(n)$, it follows that $h \in \Theta(n)$ holds as well. For a PC-LHL, each degree-1 node needs to contain its one adjacent arm node in its label and the center node of the star. All other node labels only need to contain the center node of the star. The resulting labeling is path-consistent. Thus, we get $L_{\max} \in \mathcal{O}(1)$. ◀

It follows from the lemma that the obtained parameterized upper bounds do not provide us with an approximation guarantee for the (maximum or total) label size of a PC-LHL. Therefore, we will next discuss how to design a proper approximation algorithm.

6 An Approximation Algorithm for Path-Consistent Labelings

For classical HL, an $\mathcal{O}(\log n)$-aproximation with a running time of $\mathcal{O}(n^5)$ for the total (and hence also average) label size was proposed in [7], which is based on the greedy set cover algorithm. As the respective set cover instance would have exponential size, it cannot be constructed explicitly, though. But it was shown that the greedy selection of a sufficiently good set can be accomplished in polytime based on a reduction to the Densest Subgraph Problem. We will recap the algorithm details below and show that the algorithm can easily be modified to also approximate the size of a LL or LHL. However, incorporating the PC property poses a new challenge.

6.1 Hub Labels and Densest Subgraphs

To obey the *hub cover property*, there needs to be node for each shortest path that is contained in the labels of both of its endpoints. To construct a cooresponding set cover instance $(U, \mathcal{S})$, the universe U is the set of all (unordered) pairs of nodes (representing the set of all shortest paths). The possible sets $\mathcal{S}$ are induced by a hub node w and a node subset W. Then $S(w, W)$ covers all pairs of nodes in W for which w is a valid hub, and the cost of S is equal to the size of W, that is $c(S) = |W|$. An optimal set cover solution for $(U, \mathcal{S})$ constitutes a valid hub labeling of minimum size (by including w in $L(v \in W)$ for all selected sets). A greedy set cover solution guarantees an approximation factor of $\mathcal{O}(\log |U|)$. However, the problem is, that there are exponentially many subsets W to consider for each potential hub node $w \in V$. Hence the set cover instance cannot be constructed efficiently.

But for the greedy algorithm to work, one only needs fast access to the set $S \in \mathcal{S}$ with the largest ratio $p(S)/c(S)$ where $p(S)$ denotes the number of new node pairs covered by S and $c(S)$ the cost of the set. The idea put forward in [7] is to compute in each round the currently best subset W for each potential hub node w by solving an instance of the Densest Subgraph Problem (DSG) and then simply select the best among those $\mathcal{O}(n)$ sets. Given a graph, the DSG demands to find the node subset D for which the subgraph induced by D has maximum density, i.e. largest ratio of number of edges and number of nodes. The problem can be solved to optimality in polytime and there is also linear time 2-approximation [28]. To use DSG for HL set cover, a so called *center graph* $G_w(V, E_w)$ is constructed in each round for each potential hub w. The center graph contains the same node set as the input graph and there is an edge between nodes s, t if and only if w is on a shortest path from s to t, i.e. if w is a valid hub for the node pair s, t. The DSG of the center graph then equals the currently best node subset $W = D$ to include node w in their labels. Note that in the course of the algorithm it might happen that the same node w is selected multiple times with different node subsets, and that there might be node pairs newly covered by w where one node was selected in a previous round. To make sure that such node pairs are included in $p(S)$ without double counting the addition of w to the label, nodes that already have w in their label receive a cost of 0 in the center graph and all others a cost of 1. Note that this results in a change of $c(S)$, the cost of set S. While the greedy set cover algorithm does not necessarily uphold its approximation guarantee if the costs of the sets change in the process, it was proven in [7] that for each potential hub w, the density of the best subgraph in each round may only decrease and hence the approximation factor is preserved.

For LL and LHL, the very same approximation framework can be used. The only difference is that edges in the center graphs are inserted in accordance with the respective cover property.

6.2 Ensuring Path-Consistency

The greedy set cover algorithm does not automatically produce PC labelings, though. Hence we need to change the model and also the greedy selection method appropriately to compute a valid PC-LHL (or PC-HL or PC-LL).

First of all, $\mathcal{S}$ can be restricted to sets $S(w, W)$ where W is a connected subgraph of the shortest path tree emerging from w. Otherwise the PC property would be violated. We refer to such sets W from now on as *trimmed shortest path trees* (rooted in w). The number of trimmed shortest path trees might still be exponential, though, and solving the DSG problem for the center graph of w might return a node set D which does not coincide with a valid set $S(w, W)$, i.e. where $W = D$ is not a trimmed shortest path tree.

To be able to use the greedy set cover framework nevertheless and to preserve its approximation guarantee, we need an efficient algorithm for selecting a valid set $S(w, W)$ with close-to-optimal ratio of newly covered pairs and cost. We hence investigate the problem of computing the densest subgraph of the center graph where the node set corresponds to a trimmed shortest path tree rooted in w. Thus, in each round, we construct the center graph $G_w(V, E_w)$ for w as before. We also compute the full shortest path tree T rooted in w. The density $d(S)$ of a subgraph S of T is defined as $m(S)$, the number of edges incident to the respective nodes in G_w, divided by $n(S)$, the number of nodes in the subgraph with weight 1 (again, nodes will receive a cost of zero if they already contain node w in their label). For technical purposes, we assume that $w \in L(w)$ for all nodes $w \in V$. This obviously does not affect the PC-property. As the label size of all nodes has to be at least 1 for correct query answering anyway, this will also not affect the approximation factor we are going to show.

We then propose the following greedy algorithm to find a trimmed shortest path tree with close-to-maximum density: Let T be the current trimmed shortest path tree with edges directed away from the root. Compute the density of all proper subtrees of T. Here a subtree of T rooted at node $v \in T$ contains all nodes reachable from v in T via the directed edges. Remove the subtree with smallest density from T and also remove the contained nodes and their incident edges from G_w. Repeat until T consists of only the root or until we have $n(S) = 0$ for all subtrees. Keep track of the temporary trimmed tree T with maximum density over the course of the algorithm and return it in the end.

▶ **Theorem 16.** *The greedy algorithm returns a trimmed shortest path tree with density $\lambda/2$ where λ denotes the maximum density among all trimmed shortest path trees rooted in w.*

Proof. The proof follows the argumentation for the 2-approximation of DSG [28] but is more intricate as it has to deal with the tree structure.

Let T_{OPT} be the optimal trimmed tree, i.e. the one with maximum density λ. Observe that every directed subtree of T_{OPT} has to have a density of at least λ. Otherwise, the removal of that subtree would increase the density.

Now consider the first iteration in the greedy algorithm in which a subtree S^* is removed which intersects T_{OPT}. Before the removal of S^*, all nodes of T_{OPT} are still contained in the current T and G_w. Hence the intersection tree T^* of S^* and T_{OPT} needs to have a density of at least λ by the observation made above. It follows that $d(S^*) \geq \lambda$, as either S^* is a subtree of T_{OPT}, or, in case S^* contains nodes outside of T_{OPT}, its subtree $S \setminus T_{OPT}$ needs to have higher density than S (or equal density) to justify the greedy choice. As we know that there are at least $\lambda \cdot n(T^*)$ edges incident to nodes from T^* which connect them to other nodes from T_{OPT} in G_w, adding T^* to $S \setminus T_{OPT}$ cannot decrease the density to a value below λ.

If S^* – as the subtree of currently lowest density – has $d(S^*) \geq \lambda$, then all proper subtrees of the current T need to have a density $\geq \lambda$ as well. We now show that this implies that $d(T) \geq \lambda/4$. Each node in T except for the root belongs to one maximal proper subtree S_m of T. For each S_m, we know that based on $d(S_m) \geq \lambda$ there are at least $\lambda \cdot n(S_m)$ edges in G_w that are incident to nodes from S_m. By summing up over all S_m and accounting for double counting, we conclude that the total number of edges in the current G_w and hence $m(T)$ is at least $\sum_{S_m} n(S_m) \cdot \lambda/2 = n(T) \cdot \lambda/2$. The latter follows as $w \in L(w)$ anyway and hence the root has a weight of zero and does not contribute to $n(T)$. In conclusion, we get $d(T) = m(T)/n(T) \geq \lambda/2$. As the algorithm returns the T with highest density over the course of the algorithm, the theorem follows. ◀

Note that this quality guarantee could not be shown if in each greedy step only the leaf of lowest density instead of the subtree of lowest density would be removed.

▶ **Lemma 17.** *Over the course of the set cover algorithm, the maximum density of a trimmed shortest path tree rooted in w can only decrease.*

Proof. Assume for contradiction that at some stage of the algorithm, the maximum density of a trimmed shortest path tree rooted at w is $D = d(T_1)$ but in some later round, there is a trimmed shortest path tree T_2 with $d(T_2) > D$. As the number of edges in G_w can only decrease over the course of the algorithm (as more and more node pairs are already covered), the only way to increase density for a subgraph is that some of its contained nodes already include w in their label sets (due to a selection of some set $S(w, W)$) and hence those nodes now have cost zero and do not contribute anymore to $n(T)$. We hence consider a moment where a tree T^* was selected as trimmed tree of largest density with $T^* \cap T_2 \neq \emptyset$. It follows that $d(T_2) \leq d(T^* \cup T_2)$ at this point and hence also $d(T^*) > d(T_2 \setminus T^*)$. But as the density of T_2 coincides with $d(T_2 \setminus T^*)$ after the removal of T^*, it follows that its density can not be larger than that of T^* at the time of its removal. ◀

Based on Theorem 16 and Lemma 17, the greedy set cover algorithm utilizing trimmed shortest path trees maintains an $\mathcal{O}(\log n)$ approximation guarantee.

Regarding the running time, we observe that determining a dense trimmed shortest path tree for root w requires $\mathcal{O}(n^2)$ time for initializing T and constructing G_w (assuming shortest path trees are precomputed and looking up whether a certain node pair is already covered takes constant time). Then there are $\mathcal{O}(n)$ rounds of greedy subtree removal. In each round, we compute the density of all subtrees of T and delete the one of smallest density from T an G_w, both of which can be done in $\mathcal{O}(n^2)$. Hence for each root w, the dense trimmed shortest path tree computation takes $\mathcal{O}(n^3)$. As we have to compute those trees for all nodes, this amounts to $\mathcal{O}(n^4)$ per iteration in the set cover algorithm. The set cover algorithm works then in the same way as the one described in [7] for HL. As our dense trimmed shortest path tree computations need a factor of n longer than the linear 2-approximation for DSG used in their approach, we end up with a total running time of $\mathcal{O}(n^6)$.

▶ **Theorem 18.** *A PC-LHL (as well as a PC-HL or PC-LL) with a total/average label size within a factor of $\mathcal{O}(\log n)$ of the optimal size can be computed in $\mathcal{O}(n^6)$.*

7 Hierarchical Labelings

A hierarchical labeling is a special kind of labeling $L : V \to 2^V$ that respects a given node ordering $r : V \to [n]$. Here, nodes in the label of a node have to be at least as important as the node itself, that is, for all $w \in L(v)$, we have $r(w) \geq r(v)$. For HHL, it was shown that for a given ordering r there is a simple rule that produces labels that are both necessary and sufficient for correct query answering. These are called *canonical* labels $L_c : V \to 2^V$. As discussed in Section 3.2, the canonical condition for HHL is that $w \in L_c(v)$ if and only if w is the node of highest rank on any shortest v-w-path. For any valid HHL L, it then yields $L(v) \supseteq L_c(v)$. Hence the theoretical study of optimal HHL sizes can be restricted to canonical HHL and practical algorithms should strive to produce labels close to the canonical ones. As proven in Lemma 9, a canonical HHL is always path-consistent. This implies that a canonical HHL is also a valid (hierarchical) PC-LHL. Therefore, label sizes in a (H)PC-LHL are at most as large as those in an HHL, and parametrized upper bounds for HHL transfer to (H)PC-LHL. But the hope is, of course, that HPC-LHL actually produces smaller labels than HHL. To investigate this, we now define the *canonical* HPC-LHL.

▶ **Definition 19** (Canonical HPC-LHL). *For a given node ordering $r : V \to [n]$, the* canonical *HPC-LHL that respects r assigns to each node v a label that consists of the the nodes of highest rank on each maximal geodesic that contains v.*

We first prove the sufficiency of the condition.

▶ **Lemma 20.** *A canonical HPC-LHL is a valid PC-LHL.*

Proof. Let s, t be any node pair and π^+ a maximal geodesic that contains both s and t (and is hence a superpath of the shortest s-t-path π). Let v be the node of highest rank on π^+. It follows by the canonical condition that $v \in L(s) \cap L(t)$. Accordingly, v is either a hub for s, t, in case $v \in \pi$ or it is a path landmark for s, t, in case $v \in \pi^+ \setminus \pi$. In both cases, a tight distance bound can be inferred from v. As all nodes on π^+ contain v in their labels, path-consistency is trivially obeyed. ◀

Next, we prove that the condition is also necessary.

▶ **Lemma 21.** *For a given node ordering $r : V \to [n]$, canonical hierarchical labels are a subset of any valid HPC-LHL.*

Proof. We want to show that for all $v \in V$, we have $L_c(v) \subseteq L(v)$ where L_c denotes the canonical HPC-LHL and L any valid HPC-LHL for the same r. Consider a $w \in L_c(v)$ and assume for contradiction that $w \notin L(v)$. Let Π be the set of maximal geodesics containing both v and w. We know that in this set there has to be at least one geodesic on which w has maximum rank for it to be included in $L_c(v)$. Consider such a geodesic π and let its end nodes be s and t. Based on the unique shortest path assumption, this is the only geodesic that contains both s and t. Therefore, to ensure correct query answering, s and t need to have a common node from π in their labels. And by the sake of the PC condition, all other nodes on π need to include the same node in their label. Hence, if we assume that this common node is not w but some other node w', we would demand that $w' \in L(w)$. But as w has maximum rank on π, including w' in $L(w)$ with $r(w') < r(w)$ would violate the definition of a hierarchical labeling. Accordingly, our initial assumption that $w \notin L(v)$ is wrong for all v where there exists a maximal geodesic containing v and w and where w has maximum rank on it. It follows that $w \in L_c(v) \Rightarrow w \in L(v)$. ◀

As the canonical condition is both necessary and sufficient, we know that among all valid HPC-LHL, the canonical one has the smallest possible label size. Note that for a given ranking $r : V \to [n]$, the canonical HPC-LHL can be computed in polytime by extracting the set of all maximal geodesics Π, determining the node of maximum rank on each $\pi \in \Pi$ and assigning that node to the label of all nodes in π.

8 Proof-Of-Concept Study

While it is out of scope of this paper to provide an extensive experimental study and to develop engineered algorithms for HPC-LHL that scale to large networks, we provide a small feasibility study to see whether the concept might have practical merits. For that purpose, we implemented the naive algorithm to compute a canonical HPC-LHL as well as the respective query routine in C++. Experiments were conducted on a single core of an AMD Ryzen Threadripper 3970X 32-Core Processor clocked at 2.061GHz with 256 GB RAM.

As benchmark networks, we used random rectangular cutouts of the European road network extracted from OpenStreetMap[1], restricted to the largest strongly connected component contained therein. Because the computation of all maximal geodesics is space- and time-consuming, we restricted ourselves to subgraphs of up to 100,000 nodes. Then we computed the canonical HHL based on [4] and used the resulting ordering r as basis for HPC-LHL; hence both labelings use the same node hierarchy. We observed that the average label size in the HPC-LHL was about 5% to 12% smaller than the respective label size in the HHL. We expect this gap to increase in larger networks where we have longer chains of degree-2 nodes on which HPC-LHL needs provably less label nodes than HHL. Furthermore, we analyzed and compared the different query answering methods for 10,000 random shortest path queries in each network. For distance queries, both methods answer queries within a few microseconds. HHL query answering is slightly superior to HPC-LHL query answering as it never requires path retrieval. We found that for HPC-LHL in about 88% of the cases the upper bound was tight, in 11% of the cases both upper and lower bound, and in the remaining 1% solely the lower bound. However, as path retrieval is very fast when exploiting

[1] `https://i11www.iti.kit.edu/resources/roadgraphs.php`

the PC property, the query times did never differ by more than a factor of 1.4 and on average by less than two percent. For path queries, however, using the PC-property allows for a speed-up of a factor of roughly 500. Here, both HHL and HPC-LHL can benefit from the speed-up based on our proof that HHL are always PC. We conclude that at least on the investigated graphs, HPC-LHL allows to produce smaller label sizes with a negligible increase in query time for distance queries. Furthermore, we have clearly demonstrated that the PC property is very useful in practice as it enables significantly accelerated shortest path retrieval. This effect should also be even more pronounced in bigger graphs with larger diameters.

9 Conclusions and Future Work

We proposed to augment the Hub Labeling idea with lower bound information and introduced Landmark Hub Labeling (LHL) as an alternative to classical labelings. As LHL allows for a greater variety of valid node labels than standard (hierarchical) Hub Labeling, label sizes are potentially smaller. This is important as one of the main drawbacks of labeling approaches in practical applications is the huge space consumption. We showed interesting theoretical properties of (path-consistent) LHL and conducted an initial feasibility study. To make the PC-LHL applicable to larger graphs, better scalable algorithms have to be designed that nevertheless compute concise labels. Hierarchical Hub Labels can be efficiently infered from a Contraction Hierarchy (CH) data structure [22]. It might be possible to also exploit CH for PC-LHL construction or to modify and prune labels from a given HHL.

On the theoretical side, it would be interesting to investigate whether the logarithmic gap between HL and PC-LHL on path graphs is tight or whether there exist other types of graphs with a larger separation. Furthermore, the $\mathcal{O}(n^6)$ running time of the approximation algorithm can potentially be reduced by incorporating ideas from [16], which were used to improve the running time of the HL approximation algorithm from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^3 \log n)$. Regarding graph parameters, we showed that label sizes in PC-LHL can be upper bounded by functions depending on the highway dimension or the geodesic transversal number. Another parameter to investaiget would be the skeletoon dimension, which was used in previous work to also uppper bound label sizes of (non-hierarchical) HL. Also there are no parametrized bounds for Landmark Labeling so far.

Finally, PC-LHL might provide a good basis for an approximate distance oracle by weakening the property that either the label based lower or upper bound needs to be tight. Instead, one could demand that the (absolute or relative) difference of the two bounds is below a certain threshold which would then automatically provide a quality guarantee.

References

1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Vc-dimension and shortest path algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 690–699. Springer, 2011.

2 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension and provably efficient shortest path algorithms. *Journal of the ACM (JACM)*, 63(5):1–26, 2016.

3 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Proc. 10th Int. Symp. Experimental Algorithms (SEA '11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011. doi:10.1007/978-3-642-20662-7_20.

4 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Hierarchical hub labelings for shortest paths. In *Proc. 20th Ann. Europ. Symp. Algorithms (ESA '12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012. doi:10.1007/978-3-642-33090-2_4.

5 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD '13)*, pages 349–360. ACM, 2013. doi:10.1145/2463676.2465315.

6 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *Proc. 23rd Int. Conf. World Wide Web (WWW '14)*, pages 237–248. ACM, 2014. doi:10.1145/2566486.2568007.

7 Maxim A. Babenko, Andrew V. Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller. On the complexity of hub labeling (extended abstract). In *Proc. 40th Int. Symp. Mathematical Foundations of Computer Science (MFCS '15)*, volume 9235 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 2015. doi:10.1007/978-3-662-48054-0_6.

8 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

9 Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing speed-up techniques is hard. In *International Conference on Algorithms and Complexity*, pages 359–370. Springer, 2010.

10 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016.

11 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, 15:2–1, 2010.

12 Johannes Blum. Hierarchy of transportation network parameters and hardness results. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

13 Zitong Chen, Ada Wai-Chee Fu, Minhao Jiang, Eric Lo, and Pengfei Zhang. P2H: efficient distance querying on road networks by projected vertex separators. In *Proc. 2021 Int. Conf. Management of Data (SIGMOD '21)*, pages 313–325. ACM, 2021. doi:10.1145/3448016.3459245.

14 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003. doi:10.1137/S0097539702403098.

15 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust distance queries on massive networks. In *Proc. 22th Ann. Europ. Symp. Algorithms (ESA '14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, 2014. doi:10.1007/978-3-662-44777-2_27.

16 Daniel Delling, Andrew V. Goldberg, Ruslan Savchenko, and Renato F. Werneck. Hub labels: Theory and practice. In *Proc. 13th Int. Symp. Experimental Algorithms (SEA '14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2014. doi:10.1007/978-3-319-07959-2_22.

17 Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hub label compression. In *Proc. 12th Int. Symp. Experimental Algorithms (SEA '13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2013. doi:10.1007/978-3-642-38527-8_4.

18 Daniel Delling and Giacomo Nannicini. Core routing on dynamic time-dependent road networks. *INFORMS Journal on Computing*, 24(2):187–201, 2012.

19 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Highway hierarchies star. In *The Shortest Path Problem*, pages 141–174, 2006.

20 Daniel Delling and Dorothea Wagner. Landmark-based routing in dynamic graphs. In *International Workshop on Experimental and Efficient Algorithms*, pages 52–65. Springer, 2007.

21 Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 25–30, 2013.

22 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

23 Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, pages 156–165. Citeseer, 2005.

24 Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Better landmarks within reach. In *International Workshop on Experimental and Efficient Algorithms*, pages 38–51. Springer, 2007.

25 Andrew V Goldberg and Renato Fonseca F Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

26 Daniel Harabor and Peter Stuckey. Forward search in contraction hierarchies. In *International Symposium on Combinatorial Search*, volume 9(1), 2018.

27 Famei He, Yina Xu, Xuren Wang, and Anran Feng. Alt-based route planning in dynamic time-dependent road networks. In *Proceedings of the 2019 2nd International Conference on Machine Learning and Machine Intelligence*, pages 35–39, 2019.

28 Samir Khuller and Barna Saha. On finding dense subgraphs. In *International colloquium on automata, languages, and programming*, pages 597–608. Springer, 2009.

29 Adrian Kosowski and Laurent Viennot. Beyond highway dimension: small distance labels using tree skeletons. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1462–1478. SIAM, 2017.

30 Lei Li, Sibo Wang, and Xiaofang Zhou. Time-dependent hop labeling on road network. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 902–913. IEEE, 2019.

31 Paul Manuel, Boštjan Brešar, and Sandi Klavžar. The geodesic-transversal problem. *Applied Mathematics and Computation*, 413:126621, 2022.

32 Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *Proc. 2018 Int. Conf. Management of Data (SIGMOD '18)*, pages 709–724. ACM, 2018. doi:10.1145/3183713.3196913.

33 Genaro Peque, Junji Urata, and Takamasa Iryo. Implementing an alt algorithm for large-scale time-dependent networks. In *22nd International Conference of Hong Kong Society for Transportation Studies: Transport and Society, HKSTS 2017*, pages 515–522. Hong Kong Society for Transportation Studies Limited, 2017.

34 Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 867–876, 2009.

An Optimal Oracle Separation of Classical and Quantum Hybrid Schemes

Atsuya Hasegawa
Graduate School of Information Science and Technology, The University of Tokyo, Japan

François Le Gall
Graduate School of Mathematics, Nagoya University, Japan

Abstract

Recently, Chia, Chung and Lai (STOC 2020) and Coudron and Menda (STOC 2020) have shown that there exists an oracle $\mathcal{O}$ such that $\mathsf{BQP}^{\mathcal{O}} \neq (\mathsf{BPP}^{\mathsf{BQNC}})^{\mathcal{O}} \cup (\mathsf{BQNC}^{\mathsf{BPP}})^{\mathcal{O}}$. In fact, Chia et al. proved a stronger statement: for any depth parameter d, there exists an oracle that separates quantum depth d and $2d+1$, when polynomial-time classical computation is allowed. This implies that relative to an oracle, doubling quantum depth gives classical and quantum hybrid schemes more computational power.

In this paper, we show that for any depth parameter d, there exists an oracle that separates quantum depth d and $d+1$, when polynomial-time classical computation is allowed. This gives an optimal oracle separation of classical and quantum hybrid schemes. To prove our result, we consider d-Bijective Shuffling Simon's Problem (which is a variant of d-Shuffling Simon's Problem considered by Chia et al.) and an oracle inspired by an "in-place" permutation oracle.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases small-depth quantum circuit, hybrid quantum computer, oracle separation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.6

Related Version *arXiv Version*: https://arxiv.org/abs/2205.04633

Funding JSPS KAKENHI grants Nos. JP16H01705, JP19H04066, JP20H00579, JP20H04139, JP22J22563 and the MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) grant No. JPMXS0120319794.

Acknowledgements The authors are grateful to Nai-Hui Chia for helpful discussions. They are also grateful to Atul Singh Arora, Alexandru Gheorghiu and Uttam Singh for sharing the information about their result [7].

1 Introduction

Background

In recent years, the development of quantum computers has been very active (see, e.g., [1] for information about current quantum computers) and "quantum supremacy" has been claimed [8, 22]. However, it is still difficult to implement large-depth quantum circuits with current quantum technology since such quantum devices are subjective to noise and have short coherent time. One potential way to extract the computational powers of such quantum devices is to consider a hybrid scheme combining them with classical computers. For example, variational quantum algorithms are considered in such a scheme to obtain quantum advantage (see [9] for a survey).

Therefore, understanding the capabilities and limits of this hybrid approach is an essential topic in quantum computation. As one of the most notable results, Cleve and Watrous [14] showed the quantum Fourier transformation can be implemented by combining logarithmic-depth quantum circuits with a classical polynomial-time algorithm. With the

possibility to implement Shor's algorithm in such a hybrid scheme and the developments of measurement-based quantum computation, Jozsa [18] conjectured that "Any quantum polynomial-time algorithm can be implemented with only $O(\log n)$ quantum depth interspersed with polynomial-time algorithm classical computations". This can be formalized as $\mathsf{BQP} = \mathsf{BQNC}^{\mathsf{BPP}}$. On the other hand, Aaronson [3, 4, 5] conjectured "there exists an oracle separation between BQP and $\mathsf{BPP}^{\mathsf{BQNC}}$". $\mathsf{BPP}^{\mathsf{BQNC}}$ is a complexity class recognized by a polynomial classical scheme which has access to poly-logarithmic depth quantum circuits. $\mathsf{BQNC}^{\mathsf{BPP}}$ and $\mathsf{BPP}^{\mathsf{BQNC}}$ are sets of problems recognized by two natural and seemingly incomparable models of hybrid classical and quantum computation.

Recent works by Chia, Chung and Lai [11] and Coudron and Menda [15] proved Aaronson's conjecture and refuted Jozsa's conjecture in a relativized setting. Interestingly, computational problems and oracles they considered were completely different. Coudron and Menda [15] considered, as an oracle problem, the Welded Tree Problem which exhibits a difference between quantum walks and classical random walks: this problem can be solved efficiently by a quantum algorithm [13] but, in the classical setting, exponential queries are required [13, 17]. To prove a lower bound of classical and quantum hybrid schemes, Coudron and Menda introduced "Information Bottleneck" to simulate classical and quantum hybrid schemes with fewer classical queries. They showed if we assume the hybrid schemes solve the problem, we reach a contradiction with the lower bound of classical queries from [17].

Chia, Chung and Lai [11] considered d-Shuffling Simon's Problem, which is a variant of Simon's Problem [20]. Since Simon's Problem can be solved with a constant-depth quantum circuit with classical post-processing, we cannot prove the hardness for classical and quantum hybrid schemes. To devise a harder problem, they combine Simon's function with sequential random permutations: for a Simon's function f, they consider random one-to-one functions $f_0, ..., f_{d-1}$ and two-to-one function f_d such that $f = f_d \circ \cdots \circ f_0$. They also hide the domains of the functions in larger domains and apply the idea of the Oneway-to-Hiding (O2H) lemma [6, 21] to prove the hardness. In fact, they proved a stronger statement below.

▶ **Theorem 1** ([11]). *For any $d \in \mathbb{N}$, there exists an oracle $\mathcal{O}$ such that*

$$(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}} \cup (\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}} \neq (\mathsf{BQNC}_{2d+1}^{\mathsf{BPP}})^{\mathcal{O}} \cap (\mathsf{BPP}^{\mathsf{BQNC}_{2d+1}})^{\mathcal{O}}.$$

Description of our result

In this paper, we improve Theorem 1 above and show the following result.

▶ **Theorem 2.** *For any $d \in \mathbb{N}$, there exists an oracle $\mathcal{O}$ such that*

$$(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}} \cup (\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}} \neq (\mathsf{BQNC}_{d+1}^{\mathsf{BPP}})^{\mathcal{O}} \cap (\mathsf{BPP}^{\mathsf{BQNC}_{d+1}})^{\mathcal{O}}.$$

Our result implies that, relative to an oracle, increasing the quantum depth even by *one* gives the hybrid schemes more computational power and it cannot be traded by combining polynomial-time classical processing.

In Theorem 2, quantum circuits consisting of any 1- and 2-qubit gates are considered. Indeed, we give an algorithm by $d+1$-depth quantum circuits consisting only of $\{H, CNOT\}$ with classical processing for the upper bound (this is also the case for Theorem 1 but not mentioned in [11]). Therefore we also prove that, even if we are allowed to use quantum circuits consisting of a restricted gate set contains $\{H, CNOT\}$ such as Clifford circuits, adding even one quantum depth gives the two hybrid schemes more computational power relative to an oracle.

Outline of our approach

Chia et al. gave the upper bound $(\mathsf{BQNC}_{2d+1}^{\mathsf{BPP}})^{\mathcal{O}} \cap (\mathsf{BPP}^{\mathsf{BQNC}_{2d+1}})^{\mathcal{O}}$ for d-Shuffling Simon's Problem by an algorithm inspired by the Simon's algorithm. Since they considered a standard oracle, $U_f |x\rangle |0\rangle = |x\rangle |f(x)\rangle$, it is required to erase the information of past queries and it takes d-quantum depth. To eliminate the d-quantum depth, we propose an idea to consider an "in-place" permutation oracle [2, 16] acts as $U_f |x\rangle = |f(x)\rangle$. However, when f is a Simon's function, f_d on a restricted domain is also a two-to-one function and there is no unitary operator U_{f_d} such that $U_{f_d} |x\rangle = |f_d(x)\rangle$. Therefore, in this paper, we consider another function η and make the function bijective. We name the problem d-Bijective Shuffling Simon's Problem and show an upper bound $(\mathsf{BQNC}_{d+1}^{\mathsf{BPP}})^{\mathcal{O}} \cap (\mathsf{BPP}^{\mathsf{BQNC}_{d+1}})^{\mathcal{O}}$. The other obstacle is, for f_d and the shadows to prove the lower bounds, how to define a unitary operator that includes mappings to $\perp$ (a constant with no information). Note that this is because there exists no unitary operator $U_\perp$ such that $U_\perp |x\rangle = |\perp\rangle$. In this paper, we give a solution by keeping values on domains and considering "flags" on ancilla qubits. Finally we carefully tailor the Oneway-to-Hiding lemma in our quantum oracle setting and show that the similar proofs of the lower bounds also follow as [11].

Related work

Arora, Gheorghiu and Singh [7] proved oracle separations of $(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$ and $(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$ with respect to each other. As corollaries, they obtained sharper separations than [11] for each scheme. For the quantum-classical scheme, they proved an oracle separation between quantum depth d and $d+1$ if the Hadamard measurements are allowed in every layer. In our result, we only need to measure qubits in the Hadamard basis in the last layer. For the classical-quantum scheme, they proved a separation between quantum depth d and $d+5$ relative to what they call a stochastic oracle (which is non-unitary). Our separation is between quantum depth d and $d+1$ relative to a unitary oracle.

In an independent work [12], Chia and Hung have also shown how to reduce the gap from d versus $2d+1$ to d versus $d+1$ by techniques similar to ours (they consider an oracle inspired by an "in-place" permutation oracle and manage to make the final function one-to-one). They also instantiate the oracle separation to construct a protocol such that a classical verifier can check if a prover has a quantum depth of at least $d+1$.

Organization of the paper

After giving preliminaries in Section 2, in Section 3 we define the oracle problem that we call d-Bijective Shuffling Simon's Problem and give the upper bound $(\mathsf{BQNC}_{d+1}^{\mathsf{BPP}})^{\mathcal{O}} \cap (\mathsf{BPP}^{\mathsf{BQNC}_{d+1}})^{\mathcal{O}}$. In Section 4, we prove the Oneway-to-Hiding lemma for our quantum oracle and, in Section 5, we prove the lower bounds for $(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$ and $(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$.

The main contribution of our work is to define the d-Bijective Shuffling Simon's Problem and give the upper bound with quantum depth $d+1$ (Section 3). The proof of the lower bound (Section 5) is very similar to [11] except the Oneway-to-Hiding lemma (Section 4), which has to be adapted to the quantum oracle of this paper. We are grateful to Nai-Hui Chia for discussions about this, and in particular for clarifying that all steps in the lower bound from [11] remain true for our new oracle as well, with the exception of this Oneway-to-Hiding lemma.

2 Preliminaries

2.1 State distances

Let us recall some notions about the distances of quantum states [19].

▶ **Definition 3.** *For any two mixed states ρ and σ,*

- *(Fidelity)* $F(\rho,\sigma) := \mathrm{tr}(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}})$.
- *(Bures distance)* $B(\rho,\sigma) := \sqrt{2-2F(\rho,\sigma)}$.

▷ Claim 4. For any two mixed states ρ and σ, any quantum algorithm $\mathcal{A}$ and any classical string s,

$$|\Pr[\mathcal{A}(\rho)=s]-\Pr[\mathcal{A}(\sigma)=s]| \leq B(\rho,\sigma).$$

2.2 Computational models

Let us introduce computational models in this paper, especially two relativized hybrid schemes of quantum and classical computations.

2.2.1 Quantum circuits

First, we define d-depth quantum circuits and promise problems solved by the computational schemes. We refer to [19] for a standard reference of quantum computation and circuits. We define a one-layer (depth) unitary is a set of one- and two-qubit gates act on disjoint sets of qubits. A promise problem denotes a pair $L=(L_{\mathrm{yes}},L_{\mathrm{no}})$, where $L_{\mathrm{yes}},L_{\mathrm{no}}\subseteq\Sigma^*$ are sets of strings satisfying $L_{\mathrm{yes}}\cap L_{\mathrm{no}}=\emptyset$.

▶ **Definition 5** (d-depth quantum circuit, QNC_d)**.** *QNC_d is the class of all quantum circuit family $\{C_n : n\in\mathbb{N}\}$ satisfying that C_n acts on n input qubits and $poly(n)$ ancilla qubits, and consists of successive d-layers of arbitrary one- and two-qubit gates, $U_1U_2...U_d$, and measure all qubits in the computational basis.*

▶ **Definition 6** (BQNC_d)**.** *A promise problem L is in BQNC_d if and only if there exists a circuit family $\{C_n : n\in\mathbb{N}\}\in\mathsf{QNC}_d$ satisfying the following properties:*

- *for all $x\in L_{\mathrm{yes}}$, $\Pr[C_{|x|}(x)=1]\geq\frac{2}{3}$;*
- *for all $x\in L_{\mathrm{no}}$, $\Pr[C_{|x|}(x)=1]\leq\frac{1}{3}$.*

In the definition above, we consider arbitrary one- and two-qubit gates. In this paper, we also consider a gate set $\{H,CNOT\}$, where $H=\frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1&1\\1&-1\end{smallmatrix}\right)$ is the Hadamard gate and $CNOT=\left(\begin{smallmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{smallmatrix}\right)$ is the controlled-not gate. A quantum circuit consisting only of Clifford gates $\{H,S,CNOT\}$, where $S=\left(\begin{smallmatrix}1&0\\0&i\end{smallmatrix}\right)$ is the phase gate, is called a Clifford circuit.

For an oracle $\mathcal{O}$, let us define $(\mathsf{QNC}_d)^{\mathcal{O}}$ be a computational scheme similar to QNC_d considering $U_{d+1}\mathcal{O}U_d...\mathcal{O}U_1$ as layers and $(\mathsf{BQNC}_d)^{\mathcal{O}}$ be a set of promise problems solved by $(\mathsf{QNC}_d)^{\mathcal{O}}$ with high probability. Note that for relativized d-depth quantum circuit, we consider to add an extra single layer to process the final oracle access following [11]. We refer to Figure 1 for an illustration.

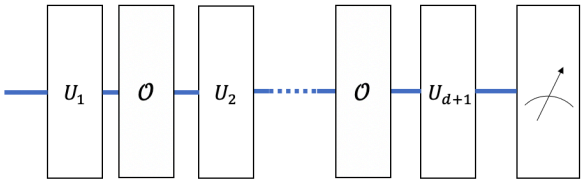


Figure 1 d-depth quantum circuit which can access to an oracle $\mathcal{O}$.

2.2.2 Quantum-classical hybrid schemes

Next, we consider the d-quantum-classical scheme, which is a generalized model for d-depth measurement-based quantum computation. This is the same model as in [11]. The scheme is denoted as d-QC scheme and we represent it as the following sequence:

$$\left(\mathcal{A}_c \xrightarrow{c} (\textstyle\prod_{0/1} \otimes I) \circ U_1\right) \xrightarrow{c,q} \left(\mathcal{A}_c \xrightarrow{c} (\textstyle\prod_{0/1} \otimes I) \circ U_2\right) \xrightarrow{c,q} (\mathcal{A}_c \cdots \circ U_d) \xrightarrow{c} \mathcal{A}_c,$$

where $\mathcal{A}_c$ is a classical probabilistic polynomial-time algorithm, U_i is a one-depth quantum layer and $\prod_{0/1}$ is a measurement in the computational basis or the Hadamard basis[1]. The arrows $\xrightarrow{c}$ and $\xrightarrow{q}$ represent transmissions of polynomial-size classical and quantum bits. Between quantum layers U_i and U_{i+1}, $\mathcal{A}_c$ can use measurement results of an arbitrary part of qubits after U_i applies and send classical polynomial-size information to U_{i+1}. We refer to Figure 2 for an illustration.

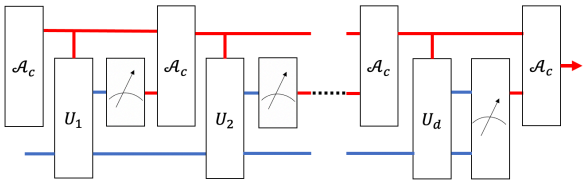


Figure 2 The d-QC scheme. Red lines stand for classical wires and blue ones stand for quantum wires.

Let us consider a relativized version of the scheme. Let $\mathcal{A}^{\mathcal{O}}$ be a d-QC scheme with access to an oracle $\mathcal{O}$. We represent the relativized d-QC scheme $\mathcal{A}^{\mathcal{O}}$ as a sequence of operators:

$$(L_1)^{\mathcal{O}} \xrightarrow{c,q} \cdots \xrightarrow{c,q} (L_d)^{\mathcal{O}} \xrightarrow{c} \mathcal{A}_c^{\mathcal{O}},$$

where $\mathcal{A}_c^{\mathcal{O}}$ is a classical polynomial-time algorithm which can query to the oracle $\mathcal{O}$, and $(L_i)^{\mathcal{O}} := \mathcal{A}_c^{\mathcal{O}} \xrightarrow{c} (\prod_{0/1} \otimes I) \circ \mathcal{O}U_i$. We refer to Figure 3 for an illustration. Then, we define the set of promise problems which can be solved by the relativized d-QC schemes.

▶ **Definition 7** ($(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$, Definition 3.8 in [11]). *A promise problem L is in $(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$ if and only if there exists a family of relativized d-QC schemes $\{\mathcal{A}_n^{\mathcal{O}} : n \in \mathbb{N}\}$ satisfying the following properties:*

- *for all $x \in L_{\text{yes}}$, $\Pr[\mathcal{A}_{|x|}^{\mathcal{O}}(x) = 1] \geq \frac{2}{3}$;*
- *for all $x \in L_{\text{no}}$, $\Pr[\mathcal{A}_{|x|}^{\mathcal{O}}(x) = 1] \leq \frac{1}{3}$.*

[1] Without the Hadamard measurements, our upper bound becomes quantum depth $d+2$ for the quantum-classical hybrid scheme. The upper bound of Theorem 1 for the scheme also becomes quantum depth $2d+2$ without the Hadamard measurements.

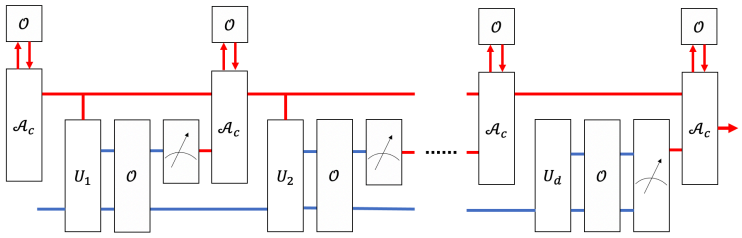


Figure 3 The d-QC scheme with access to an oracle $\mathcal{O}$.

2.2.3 Classical-quantum hybrid schemes

Finally, we define the d-classical-quantum scheme, which is a classical polynomial-time algorithm which has access to a d-depth quantum circuit during the computation (up to polynomial times). This is the same model as in [11]. The scheme is denoted as d-CQ scheme and represented as follows:

$$\mathcal{A}_{c,1} \xrightarrow{c} \prod_{0/1} \circ U_d \cdots U_1 \xrightarrow{c} \cdots \xrightarrow{c} \mathcal{A}_{c,m-1} \xrightarrow{c} \prod_{0/1} \circ U_d \cdots U_1 \xrightarrow{c} \mathcal{A}_{c,m},$$

where m is a polynomial in n, $\mathcal{A}_{c,i}$ is an ith classical probabilistic polynomial-time algorithm, U_i is a one-depth quantum layer, and $\prod_{0/1}$ is the computational basis measurement. Each $\mathcal{A}_{c,i+1}$ can depend on the polynomial-size information sent from $\mathcal{A}_{c,i}$ and the measurement results of the d-depth quantum circuit $\mathcal{A}_{c,i}$ calls. We refer to Figure 4 for an illustration.

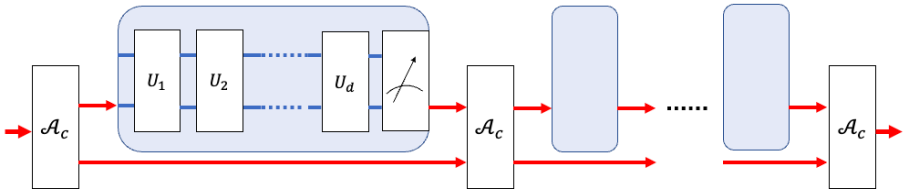


Figure 4 The d-CQ scheme. Red lines stand for classical wires and blue ones stand for quantum wires.

Let us consider a relativized version of the scheme. Let $\mathcal{A}^{\mathcal{O}}$ be a d-CQ scheme with an access to an oracle $\mathcal{O}$. We represent $\mathcal{A}^{\mathcal{O}}$ as follows:

$$(L_1)^{\mathcal{O}} \xrightarrow{c} (L_2)^{\mathcal{O}} \xrightarrow{c} \cdots \xrightarrow{c} (L_{m-1})^{\mathcal{O}} \xrightarrow{c} (\mathcal{A}_{c,m})^{\mathcal{O}},$$

where $\mathcal{A}_{c,i}$ is an ith classical polynomial-time algorithm which can query to the oracle $\mathcal{O}$, and $(L_i)^{\mathcal{O}} := (\mathcal{A}_{c,i})^{\mathcal{O}} \xrightarrow{c} \prod_{0/1} \circ (U_{d+1}\mathcal{O}U_d \cdots \mathcal{O}U_1)$. We refer to Figure 5 for an illustration. Then, we define the set of promise problems which can be solved by the relativized d-CQ schemes.

▶ **Definition 8** ($(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$, Definition 3.10 in [11]). *A promise problem L is in $(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$ if and only if there exists a family of relativized d-CQ schemes $\{\mathcal{A}_n^{\mathcal{O}} : n \in \mathbb{N}\}$ satisfying the following properties:*

- *for all $x \in L_{\text{yes}}$, $\Pr[\mathcal{A}_{|x|}^{\mathcal{O}}(x) = 1] \geq \frac{2}{3}$;*
- *for all $x \in L_{\text{no}}$, $\Pr[\mathcal{A}_{|x|}^{\mathcal{O}}(x) = 1] \leq \frac{1}{3}$.*

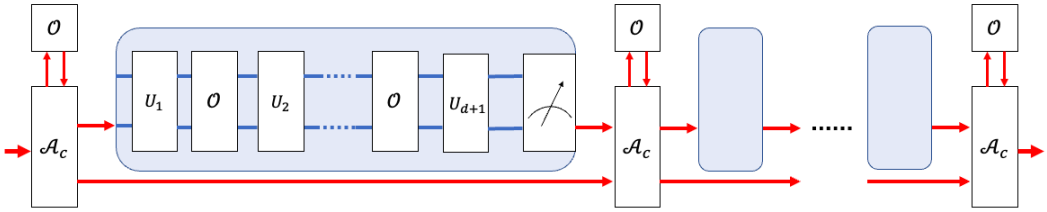


Figure 5 The d-CQ scheme with access to an oracle $\mathcal{O}$.

2.3 Simon's problem

Let us recall the definitions of Simon's function and Simon's problem [20].

▶ **Definition 9** (Simon's function). *A two-to-one function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ for $n \in \mathbb{N}$ is a Simon's function if there exists a period $s \in \mathbb{Z}_2^n$ such that $f(x) = f(x \oplus s)$ for $x \in \mathbb{Z}_2^n$. Let $\mathbf{F}$ be the set of all Simon's functions from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$.*

▶ **Definition 10** (Simon's problem). *Given f chosen from $\mathbf{F}$ uniformly at random, the problem is to obtain the period s.*

▶ **Definition 11** (Decision Simon's problem). *Given f to be either a random Simon's function from $\mathbf{F}$ or a random one-to-one function from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$ with equal probability, the promise problem is to distinguish the two cases.*

3 d-Bijective Shuffling Simon's Problem (d-BSSP)

In this section, we introduce the oracle problem, the d-Bijective Shuffling Simon's Problem, which is abbreviated as d-BSSP in the rest of the paper. It is similar to d-Shuffling Simon's Problem (d-SSP) in [11] but there are several modifications of quantum oracles to make the upper bound $(\mathsf{BPP}^{\mathsf{BQNC}_{d+1}})^{\mathcal{O}} \cap (\mathsf{BQNC}_{d+1}^{\mathsf{BPP}})^{\mathcal{O}}$.

Let us consider shufflings of functions. For any two set X and Y, let $P(X, Y)$ be the set of one-to-one functions from X to Y. In this work, it is enough to consider random shufflings of one-to-one functions and Simon's functions.

▶ **Definition 12** ((d, f)-shuffling, Definition 4.1 in [11]). *Let $d \in \mathbb{N}$ and f be a one-to-one function or a Simon's function from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$. A (d, f)-shuffling is $\mathcal{F} := (f_0, f_1, ..., f_d)$, where $f_0, ..., f_{d-1}$ are chosen uniformly at random from $P(\mathbb{Z}_2^{(d+2)n}, \mathbb{Z}_2^{(d+2)n})$ and $f_d : \mathbb{Z}_2^{(d+2)n} \to \mathbb{Z}_2^{(d+2)n}$ is a function satisfying the following properties: Let $S_d := \{f_{d-1} \circ \cdots \circ f_0(x') : x' = 0, ..., 2^n - 1\}$.*

- *For $x \in S_d$, $f_d(x) = f \circ f_0^{-1} \circ \cdots \circ f_{d-1}^{-1}(x)$.*
- *For $x \notin S_d$, $f_d(x) = \perp$.*

Let $\mathbf{SHUF}(d, f)$ be a set of all (d, f)-shuffling functions.

Note that the way to choose f_d is unique for each $f_0, ..., f_{d-1}$.

We consider a standard oracle for f_0 and "in-place" permutation oracles for $f_1, ..., f_{d-1}, f_d$. When f is a Simon's function, f_d on S_d is also two-to-one and there exists no quantum oracle to implement f_d "in-place" since a two-to-one function is not a unitary operator. Therefore, we consider a function η to make the mapping bijective. We also consider a function ζ which represents whether x_d is in S_d.

▶ **Definition 13** ((d,f)-bijective shuffling)**.** *For a (d,f)-shuffling $\mathcal{F} = (f_0, f_1, ..., f_d)$, a (d,f)-bijective shuffling is $\mathcal{F}_b := (f_0, f_1, ..., f_{d-1}, f'_d, \zeta, \eta)$, where $f'_d : \mathbb{Z}_2^{(d+2)n} \to \mathbb{Z}_2^{(d+2)n}$, $\zeta : \mathbb{Z}_2^{(d+2)n} \to \mathbb{Z}_2$ and $\eta : \mathbb{Z}_2^{(d+2)n} \to \mathbb{Z}_2$ satisfying the following properties: Let $S_d := \{f_{d-1} \circ \cdots \circ f_0(x') : x' = 0, ..., 2^n - 1\}$.*

- *For $x \in S_d$, $f'_d(x) = f_d(x)$ and $\zeta(x) = 1$*
- *For $x \notin S_d$, $f'_d(x) = x$ and $\zeta(x) = 0$*
- *If f is a Simon's function,*
  - *For $x \in S_d$, $\eta(x) = \eta(x') \oplus 1$ where x' is the unique element in S_d such that $f_d(x) = f_d(x')$.*
  - *For $x \notin S_d$, $\eta(x) = 1$*

  Otherwise,
  - *For $x \in S_d$, $\eta(x)$ is chosen uniformly at random from $\{0,1\}$*
  - *For $x \notin S_d$, $\eta(x) = 1$*

Let $\mathbf{BSHUF}(d,f)$ be a set of all (d,f)-bijective shuffling functions.

For fixed $d \in \mathbb{N}$ and a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$, we can define a random oracle which is a (d,f)-bijective shuffling chosen uniformly randomly from $\mathbf{BSHUF}(d,f)$.

▶ **Definition 14** (Bijective shuffling oracle $\mathcal{O}_{\mathrm{unif}}^{f,d}$)**.** *Let $d, n \in \mathbb{N}$. Let f be a Simon's function or a one-to-one function from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$. The bijective shuffling oracle $\mathcal{O}_{\mathrm{unif}}^{f,d}$ is a (d,f)-bijective shuffling $\mathcal{F}_b$ uniformly chosen from $\mathbf{BSHUF}(d,f)$.*

If we sample a (d,f)-bijective shuffling uniformly randomly from $\mathbf{BSHUF}(d,f)$, the permutations $f_0, ..., f_{d-1}$ are uniformly distributed in $P(\mathbb{Z}_2^{(d+2)n}, \mathbb{Z}_2^{(d+2)n})$ independently of f. This is because for each (d,f)-shuffling, f'_d and ζ are uniquely chosen and the number of ways to choose η and (d,f)-bijective shufflings is the same. From the definition of $\mathbf{BSHUF}(d,f)$, for such a (d,f)-bijective shuffling $\mathcal{F}_b$, only f'_d and η on S_d encodes the information of f.

Next, let us define the quantum oracle access to the bijective shuffling oracle $\mathcal{O}_{\mathrm{unif}}^{f,d}$. In this paper, as in [11], we represent the input quantum state $|\phi\rangle$ to $\mathcal{O}_{\mathrm{unif}}^{f,d}$ as follows:

$$|\phi\rangle := \sum_{\mathbf{X}_0,...,\mathbf{X}_d} c(\mathbf{X}_0, ..., \mathbf{X}_d) \left(\bigotimes_{i=0}^{d} |i, \mathbf{X}_i\rangle \right)_{\mathbf{R}_Q} \otimes |0\rangle_{\mathbf{R}_N} \otimes |w(\mathbf{X}_0, ..., \mathbf{X}_d)\rangle_{\mathbf{R}_W},$$

where $\mathbf{X}_i$ is a set of elements in the domain of the functions $f_0, ..., f_{d-1}, f'_d$ and $c(\mathbf{X}_0, ..., \mathbf{X}_d)$ is an arbitary coefficient and $|w(\mathbf{X}_0, ..., \mathbf{X}_d)\rangle$ is an arbitrary working state for quantum layers, U_is. By the access to the quantum oracle, the parallel queries in the register $\mathbf{R}_Q$ and the ancilla qubits in the register $\mathbf{R}_N$ are processed, while the remaining qubits in the register $\mathbf{R}_W$ are unchanged.

For $\mathcal{F}_b \in \mathbf{BSHUF}(d,f)$, we define

$$\mathcal{F}_b |\phi\rangle := \sum_{\mathbf{X}_0,...,\mathbf{X}_d} c(\mathbf{X}_0, ..., \mathbf{X}_d) \Bigg(|0, \mathbf{X}_0\rangle |f_0(\mathbf{X}_0)\rangle \otimes \bigotimes_{i=1}^{d-1} |i, f_i(\mathbf{X}_i)\rangle |0\rangle \otimes |d, f'_d(\mathbf{X}_d)\rangle |\zeta(\mathbf{X}_d)\rangle |\eta(\mathbf{X}_d)\rangle |0\rangle \Bigg)_{\mathbf{R}_Q, \mathbf{R}_N} \otimes |w(\mathbf{X}_0, ..., \mathbf{X}_d)\rangle_{\mathbf{R}_W}.$$

Note that the ancilla qubits $|0\rangle$ are required to define the shadow in Section 4. We also define applying $\mathcal{O}_{\mathrm{unif}}^{f,d}$ on $|\phi\rangle$ as

$$\mathcal{O}_{\mathrm{unif}}^{f,d}(|\phi\rangle\langle\phi|) := \sum_{\mathcal{F}_b \in \mathbf{BSHUF}(d,f)} \frac{1}{|\mathbf{BSHUF}(d,f)|} \mathcal{F}_b |\phi\rangle\langle\phi| \mathcal{F}_b.$$

Now we can define the d-Bijective Shuffling Simon's Problem (d-BSSP).

▶ **Definition 15** (The search d-Bijective Shuffling Simon's Problem). *Let $d \in \mathbb{N}$ and f be a random Simon's function. Given the (d, f)-bijective shuffling oracle $\mathcal{O}_{\mathrm{unif}}^{f,d}$, the problem is to obtain the period s.*

▶ **Definition 16** (The decision d-Bijective Shuffling Simon's Problem). *Let $d \in \mathbb{N}$ and f be either a random Simon's function or a random one-to-one function with equal probability. Given the (d, f)-bijective shuffling oracle $\mathcal{O}_{\mathrm{unif}}^{f,d}$, the promise problem is to distinguish the two cases.*

We show that the d-BSSP can be solved with $d+1$-depth quantum circuits and classical processing.

▶ **Theorem 17.** *The search d-BSSP can be solved with a $(d+1)$-depth quantum circuit composed of $\{H, CNOT\}$ with polynomial-time classical processing. Therefore it can be solved with the $(d+1)$-QC scheme and the $(d+1)$-CQ scheme, and the decision d-BSSP is in $(\mathsf{BPP}^{\mathsf{BQNC}_{d+1}})^{\mathcal{O}} \cap (\mathsf{BQNC}_{d+1}^{\mathsf{BPP}})^{\mathcal{O}}$.*

Proof. The d-BSSP can be solved by the following algorithm which is inspired by the Simon's algorithm and queries to the quantum oracle $d+1$ times.

$$|0\rangle|0\rangle|0\rangle|0\rangle \xrightarrow{H^{\otimes n}} \sum_{x\in\mathbb{Z}_2^n} |x\rangle|0\rangle|0\rangle|0\rangle \xrightarrow{f_0} \sum_{x\in\mathbb{Z}_2^n} |x\rangle|f_0(x)\rangle|0\rangle|0\rangle$$

$$\xrightarrow{f_1} \sum_{x\in\mathbb{Z}_2^n} |x\rangle|f_1(f_0(x))\rangle|0\rangle|0\rangle \xrightarrow{f_2} \cdots$$

$$\xrightarrow{f_{d-1}} \sum_{x\in\mathbb{Z}_2^n} |x\rangle|f_{d-1}(\cdots f_1(f_0(x)))\rangle|0\rangle|0\rangle$$

$$\xrightarrow{f'_d,\zeta,\eta} \sum_{x\in\mathbb{Z}_2^n} |x\rangle|f(x)\rangle|\zeta(f_{d-1}(\cdots f_1(f_0(x))))\rangle|\eta(f_{d-1}(\cdots f_1(f_0(x))))\rangle$$

$$\xrightarrow{Measure} [|x\rangle|\eta(f_{d-1}(\cdots f_1(f_0(x))))\rangle + |x\oplus s\rangle|\eta(f_{d-1}(\cdots f_1(f_0(x\oplus s))))\rangle]|f(x)\rangle|1\rangle$$

$$\xrightarrow{H^{\otimes n+1}} \sum_{j\in\mathbb{Z}_2^n} \left[(-1)^{x\cdot j}(1+(-1)^{s\cdot j})|j\rangle|0\rangle + (-1)^{(x\cdot j)\oplus\eta(f_{d-1}(\cdots f_1(f_0(x))))}(1-(-1)^{s\cdot j})|j\rangle|1\rangle\right]$$

When the measurement result of the last qubit is 0, then $s\cdot j = 0$. Otherwise, $s\cdot j = 1$. Therefore, by sampling $O(n)$ times and solving linear equations, we can obtain the period s with high probability. ◀

4 Analyzing the Bijective Shuffling Oracle and Oneway-to-Hiding (O2H) Lemma

In this section, we define several notations of the bijective shuffling problem and prove Oneway-to-Hiding (O2H) Lemma [6] for our quantum oracle defined in the previous section. First, let us introduce notations of hidden sets.

▶ **Definition 18.** *Let $S_0 = \{0, ..., 2^n - 1\}$. For $j = 1, ..., d$, let $S_j = f_{j-1} \circ \cdots f_0(S_0)$.*

▶ **Definition 19** (The hidden sets **S**, Definition 5.2 in [11]). *Let $\mathcal{F}_b$ be a (d,f)-bijective shuffling. The sequence of hidden sets $\mathbf{S}=(\overline{S}^{(0)},...,\overline{S}^{(d)})$ is defined as follows:*

- *Let $S_j^{(0)}=\mathbb{Z}_2^{(d+2)n}$ for $j=0,...,d$. $\overline{S}^{(0)}:=(S_0^{(0)},...,S_0^{(d)})$.*
- *For $l=1,...,d$, for $j=l,...,d$, we choose $S_j^{(l)}\subseteq S_j^{(l-1)}$ randomly satisfying that $\frac{|S_j^{(l)}|}{|S_j^{(l-1)}|}\leq \frac{1}{2^n}$, $f_{j-1}(S_{j-1}^{(l)})=S_j^{(l)}$, and $S_j\subseteq S_j^{(l)}$. $\overline{S}^{(l)}:=(S_l^{(l)},...,S_d^{(l)})$.*

▶ **Definition 20** ($\mathcal{F}^{(l)}$ and $\hat{\mathcal{F}}^{(l)}$). *For $l=1,...,d-1$, let $f_j^{(l)}$ be f_j on $S_j^{(l-1)}\setminus S_j^{(l)}$ and $\hat{f}_j^{(l)}$ be f_j on $S_j^{(l)}$. Let $f_d'^{(l)}$ be f_d' on $S_d^{(l-1)}\setminus S_d^{(l)}$ and $\hat{f}_d'^{(l)}$ be f_d' on $S_d^{(l)}$. For $l=1,...,d$, let $\zeta^{(l)}$ be ζ on $S_d^{(l-1)}\setminus S_d^{(l)}$ and $\hat{\zeta^{(l)}}$ be ζ on $S_d^{(l)}$. Also for $l=1,...,d$, let $\eta^{(l)}$ be η on $S_d^{(l-1)}\setminus S_d^{(l)}$ and $\hat{\eta}^{(l)}$ be η on $S_d^{(l)}$. Then, we define $\mathcal{F}^{(1)}:=(f_0,f_1^{(1)},...,f_d'^{(1)},\zeta^{(1)},\eta^{(1)})$, $\mathcal{F}^{(d+1)}:=(\hat{f}_d'^{(d)},\hat{\zeta}^{(d)},\hat{\eta}^{(d)})$, and $\mathcal{F}^{(l)}:=(\hat{f}_{l-1}^{(l)},f_l^{(l)},...,f_d'^{(l)},\zeta^{(l)},\eta^{(l)}))$ for $l=2,...,d$. Also, we define $\hat{\mathcal{F}}^{(0)}:=(f_0,...,f_d',\zeta,\eta)$ and $\hat{\mathcal{F}}^{(l)}:=(\hat{f}_l^{(l)},...,\hat{f}_d'^{(l)},\hat{\zeta}^{(l)},\hat{\eta}^{(l)})$ for $l=1,...,d$.*

We stress that for $l=1,...,d$, conditioned on $\mathcal{F}^{(1)},...,\mathcal{F}^{(l)}$, the function $\hat{f}_j^{(l)}$ is still uniformly distributed in $P(S_j^{(l)},S_{j+1}^{(l)})$ for $j=l,...,d-1$. This is because the number of the ways to choose $\hat{\eta}^{(l)}$ is the same for any condition $\mathcal{F}^{(1)},...,\mathcal{F}^{(l)}$ and $\hat{f}_l^{(l)},...,\hat{f}_{d-1}^{(l)}$ in $\mathbf{BSHUF}(d,f)$. In this paper, similarly to [11], we say that a quantum state ρ or a classical string s is uncorrelated to $\mathcal{F}^{(l)}$ if the process which outputs ρ or s will not change the output distribution even if we replace $\mathcal{F}^{(l)}$ by any other mapping.

Following the concept of the hidden set **S**, let us define the shadow. Definition 21 is similar to Definition 5.3 in [11], but, instead of $\perp$ (a symbol represents a constant with no information), we consider the value of the domain and a function ξ which can be recognized as a boolean flag of the shadow. Note that the reason why we also consider the function ξ is to make the quantum oracle unitary.

▶ **Definition 21** (Shadow function). *Let $\mathcal{F}_b:=(f_0,...,f_{d-1},f_d',\zeta,\eta)$ be a (d,f)-bijective shuffling. Fix the hidden sets $\mathbf{S}=(\overline{S}^{(0)},...,\overline{S}^{(d)})$. Let c be a constant bit independent of $\mathcal{F}_b$ and **S**. For $j=l,...,d-1$, let g_j be the function such that if $x\in S_j^{(l)}$, $g_j(x)=x$; otherwise, $g_j(x)=f_j(x)$. Let g_d be the function such that if $x\in S_d^{(l)}$, $g_d(x)=x$; otherwise, $g_d(x)=f_d'(x)$. Let ζ_g be the function such that if $x\in S_d^{(l)}$, $\zeta_g(x)=c$; otherwise, $\zeta_g(x)=\zeta(x)$. Let η_g be the function such that if $x\in S_d^{(l)}$, $\eta_g(x)=c$; otherwise, $\eta_g(x)=\eta(x)$. Finally, for $j=l,...,d$, let ξ_j be the function such that if $x\in S_j^{(l)}$, $\xi_j=c$; otherwise, $\xi_j(x)=c\oplus 1$. The shadow $\mathcal{G}$ of $\mathcal{F}_b$ in $\overline{S}^{(l)}=(S_l^{(l)},...,S_d^{(l)})$ is defined as $\mathcal{G}:=(f_0,...,f_{l-1},g_l,...,g_d,\xi_l,...,\xi_d,\zeta_g,\eta_g)$.*

We stress that the shadow $\mathcal{G}$ does not contain any information of $\mathcal{F}_b$ in $\overline{S}^{(l)}$, which is $\hat{\mathcal{F}}^{(l)}$. Now we define the quantum oracle $\mathcal{G}$ of $\mathcal{F}_b$ in $\overline{S}^{(l)}$ as follows:

$$\mathcal{G}|\phi\rangle := \sum_{\mathbf{X}_0,...,\mathbf{X}_d} c(\mathbf{X}_0,...,\mathbf{X}_d)\Bigg(|0,\mathbf{X}_0\rangle|f_0(\mathbf{X}_0)\rangle\otimes\bigotimes_{i=1}^{l-1}|i,f_i(\mathbf{X}_i)\rangle|0\rangle\otimes \bigotimes_{i=l}^{d-1}|i,g_i(\mathbf{X}_i)\rangle|\xi_i(\mathbf{X}_i)\rangle\otimes|d,g_d(\mathbf{X}_d)\rangle|\zeta_g(\mathbf{X}_d)\rangle|\eta_g(\mathbf{X}_d)\rangle|\xi_d(\mathbf{X}_d)\rangle\Bigg)_{\mathbf{R}_Q,\mathbf{R}_N} \otimes|w(\mathbf{X}_0,...,\mathbf{X}_d)\rangle_{\mathbf{R}_W}.$$

We will introduce the "semi-classical" oracle [6] in our setting.

▶ **Definition 22** ($U^{\mathcal{F}_b\setminus\overline{S}^{(l)}}$, Definition 5.5 in [11]). *Let $\mathcal{F}_b$ be a (d,f)-bijective shuffling. Let $\mathbf{S}=(\overline{S}^{(0)},...,\overline{S}^{(d)})$ be the hidden sets. Let U be a unitary operator acts on qubits of the register $\mathbf{R}$. For $l=1,...,d$, let $U^{\mathcal{F}_b\setminus\overline{S}^{(l)}}:=\mathcal{F}_bU_{\overline{S}^{(l)}}U$ be a unitary operator acts on qubits of registers $(\mathbf{R},\mathbf{I})$ where $\mathbf{I}$ is a one-qubit register and $U_{\overline{S}^{(l)}}$ is defined as follows:*

$$U_{\overline{S}^{(l)}}\left|(l,\mathbf{X}_l),...,(d,\mathbf{X}_d)\right\rangle_{\mathbf{R}}\left|b\right\rangle_{\mathbf{I}}:=\begin{cases}\left|(l,\mathbf{X}_l),...,(d,\mathbf{X}_d)\right\rangle_{\mathbf{R}}\left|b\right\rangle_{\mathbf{I}} & \text{if every } \mathbf{X}_i\cap S_i^{(l)}=\emptyset,\\ \left|(l,\mathbf{X}_l),...,(d,\mathbf{X}_d)\right\rangle_{\mathbf{R}}\left|b\oplus 1\right\rangle_{\mathbf{I}} & \text{otherwise.}\end{cases}$$

▶ **Definition 23** ($\Pr(find\ \overline{S}^{(k)}:U^{\mathcal{F}_b\setminus\overline{S}^{(k)}},\rho)$, Definition 5.6 in [11]). *Let $k,d\in\mathbb{N}$ satisfying $k\leq d$. Let ρ be any input quantum state and U be any unitary operator acts on ρ. We define*

$$\Pr[find\ \overline{S}^{(k)}:U^{\mathcal{F}_b\setminus\overline{S}^{(k)}},\rho]:=\mathbb{E}\left[\mathrm{tr}\left((I_{\mathbf{R}}\otimes(I-\left|0\right\rangle\left\langle 0\right|)_{\mathbf{I}})\circ U^{\mathcal{F}_b\setminus\overline{S}^{(k)}}\circ\rho\otimes\left|0\right\rangle\left\langle 0\right|_{\mathbf{I}}\right)\right],$$

where $\mathbb{E}$ is the expectation value over the random $\mathcal{F}_b$ and $\overline{S}^{(k)}$.

When ρ is a pure state, say $\left|\psi\right\rangle$, we have

$$U^{\mathcal{F}_b\setminus\overline{S}^{(l)}}\left|\psi\right\rangle_{\mathbf{R}}\left|0\right\rangle_{\mathbf{I}}:=\left|\phi_0\right\rangle_{\mathbf{R}}\left|0\right\rangle_{\mathbf{I}}+\left|\phi_1\right\rangle_{\mathbf{R}}\left|1\right\rangle_{\mathbf{I}}$$

and $\Pr[find\ \overline{S}^{(k)}:U^{\mathcal{F}_b\setminus\overline{S}^{(k)}},\left|\psi\right\rangle]=\mathbb{E}[||\left|\phi_1\right\rangle_{\mathbf{R}}||^2]$. Note that, due to the definition of $\mathcal{F}_b$, $\left|\phi_0\right\rangle$ and $\left|\phi_1\right\rangle$ are orthogonal because $\left|\phi_0\right\rangle$ involves no query to $\overline{S}^{(k)}$ but $\left|\phi_1\right\rangle$ does. Thus,

$$\mathcal{F}_bU\left|\psi\right\rangle=\left|\phi_0\right\rangle+\left|\phi_1\right\rangle.$$

Similarly, let us consider

$$\mathcal{G}U_{\overline{S}^{(k)}}U\left|\psi\right\rangle_{\mathbf{R}}\left|0\right\rangle_{\mathbf{I}}:=\left|\phi_0\right\rangle_{\mathbf{R}}\left|0\right\rangle_{\mathbf{I}}+\left|\phi_1^{\perp}\right\rangle_{\mathbf{R}}\left|1\right\rangle_{\mathbf{I}}.$$

Since $\left|\phi_0\right\rangle$ and $\left|\phi_1^{\perp}\right\rangle$ are orthogonal from the definition of $\mathcal{G}$,

$$\mathcal{G}U\left|\psi\right\rangle=\left|\phi_0\right\rangle+\left|\phi_1^{\perp}\right\rangle.$$

Note that $\left|\phi_1\right\rangle$ and $\left|\phi_1^{\perp}\right\rangle$ are orthogonal from the definition of $\mathcal{F}_b$ and $\mathcal{G}$. Then, by the concavity of the mixed state, we can prove the following lemma in a very similar way to Lemma 5.7 in [11].

▶ **Lemma 24** (Oneway-to-hiding(O2H) lemma for the bijective shuffling oracle). *Let $k,d\in\mathbb{N}$ satisfying $k\leq d$. Let $\mathcal{F}_b$ be a (d,f)-bijective shuffling. Let $\mathbf{S}=(\overline{S}^{(0)},...,\overline{S}^{(d)})$ be the hidden sets. Let $\mathcal{G}$ be the shadow of $\mathcal{F}_b$ in $\overline{S}^{(k)}$. Then, for any quantum algorithm $\mathcal{A}$, any unitary operator U, any initial quantum state ρ and any classical string t,*

$$\begin{aligned}|\Pr[\mathcal{A}(\mathcal{F}_bU(\rho))=t]-\Pr[\mathcal{A}(\mathcal{G}U(\rho))=t]| &\leq B(\mathcal{F}_bU(\rho),\mathcal{G}U(\rho))\\ &\leq \sqrt{2\Pr[find\ \overline{S}^{(k)}:U^{\mathcal{F}_b\setminus\overline{S}^{(k)}},\rho]}.\end{aligned}$$

By the union bound, it is also shown that the finding probability of $\overline{S}^{(k)}$ is bounded. The following lemma and its proof are very similar to Lemma 5.8 in [11] since

$$\left(\left|0,\mathbf{X}_0\right\rangle\left|f_0(\mathbf{X}_0)\right\rangle\otimes\bigotimes_{i=1}^{d-1}\left|i,f_i(\mathbf{X}_i)\right\rangle\left|0\right\rangle\otimes\left|d,f_d'(\mathbf{X}_d)\right\rangle\left|\zeta(\mathbf{X}_d)\right\rangle\left|\eta(\mathbf{X}_d)\right\rangle\left|0\right\rangle\right)$$

are orthogonal for different sets of queries $\mathbf{X}_0,...,\mathbf{X}_d$.

▶ **Lemma 25.** *Suppose* $\Pr[x \in S_i^{(k)} | x \in S_i^{(k-1)}] \leq p$ *for* $i = k, ..., d$. *Then for any unitary operator* U *and initial quantum state* ρ, *which are promised to uncorrelated to* $\hat{\mathcal{F}}^{(k-1)}$ *and* $\overline{S}^{(k)}$,

$$\Pr[\textit{find } \overline{S}^{(k)} : U^{\mathcal{F}_b \setminus \overline{S}^{(k)}}, \rho] \leq p \cdot q,$$

where q *is the number of parallel queries* $\mathcal{F}_b U$ *performs, which is* $\sum_{i=0}^{d} |\mathbf{X}_i|$.

5 Proof of Lower Bounds

In this section, we prove the lower bounds of d-BSSP for $(\mathsf{BQNC}_d)^{\mathcal{O}}$, $(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$ and $(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$. The proofs are almost the same as the proof of Theorem 6.1, 7.1 and 8.1 in [11] respectively except applying Lemma 24 and Lemma 25 instead of Lemma 5.7 and Lemma 5.8 in [11].

5.1 Lower bounds for $(\mathsf{BQNC}_d)^{\mathcal{O}}$

First we establish d-BSSP is intractable for any QNC_d circuit. By applying the Oneway-to-Hiding lemma inductively, it can be shown that $U_{d+1}\mathcal{F}_b U_d \cdots \mathcal{F}_b U_1$ is indistinguishable from $U_{d+1}\mathcal{G} U_d \cdots \mathcal{G} U_1$. The same argument as in [11] gives the following result.

▶ **Theorem 26.** *Let* $n, d \in \mathbb{N}$. *Let* $\mathcal{A}$ *be any* d*-depth quantum circuit and* ρ *be any initial state. Let* f *be a random Simon's function from* $\mathbb{Z}_2^n$ *to* $\mathbb{Z}_2^n$ *with period* s *and* $\mathcal{F}_b$ *be the* (d, f)*-bijective shuffling sampled from* $\mathcal{O}_{\mathrm{unif}}^{f,d}$. *Then*

$$\Pr[\mathcal{A}^{\mathcal{F}_b}(\rho) = s] \leq d \cdot \sqrt{\frac{\mathsf{poly(n)}}{2^n}} + \frac{1}{2^n}.$$

▶ **Corollary 27.** *The search* d-BSSP *can be solved with at most negligible probability by any* $(\mathsf{QNC}_d)^{\mathcal{O}}$ *and the decision* d-BSSP *is not in* $(\mathsf{BQNC}_d)^{\mathcal{O}}$

5.2 Lower bounds for $(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$

Next, we establish the search d-BSSP is intractable by any d-QC scheme. Since each classical algorithms interspersed quantum layers can find polynomial-size number of points of each domains, a new procedure to define the hidden sets is needed. The same argument as in [11] gives the following result.

▶ **Theorem 28.** *Let* $n, d \in \mathbb{N}$. *Let* $\mathcal{A}$ *be any* d*-QC scheme and* ρ *be any initial state. Let* f *be a random Simon's function from* $\mathbb{Z}_2^n$ *to* $\mathbb{Z}_2^n$ *with period* s *and* $\mathcal{F}_b$ *be the* (d, f)*-bijective shuffling sampled from* $\mathcal{O}_{\mathrm{unif}}^{f,d}$. *Then*

$$\Pr[\mathcal{A}^{\mathcal{F}_b}(\rho) = s] \leq d \cdot \sqrt{\frac{\mathsf{poly(n)}}{2^n}}.$$

▶ **Corollary 29.** *The search* d-BSSP *can be solved with at most negligible probability by any* d*-QC scheme and the decision* d-BSSP *is not in* $(\mathsf{BQNC}_d^{\mathsf{BPP}})^{\mathcal{O}}$.

5.3 Lower bounds for $(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$

Finally, we establish the search d-BSSP is hard for any d-CQ scheme. The main difficulty lies in that conditioned on measurement results of quantum circuits, the distribution of the permutations might be not uniform enough to prove the hardness in a similar way. Chia et al. resolved the problem by approximating it by a convex combination of "almost" uniform shufflings. The same argument as in [11] gives the following result. [2]

▶ **Theorem 30.** *Let $n, d \in \mathbb{N}$. Let $\mathcal{A}$ be any d-CQ scheme. Let f be a random Simon's function from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$ with period s and $\mathcal{F}_b$ be the (d, f)-bijective shuffling sampled from $\mathcal{O}_{\mathrm{unif}}^{f,d}$. Then*

$$\Pr[\mathcal{A}^{\mathcal{F}_b}() = s] \leq d \cdot \sqrt{\frac{\mathsf{poly}(n)}{2^n}}.$$

▶ **Corollary 31.** *The search d-BSSP can be solved with at most negligible probability by any d-CQ scheme and the decision d-BSSP is not in $(\mathsf{BPP}^{\mathsf{BQNC}_d})^{\mathcal{O}}$.*

References

1 URL: https://en.wikipedia.org/wiki/List_of_quantum_processors.

2 Scott Aaronson. Quantum lower bound for the collision problem. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC 2002)*, pages 635–642, 2002. doi:10.1145/509907.509999.

3 Scott Aaronson. Ten semi-grand challenges for quantum computing theory, 2005. URL: https://www.scottaaronson.com/writings/qchallenge.html.

4 Scott Aaronson. BQP and the polynomial hierarchy. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 141–150, 2010. doi:10.1145/1806689.1806711.

5 Scott Aaronson. Projects aplenty, 2011. URL: https://scottaaronson.blog/?p=663.

6 Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In *Advances in Cryptology – CRYPTO 2019*, volume 11693 of *Lecture Notes in Computer Science*, pages 269–295. Springer, 2019. doi:10.1007/978-3-030-26951-7_10.

7 Atul Singh Arora, Alexandru Gheorghiu, and Uttam Singh. Oracle separations of hybrid quantum-classical circuits. *arXiv*, 2022. doi:10.48550/arXiv.2201.01904.

8 Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. doi:10.1038/s41586-019-1666-5.

9 Marco Cerezo et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021. doi:10.1038/s42254-021-00348-9.

10 Nai-Hui Chia. Personal communication, 2022.

11 Nai-Hui Chia, Kai-Min Chung, and Ching-Yi Lai. On the need for large quantum depth. In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC 2020)*, pages 902–915, 2020. doi:10.1145/3357713.3384291.

12 Nai-Hui Chia and Shih-Han Hung. Classical verification of quantum depth. *arXiv*, 2022. doi:10.48550/arXiv.2205.04656.

13 Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003)*, pages 59–68, 2003. doi:10.1145/780542.780552.

[2] In fact, there exists a small issue in the current proof of Theorem 8.1 in [11]. However, it has already been fixed and the fixed proof can be applied to the setting of this work [10].

14 Richard Cleve and John Watrous. Fast parallel circuits for the quantum Fourier transform. In *Proceedings of 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 526–536, 2000. doi:10.1109/SFCS.2000.892140.

15 Matthew Coudron and Sanketh Menda. Computations with greater quantum depth are strictly more powerful (relative to an oracle). In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC 2020)*, pages 889–901, 2020. doi:10.1145/3357713.3384269.

16 Bill Fefferman and Shelby Kimmel. Quantum vs. classical proofs and subset verification. In *Proceedings of 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, pages 22:1–22:23, 2018. doi:10.4230/LIPIcs.MFCS.2018.22.

17 Stephen A Fenner and Yong Zhang. A note on the classical lower bound for a quantum walk algorithm. *arXiv*, 2003. doi:10.48550/arXiv.quant-ph/0312230.

18 Richard Jozsa. An introduction to measurement based quantum computation. *arXiv*, 2005. doi:10.48550/arXiv.quant-ph/0508124.

19 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

20 Daniel R. Simon. On the power of quantum computation. In *Proceedings of 35th IEEE Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pages 116–123, 1994. doi:10.1109/SFCS.1994.365701.

21 Dominique Unruh. Revocable quantum timed-release encryption. *Journal of the ACM*, 62(6):1–76, 2015. doi:10.1145/2817206.

22 Han-Sen Zhong et al. Quantum computational advantage using photons. *Science*, 370:1460–1463, 2020. doi:10.1126/science.abe8770.

Approximating the Minimum Logarithmic Arrangement Problem

Julián Mestre
Meta Platforms Inc., USA
School of Computer Science, The University of Sydney, Australia

Sergey Pupyrev
Meta Platforms Inc., USA

Abstract

We study a graph reordering problem motivated by compressing massive graphs such as social networks and inverted indexes. Given a graph, $G = (V, E)$, the *Minimum Logarithmic Arrangement* problem is to find a permutation, π, of the vertices that minimizes

$$\sum_{(u,v)\in E} (1 + \lfloor \lg |\pi(u) - \pi(v)| \rfloor) .$$

This objective has been shown to be a good measure of how many bits are needed to encode the graph if the adjacency list of each vertex is encoded using relative positions of two consecutive neighbors under the π order in the list rather than using absolute indices or node identifiers, which requires at least $\lg n$ bits per edge.

We show the first non-trivial approximation factor for this problem by giving a polynomial time $\mathcal{O}(\log k)$-approximation algorithm for graphs with treewidth k.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases approximation algorithms, graph compression

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.7

Acknowledgements We thank Okke Schrijvers, Karthik Abinav Sankararaman and Riccardo Colini Baldeschi for fruitful discussions of the problem.

1 Introduction

We study theoretical aspects of a graph reordering problem that has applications to compressing social networks and inverted indexes. The formal model of the problem has been suggested by Chierichetti et al. [6], who proposed a simple heuristic for reordering web-scale graphs. Later Dhulipala et al. [11] extended the model and described a practical approach for graph reordering based on recursive bisection. The algorithm of [11] is widely used in practice producing the most "compression-friendly" vertex orders for a large variety of real-world datasets and is considered the state-of-the-art in the field [31].

A *linear layout* (an order or an arrangement) of a graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges is a bijection $\pi : V \to \{1, \ldots, n\}$. Most graph encoding schemes are based on performing a *delta-encoding* of the adjacency lists using a linear layout. The basic idea is to sort each adjacency list according to the layout π, store the index of the first neighbor in the list, followed by the the gaps between two consecutive neighbors using a variable length encoding. As such, it is desirable that the neighborhood of each vertex is laid out close together, since that translates into smaller gaps and higher compression rates. This motivates two problems that we define next.

The *minimum linear arrangement* (MLA) problem is to find a layout π so that

$$\mathrm{LA}_\pi(G) := \sum_{(u,v)\in E} |\pi(u) - \pi(v)|$$

is minimized. This is a classical NP-hard problem [23], even when restricted to certain graph classes. The problem is APX-hard under Unique Games Conjecture [10] but admits an $\mathcal{O}(\sqrt{\log n}\log\log n)$ approximation [5, 19]. Notice that the objective measures the total length of the gaps across all edges.

A closely related problem is *minimum logarithmic arrangement* (MLogA) in which the goal is to minimize

$$\mathrm{LGA}_\pi(G) := \sum_{(u,v)\in E} (1 + \lfloor \lg |\pi(u) - \pi(v)| \rfloor),$$

where $\lg x$ denotes the logarithm base 2 of x. Note that for an integer x, $1 + \lfloor \lg x \rfloor$ is the number of bits needed to represent x. We write $\mathrm{LGA}(G) = \min_\pi \mathrm{LGA}_\pi(G)$, where the minimum is taken over all permutations of vertices V. Seen from this perspective, $\mathrm{LGA}(G)$ is a measure of the compressed size of G. It is worth noting that in practice, the size of an encoded integer and the total size of a graph depends on the utilized encoding scheme; we refer to [2] for a survey of modern graph compression techniques.

1.1 Our Contributions

In this paper we study MLogA from a theoretical perspective. First, in Section 2, we investigate basic properties of the problem: analyze the performance of two natural heuristics, provide explicit optimal and near-optimal layouts for several graph classes, and describe a lower bound for the LGA cost of a graph.

Section 3 describes our main result, an $\mathcal{O}(\log k)$-approximation for graphs with treewidth k. It is worth noting that the optimal ordering has cost at least m and that *every* ordering has cost $\mathcal{O}(m \log n)$. Therefore outputting an arbitrary order is, technically speaking, a logarithmic approximation for MLogA. The challenge is to design an approximation algorithm with approximation factor $o(\log n)$. Our result is the first such approximation for the natural and broad class of graphs with low treewidth. The algorithm works by recursively splitting the input graph using small balanced separators. It is worth noting that our algorithm can be implemented to run in polynomial time regardless of the value of k. While the algorithm is fairly straightforward, its analysis is highly non-trivial.

Regarding the applicability of our approach, we point to the recent work of Maniu *et al.* [32] who experimentally estimated the treewidth of graphs arising from a variety of domains. They found that real-world instances usually have treewidth values that are very small compared the number of vertices in the graph. Therefore, we can reasonably expect our $\mathcal{O}(\log k)$ approximation to yield much better results in real-world instances over the trivial $\mathcal{O}(\log n)$ approximation.

We conclude the paper in Section 4 with some interesting open problems.

1.2 Related Work

Not many results on MLogA are known. Chierichetti et al. [6] show that the problem is NP-hard on multi-graphs and present lower bounds on expander-like graphs. More specifically, they show that if a graph G has constant conductance then the cost of MLogA is $\Omega(m \log n)$, and that if G has constant node or edge expansion then the cost of MLogA is $\Omega(n \log n)$.

The *minimum logarithmic gap arrangement* (MLogGapA) problem [6, 11] is a related objective that captures more faithfully the information-theoretic space needed to represent a graph using a delta-encoding representation for its adjacency lists. For a vertex $v \in V$ of degree k and an order π, consider the neighbors $out(v) = (v_1, \ldots, v_k)$ of v such that

$\pi(v_1) < \cdots < \pi(v_k)$. Then the cost compressing the list $out(v)$ under π is related to $f_\pi(v, out(v)) = \sum_{i=1}^{k-1} \log|\pi(v_{i+1}) - \pi(v_i)|$. MLogGapA consists in finding an order π, which minimizes

$$\sum_{v \in V} f_\pi(v, out(v)).$$

Similarly to MLogA, the MLogGapA is known to be NP-hard [11]. Furthermore, Dhulipala et al. [11] experimentally verify that the cost of MLogGapA accurately predicts the compressed size of real-world instances for various modern encoding schemes.

For some applications, such as index compression, it is convenient to study a generalization of MLogA and MLogGapA by considering a bipartite graph with *query* and *data* vertices. To this end, let $G = (\mathcal{Q} \cup \mathcal{D}, E)$ be an undirected unweighted bipartite graph with disjoint sets of vertices $\mathcal{Q}$ and $\mathcal{D}$. The goal is to find a permutation, π, of data vertices, $\mathcal{D}$, so that the following objective is minimized:

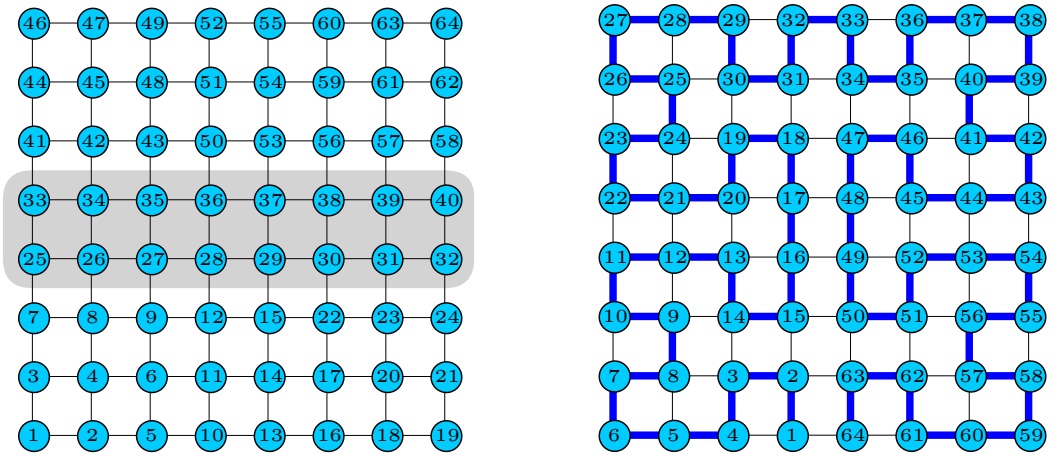
$$\sum_{q \in \mathcal{Q}} \sum_{i=1}^{\deg_q - 1} \log(\pi(u_{i+1}) - \pi(u_i)),$$

where $\deg_q$ is the degree of query vertex $q \in \mathcal{Q}$, and q's neighbors are $\{u_1, \ldots, u_{deg_q}\}$ with $\pi(u_1) < \cdots < \pi(u_{deg_q})$. The optimization problem is called *bipartite minimum logarithmic arrangement* (BiMLogA). Notice that BiMLogA is different from MLogGapA in that the latter does not differentiate between data and query vertices. It is easy to see that the new problem generalizes both MLogA and MLogGapA: to model MLogA, add a query vertex for every edge of the input graph; to model MLogGapA, add a query for every vertex of the input graph.

While according to Chierichetti et al. [6] and Dhulipala et al. [11] MLogGapA and BiMLogA are arguably more relevant for compression than MLogA, we find that the latter problem interesting on its own from a theoretical point of view, and we regard our main contribution as a first toward obtaining approximation algorithms for these more general variants.

Also closely related to our objective is the minimum linear arrangement problem, which has been studied under various names [12], such as optimal linear ordering, minimum-1-sum, or the edge sum problem. MLA was originally proposed in [26]. It was proven to be strongly NP-hard [22] and this was later shown to hold even for bipartite graphs [16] and interval graphs [9]. For general graphs, the fastest known exact algorithm is based on dynamic programming and runs in $\mathcal{O}(2^n \cdot m)$ time [30]. The best approximation factor known for general graphs is $\mathcal{O}(\sqrt{\log n} \log\log n)$ [5, 19]; however, better approximations are known for special graph classes such as interval graphs [9], planar graphs [35], and series-parallel graphs [15]. On the positive side, MLA is known to be solvable in polynomial time on trees [1, 8, 24]. Furthermore, for some restricted classes of graphs, optimal layouts are known explicitly [7, 26, 28].

There is vast literature on the problem of computing an ordering of a graph vertex set to minimize or maximize a given objective function. Here we only mention a few notable examples. The *minimum bandwidth problem* [13, 17, 20, 25, 37] is to find an ordering minimizing the maximum distance between any two vertices connected with an edge; that is, $\min_\pi \max|\pi(u) - \pi(v)|$. Finding a tree (path) decomposition with minimum treewidth (pathwidth) can be cast as the problem of finding a elimination order of the vertices [29]. Finally, we mention the *traveling salesman problem* [27] and its many variants [3, 33, 34], which have inspired ground breaking algorithmic research for over five decades.



(a) MLA. **(b)** MLogA.

Figure 1 Layouts of the 8×8 grid graph optimizing for (a) MLA and (b) MLogA. The layout for MLA is constructed by an algorithm of Fishburn et al. [21] and contains $\Omega(h)$ consecutively numbered rows (shaded). The layout for MLogA is constructed following a space-filling curve as described in Lemma 7.

2 Preliminaries

In this section we first discuss natural heuristics for MLogA and their (worst-case) approximation factors. Then we derive optimal arrangements for several graph classes.

2.1 Heuristics

Greedy

Arguably the easiest approach for MLogA is a greedy one. Start with a vertex, and iteratively add the next vertex that yields the lowest increase of the objective. There are several greedy criteria that we could use.

The simplest version of this approach that does not constrain in any way how we pick our vertices does not yield anything useful even in very simple instances: If the input is a path, the algorithm might pick every other vertex along the path and then the remaining vertices for a total cost of $\Omega(n \log n)$, whereas an optimal solution has cost $n - 1$.

One can refine the greedy criterion by asking that a newly added vertex is connected to one of the vertices already processed, and subject to this that the increase of the objective is minimized. Unfortunately, this also fails: If the input is a $2 \times n$ grid, the algorithm might pick the upper path of the grid followed by the lower path (in opposite direction) for a total cost of $\Omega(n \log n)$ whereas an optimal solution, which interleaves nodes from the top and bottom paths, has cost $\mathcal{O}(n)$.

Minimum Linear Arrangement

It is tempting to apply an algorithm designed for MLA to solve the related objective of MLogA, given that MLA admit an $o(\log n)$-approximation. Here we show that such an approach may result in an $\Omega(\log n)$ approximation for MLogA even if we have an exact algorithm for MLA. Consider the *square grid* graph; that is, the graph whose vertices

correspond to the points in the plane with integer coordinates in the range $1, \ldots, h$ and two vertices connected by an edge whenever the corresponding points are at distance 1. The $h \times h$ grid is denoted by $G_{h,h}$ and contains $n = h^2$ vertices and $m = 2h(h-1)$ edges.

Fishburn et al. [21] describe the optimal arrangement, π, of the square grid; it contains t consecutively numbered rows with $t/h \to 1 - 1/\sqrt{2}$ as $h \to \infty$; see Figure 1a. The corresponding vertical edges between the rows in the grid have length h and there are $t \times h$ such edges. Summing up the contribution of the edges for MLogA, we get $\mathrm{LGA}_\pi \geq t \times h \times \lg h = \Omega(h^2 \times \log h) = \Omega(m \times \log n)$. However, as we show in Lemma 7, there is an $\mathcal{O}(m)$ solution for MLogA; see Figure 1b.

2.2 Lower bounds

Before proving a lower bound for the objective of MLogA, we show a simple fact about sums of logarithmic values.

▶ **Lemma 1.** *For any integer $\ell \geq 1$ we have*

$$(\ell - 1) \cdot \lg(\ell + 1) < \sum_{i=1}^{\ell} (1 + \lfloor \lg i \rfloor) < (\ell + 1) \cdot \lg(\ell + 1).$$

Proof. We can use integrals to prove the upper bound:

$$\sum_{i=1}^{\ell} (1 + \lfloor \lg i \rfloor) \leq \int_1^{\ell+1} 1 + \lg x \, \mathrm{d}x \leq (\ell + 1) \cdot \lg(\ell + 1).$$

And the lower bound:

$$\sum_{i=1}^{\ell} (1 + \lfloor \lg i \rfloor) \geq \int_1^{\ell+1} \lg x \, \mathrm{d}x \geq (\ell - 1) \cdot \lg(\ell + 1). \qquad \blacktriangleleft$$

It is clear that $\mathrm{LGA}(G) \geq m$ for every graph G, since the contribution of each edge to the objective is at least 1. The next lemma improves upon this trivial bound for dense graphs.

▶ **Lemma 2.** *Let $G = (V, E)$ be a graph with n vertices and m edges, then*

$$\mathrm{LGA}(G) \geq (m - n) \cdot \lg \frac{m}{n}$$

Proof. Consider a vertex, $v \in V$, and all incident edges. The optimal layout of the star subgraph is achieved when v is placed in the middle of the order and the neighbors occupy consecutive intervals to the left and to the right of v. Thus the edges incident on v contribute to the objective at least

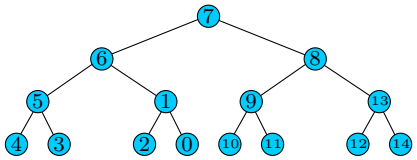
$$\sum_{i=1}^{\lfloor \deg(u)/2 \rfloor} (1 + \lfloor \lg i \rfloor) + \sum_{i=1}^{\lceil \deg(u)/2 \rceil} (1 + \lfloor \lg i \rfloor) \geq (\deg(u) - 2) \cdot \lg \frac{\deg(u)}{2}$$

where the inequality follows from applying Lemma 1 to each sum.

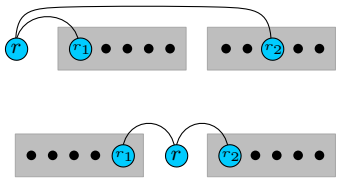
Summing over all vertices and observing that every edge is counted twice, gives a global lower bound of

$$\mathrm{LGA}(G) \geq \sum_{u \in V} \frac{\deg(u) - 2}{2} \cdot \lg \frac{\deg(u)}{2} \geq (m - n) \cdot \lg \frac{m}{n}.$$

where the last inequality follows from Jensen's inequality and the fact $f(x) = (x - 2) \cdot \lg x$ is a concave function, which means that the sum is minimized when all n terms are equal. $\blacktriangleleft$



(a) A complete binary tree with 4 levels numbered according to Lemma 6.



(b) Two ways of embedding a binary tree, *side-based* (top) and *center-based* (bottom).

Figure 2 Embedding a complete binary tree with LGA $\leq \frac{5}{3}n$.

2.3 Specific Graph Classes

▶ **Lemma 3.** *Let K_n denote the complete graph with n vertices. Then*

$$\mathrm{LGA}(K_n) \leq \frac{n^2 \lg n}{2}.$$

Proof. The bound follows the observation that all layouts of a complete graph are equivalent, and applying Lemma 1 to each node and accounting for double counting:

$$\mathrm{LGA}(K_n) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n-1}(1 + \lfloor \lg j \rfloor) \leq \frac{n^2 \lg n}{2}.$$ ◀

▶ **Lemma 4.** *Let P_n and C_n denote the path and cycle with n vertices, respectively. Then*

$$\mathrm{LGA}(P_n) = n - 1 = m \qquad \textit{and} \qquad \mathrm{LGA}(C_n) = n + \lfloor \lg(n-1) \rfloor = m + \lfloor \lg(n-1) \rfloor.$$

Proof. The bound for the path is trivial. For the cycle, denote the lengths of the edges of C_n by $e_1, \ldots, e_n$. Observe that for every ordering of C_n, there exist two edge-dis int paths connecting the first and the last vertices in the order. Hence, $e_1 + e_2 + \cdots + e_n \geq 2n - 2$ and $e_i \geq 1$. Using an exchange argument, it is straightforward to show that given those constraints $\sum_{i=1}^{n}(1 + \lfloor \lg e_i \rfloor)$ is minimized when $e_1 = \cdots = e_{n-1} = 1$ and $e_n = n - 1$, which yields the claim. ◀

▶ **Lemma 5.** *Let $K_{1,\ell}$ denote a star with ℓ leaves. Then*

$$(\ell - 2) \cdot (1 + \lg \frac{\ell}{2}) \leq \mathrm{LGA}(K_{1,\ell}) \leq (\ell + 2) \cdot \lg \frac{\ell + 1}{2}.$$

Proof. The equality follows from the observation that the optimal layout is achieved when the central vertex of the star is placed in the middle of the order. The inequalities are the result of applying Lemma 1 to this sum. ◀

▶ **Lemma 6.** *Let T_n denote the k-level complete binary tree with $n = 2^k - 1$ vertices. Then*

$$\mathrm{LGA}(T_n) \leq \left\lceil \tfrac{5}{3}(2^k - 1) \right\rceil - k - 1 \leq \tfrac{5}{3}n.$$

Proof. Consider a complete binary tree, T_n, with k levels such that $n = 2^k - 1$. Let r be the root of the tree connected to two copies of a complete tree with $k - 1$ levels; see Figure 2a. In order to prove the claim, we consider two ways of embedding T_n: a *side-based* layout in which r is the rightmost (or leftmost) in the order, and a *center-based* layout in which r is positioned between the two copies of the subtrees, T_{n-1}. See Figure 2b for an illustration and observe that the vertices of each of the subtrees do not overlap in the resulting order.

Define the cost of embedding a complete binary tree with k levels using the side-based and center-based approaches by $S(k)$ and $C(k)$, respectively. It follows directly from the construction that

$$\begin{aligned} C(k) &= 2 + 2S(k-1) && \text{and} \\ S(k) &= 2 + \lfloor \lg(2^{k-1} + 2^{k-2} - 1) \rfloor + S(k-1) + C(k-1) \\ &= 1 + k + S(k-1) + C(k-1) && \text{for } k \geq 2 \text{ and} \\ C(1) &= S(1) = 0. \end{aligned}$$

We claim that $C(k) = \lceil \frac{5}{3}(2^k - 1) \rceil - k - 1$ and $S(k) = C(k) + \lfloor \frac{k}{2} \rfloor$. By induction, the two bounds clearly hold for $k = 1$. For $k \geq 2$, we have

$$C(k) = 2 + 2S(k-1) = 2 + 2\Big(C(k-1) + \lfloor \tfrac{k-1}{2} \rfloor\Big) = 2 + 2\Big(\lceil \tfrac{5}{3}(2^{k-1} - 1) \rceil - k + \lfloor \tfrac{k-1}{2} \rfloor\Big).$$

Observe that for even k, we have $2^k \bmod 3 = 1$, while for odd k, it holds $2^k \bmod 3 = 2$. Thus, when $k = 2t$ is even, $2\lceil \frac{5}{3}(2^{k-1} - 1) \rceil = \lceil \frac{5}{3}(2^k - 1) \rceil - 1$; therefore,

$$C(k) = 2 + \lceil \tfrac{5}{3}(2^k - 1) \rceil - 1 - 4t + 2\lfloor \tfrac{2t-1}{2} \rfloor = \lceil \tfrac{5}{3}(2^k - 1) \rceil - 2t - 1.$$

Similarly, when $k = 2t + 1$, we have $2\lceil \frac{5}{3}(2^{k-1} - 1) \rceil = \lceil \frac{5}{3}(2^k - 1) \rceil - 2$; therefore,

$$C(k) = 2 + \lceil \tfrac{5}{3}(2^k - 1) \rceil - 2 - 2(2t+1) + 2\lfloor \tfrac{2t}{2} \rfloor = \lceil \tfrac{5}{3}(2^k - 1) \rceil - (2t+1) - 1.$$

The inductive step for $S(k)$ is verified analogously.

Finally, observe that $\mathrm{LGA}(T_n) \leq \min(C(k), S(k))$, which proves the desired bound. ◀

We conjecture that the bound given by Lemma 6 is optimal for complete binary trees; it has been verified computationally for trees with up to 15 vertices.

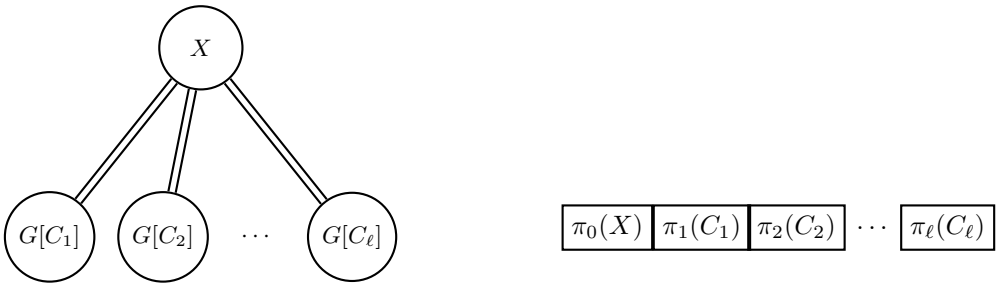
Next we explore MLogA on the $h \times h$ grid graph, $G_{h,h}$, and suggest using a space-filling curve to layout the vertices. A space filling curve is a continuous mapping from the unit interval $[0, 1]$ to the unit square $[0, 1]^2$. The idea is to overlay the grid over the unit square and then use the curve order to sort the vertices of the grid. We show that the layout obtained from the well-known Hilbert curve [38] yields a constant factor approximation for MLogA.

▶ **Lemma 7.** *Let $G_{h,h}$ denote the $h \times h$ grid graph with h being a power of two. Then*

$$\mathrm{LGA}(G_{h,h}) \leq 4h^2 = \mathcal{O}(m).$$

Proof. At a very high level, the Hilbert curve orders the points in the unit square by recursively dividing it into four smaller squares, visiting each of smaller square in turn and concatenating the partial traversals. This construction yields a hierarchical decomposition of the grid. At the top level of the hierarchy we have a single square holding all h^2 points. One level down, at level 1, we have 4 smaller squares of $h/2 \times h/2$ each holding $h^2/4$ points. In general, level i has 4^i squares each holding $h^2/4^i$ points; see Figure 1b.

We say that an edge (u, v) is *cut at level i* if u and v belong to the same square at level i but different squares at level $i + 1$. Notice that this means that the distance between u and v is no larger than the size of the the squares at level i, namely, $|\pi(u) - \pi(v)| \leq h^2/4^i$. Furthermore, notice that there are $2h$ edges cut at level 0, $4h$ edges at level 1, and in general, $2^{i+1}h$ edges cut at level i.



(a) A balanced separator X and the connected components of $G[V \setminus X]$.

(b) Partial layouts of each component are sequenced in an arbitrary order.

Figure 3 Algorithm BALANCED finds a small balanced separator X, recursively computes a partial layout π_i for each connected component C_i of $G[V \setminus X]$, and builds a full layout of G by concatenating these partial layouts and an arbitrary sequencing of X.

Therefore, the objective value of the layout is upper bounded by

$$\begin{aligned}
\mathrm{LGA}(G_{h,h}) &\leq \sum_{i=0}^{\lg h-1} 2^{i+1} h \cdot (1 + \lfloor \lg \frac{h^2}{4^i} \rfloor) \\
&\leq 2h^2 + 2h \sum_{i=0}^{\lg h-1} 2^i \lg \frac{h^2}{4^i} \\
&\leq 2h^2 + 2h \int_{i=1}^{\lg h} 2^x \lg \frac{h^2}{4^x} \mathrm{d}x \\
&\leq 2h^2 + \frac{2}{\lg 2} h^2 \\
&\leq 4h^2
\end{aligned}$$

Finally, we note that the grid contains $2h^2 - 2h$ edges, so the layout is at least 2-approximate. ◀

3 Balanced Separator

In this section we explore the performance of a divide-and-conquer algorithm based on balanced separators. Recall that a set of vertices $X \subseteq V$ is a *balanced vertex separator* for $G = (V, E)$ if every connected component of $G[V \setminus X]$ has at most $\left\lceil \frac{|V \setminus X|}{2} \right\rceil$ vertices. The *separation number* of G is the minimum integer k such that every subgraph of G has a balanced separator of order at most k. It is known that the separation number of a graph is linearly related to its treewidth [14, 36].

Our algorithm, which we call BALANCED, recursively finds a small balanced separator X, arbitrarily sequences X to get a partial layout $\pi_0(X)$, identifies the connected components $C_1, C_2, \ldots, C_\ell$ of $G[V \setminus X]$, recursively finds a layout $\pi_i(C_i)$ for each subgraph $G[C_i]$, and then concatenates all these layouts in arbitrary order, say $\langle \pi_0(X), \pi_1(C_1), \pi_2(C_2), \ldots, \pi_\ell(C_\ell) \rangle$.

For each problem $G' = (V', E')$ we find along the way, we assign a level value to the problem based on its size; more precisely, we say that the subproblem is in level i if $\frac{n}{2^i} \leq |V'| < \frac{n}{2^{i-1}}$. Note that because of the balanced nature of the separators, a problem can only generate subproblems in lower levels. Thus, it follows that the collection of subproblems at level i forms a partition of a vertex subset of the input instance. Furthermore, since each

subproblem at level i has cardinality at least $\frac{n}{2^i}$, it follows that we can have at most 2^i subproblems at level i. We extend the level assignment of subproblems to vertices as follows. If u belongs to the separator chosen for a subproblem at level i, then we say that u belongs to level i. Notice that once a vertex is chosen into a separator, it never again shows up in subproblems.

Now consider a separator X of a subproblem $G' = (V', E')$. We assign every edge in E' incident on X towards an endpoint in X (edges in $E'[X]$ can pick an endpoint arbitrarily). For each $u \in X$, let μ_u be the number of edges assigned to u in this way; note that $\mu_u > 0$ since every u must be connected to $V' \setminus X$, otherwise $X - u$ is also a balanced separator. Lastly, let L_i denote the set of nodes in level i and $\ell - 1$ be the deepest non-empty level. Since every edge in the input instance is assigned in this way, it follows that $\sum_u \mu_u = |E|$ and that $1 \leq \mu_u \leq n$.

On one hand, the cost of the layout is upper bounded by

$$\mathrm{UB}(\mu) = \sum_{i=0}^{\ell-1} \sum_{u \in L_i} \mu_u \cdot \left(1 + \lg \frac{n}{2^i}\right)$$

On the other hand, every node u with μ_u edges assigned to it needs at least as many bits to encode the edges as a star with μ_u leaves does. Using Lemma 5 we can infer that we need at least $(\mu_u - 2) \cdot \lg\left(1 + \frac{\mu_u}{2}\right)$ bits. Together with the fact that we always need at least μ_u bits, we get that $\frac{\mu_u}{4} \cdot (1 + \lg \mu_u)$ bits are always needed. Therefore, by ignoring the constant factor, we can use the following as the lower bound:

$$\mathrm{LB}(\mu) = \sum_{i=0}^{\ell-1} \sum_{u \in L_i} \mu_u \cdot (1 + \lg \mu_u)\,.$$

Define $\rho(\mu) = \frac{\mathrm{UB}(\mu)}{\mathrm{LB}(\mu)}$. The approximation ratio of the algorithm is bounded up to constant factors by $\rho(\mu)$. The rest of this section is devoted to showing that if the graph has small balanced separators, this ratio is small.

Consider the auxiliary problem of finding an assignment μ and levels L_i that maximizes $\rho(\mu)$ subject to the following constraints:

- $\sum_u \mu_u \geq n$, and $1 \leq \mu_u \leq n$ for all u,
- $|L_i| \leq k2^i$, where k is an absolute upper bound on the cardinality of the balanced separators we find along the way.

Strictly speaking the first constraint should be $\sum_u \mu_u = m$, but as we shall soon see, the worst bound of $\rho(\mu)$ occurs when $m = n$. The second constraint follows from the fact that there are at most 2^i sub-problems at level i and that each of these has a separator of size at most k.

▶ **Lemma 8.** *For any assignment μ and levels L_i subject to the above constraints, $\rho(\mu)$ is upper bounded by $\mathcal{O}(\log k)$.*

Proof. First we identify further constraints that we can assume without loss of generality:

- $\sum_u \mu_u = n$. Otherwise, we can multiply μ by $\gamma = \sum_u \mu/n$, which cause $\mathrm{UB}(\mu)$ to scale down by a factor of γ, while $LB(\mu)$ decreases by a factor strictly greater than γ (due to its super-linear terms).
- $\forall u, v \in L_i$, we have $\mu_u = \mu_v$. Otherwise, average their values, which does not change $\mathrm{UB}(\mu)$ but decreases $\mathrm{LB}(\mu)$.
- $\forall u \in L_i, v \in L_j$, if $i < j$ then $\mu_u \geq \mu_v$. Otherwise, we can swap their values, increasing $\mathrm{UB}(\mu)$ without changing $\mathrm{LB}(\mu)$.

- $|L_i| = 2^i$ for all $i < \ell - 1$. Otherwise, if a level is not full we can promote a node for a lower level, which increases $\mathrm{UB}(\mu)$ but does not change $\mathrm{LB}(\mu)$.

In every case, the change increases $\rho(\mu)$, so we can assume all these properties without loss of generality.

We also assume that $|L_{\ell-1}| = 2^{\ell-1}$. Otherwise, if the level is not full we get rid of it altogether and scale up other values to add up to n. This can decrease the value of the solution a single time by a constant multiplicative amount; that is, at most 2.

Furthermore, we can assume that if we have two nodes $u \in L_i$ and $v \in L_{i+1}$ in consecutive layers and we increase/decrease μ_u and decrease/increase μ_v the change should not improve the ratio $\rho(\mu)$, which we denote for brevity with ρ from now on. Out of this requirement we get the following property.

▷ Claim 9. The worst ratio $\rho(\mu) = \frac{\mathrm{UB}(\mu)}{\mathrm{LB}(\mu)}$ is attained when for any two nodes $u \in L_i$ and $v \in L_{i+1}$ in consecutive layers we have

$$\frac{\mu_u}{\mu_v} = 2^{\frac{1}{\rho(\mu)}}.$$

Proof. Consider the operation of deviating slightly from the give vector μ to another vector increasing μ_u by a small δ amount while decreasing μ_v by the same amount. Let us denote with $\mu|\delta$ this new vector. And let $f(\delta) = \frac{\mathrm{UB}(\mu|\delta)}{\mathrm{LB}(\mu|\delta)}$

Assuming that μ is the vector maximizing the ratio, we expect that $f'(0) = 0$; for otherwise, we can deviate from μ and improve the ratio (either with $\delta > 0$ or $\delta < 0$ depending on the sign of $f'(0)$).

In order to derive the equation $f'(0) = 0$, we first compute the derivatives of the numerator $g(\delta) = \mathrm{UB}(\mu|\delta)$ and the denominator $h(\delta) = \mathrm{LB}(\mu|\delta)$:

$$g'(\delta) = \left(1 + \lg \frac{n}{2^i}\right) - \left(1 + \lg \frac{n}{2^{i+1}}\right) = 1$$

$$h'(\delta) = (2 + \lg(\mu + \delta)) - (2 + \lg(\mu_v - \delta)) = \lg \frac{\mu_u + \delta}{\mu_v - \delta}$$

We can write the constraint $f'(0) = 0$ in terms of these functions as follows

$$g'(0)\mathrm{LB}(\mu) - \mathrm{UB}(\mu)h'(0) = 0,$$

which we can re-write as

$$\frac{1}{\lg \frac{\mu_u}{\mu_v}} = \frac{\mathrm{UB}(\mu)}{\mathrm{LB}(\mu)} = \rho(\mu),$$

which in turn is equivalent to the relation shown in the lemma statement. ◁

Thus, for the purposes of finding a bad assignment for our analysis, we can focus our attention on those obeying the above properties. To that end, we define μ_i to be the value of those nodes in level i. Therefore, without loss of generality, we focus on the following quantities

$$\widehat{\mathrm{UB}}(\mu) = n + \sum_{i=0}^{\ell-1} k 2^i \mu_i \lg \frac{n}{2^i}$$

and

$$\widehat{\mathrm{LB}}(\mu) = n + \sum_{i=0}^{\ell-1} k 2^i \mu_i \lg \mu_i$$

Furthermore, using Claim 9 we infer that

$$\mu_i = \frac{\mu_0}{2^{\frac{i}{\rho}}} \tag{1}$$

Let $\alpha = 2^{1-\frac{1}{\rho}}$. Note that since $\rho > 1$, it follows that $1 < \alpha < 2$. Plugging (1) into the upper and lower bounds we get

$$\widehat{\mathrm{UB}}(\mu) = n + k\mu_0 \sum_{i=0}^{\ell-1} \alpha^i \lg \frac{n}{2^i}$$

and

$$\widehat{\mathrm{LB}}(\mu) = n + k\mu_0 \sum_{i=0}^{\ell-1} \alpha^i \lg \frac{\mu_0}{2^{\frac{i}{\rho}}}$$

Approximating the value of the upper bound using integrals to get:

$$\begin{aligned}
\widehat{\mathrm{UB}}(\mu) &\le n + k\mu_0 \int_1^{\ell} \alpha^x \lg \frac{n}{2^x} \mathrm{d}x \\
&= n + k\mu_0 \left[\frac{\alpha^x}{\ln \alpha} \lg \frac{n}{2^x} + \frac{\alpha^x}{\ln^2 \alpha} \right|_1^{\ell} \\
&\le n + k\mu_0 \frac{\alpha^\ell}{\ln \alpha} \left(\lg \frac{n}{2^\ell} + \frac{1}{\ln \alpha} \right)
\end{aligned}$$

Approximating the value of the lower bound yields:

$$\begin{aligned}
\widehat{\mathrm{LB}}(\mu) &\ge n + k\mu_0 \int_1^{\ell-1} \alpha^x \lg \frac{\mu_0}{2^{x/\rho}} \mathrm{d}x \\
&= n + k\mu_0 \left[\frac{\alpha^x}{\ln \alpha} \lg \frac{\mu_o}{2^{x/\rho}} + \frac{\alpha^x}{\rho \ln^2 \alpha} \right|_0^{\ell-1} \\
& n + k\mu_0 \left[\frac{\alpha^{\ell-1}}{\ln \alpha} \left(\lg \frac{\mu_o}{2^{(\ell-1)/\rho}} + \frac{1}{\rho \ln \alpha} \right) - \left(\frac{\lg \mu_0}{\ln \alpha} + \frac{1}{\rho \ln^2 \alpha} \right) \right] \\
& \ge c \left[n + k\mu_0 \frac{\alpha^{\ell-1}}{\ln \alpha} \left(\lg \frac{\mu_o}{2^{(\ell-1)/\rho}} + \frac{1}{\rho \ln \alpha} \right) \right]
\end{aligned}$$

where the last inequality holds for a constant $c > 1/2$ assuming that $\rho > 2$ and $\ell > 1$. Both of these assumptions are safe to make for otherwise $\rho = \mathcal{O}(1)$.

Finally, note that $n = \sum_{i=0}^{\ell-1} k\mu_0 \alpha^i$, which yields $n \le k\mu_0 \frac{\alpha^\ell}{\alpha - 1}$. Therefore,

$$\lg \frac{n}{2^\ell} \le \lg k + \lg \frac{\mu_0}{2^{\ell/\rho}} - \lg(\alpha - 1) \le \lg k + \lg \frac{\mu_0}{2^{(\ell-1)/\rho}} + 1,$$

where the last inequality holds for $\rho > 2$. Also, using the same assumption we note that $1/\lg \alpha = \frac{\rho}{\rho - 1} \le 2$, and so $\frac{1}{\ln \alpha} = \mathcal{O}(1)$.

Using the fact that $\mu_i \ge 1$ for all i, we get $\lg \frac{\mu_0}{2^{(\ell-1)/\rho}} \ge 0$. Therefore, the ratio $\frac{\widehat{\mathrm{UB}}(\mu)}{\widehat{\mathrm{LB}}(\mu)}$ is maximized when the previous inequality is tight, which yields that $\frac{\widehat{\mathrm{UB}}(\mu)}{\widehat{\mathrm{LB}}(\mu)} = \mathcal{O}(\log k)$. ◀

▶ **Theorem 10.** *For a graph with separation number at most k, algorithm* BALANCED *is an $\mathcal{O}(\log k)$-approximation for* MLOGA.

Proof. The claim follows readily from Lemma 8. ◀

3.1 Implementation Details

In this section we discuss implementation details of BALANCED. While the guarantee in Theorem 10 is expressed in terms of the separation number of the input graph, we observe that finding a minimum balanced separator is an NP-hard problem [4]. However, we can get the same asymptotic guarantee by applying an approximation algorithm instead.

▶ **Lemma 11.** *Algorithm* BALANCED *can be implemented to run in polynomial time while maintaining an approximation factor of* $\mathcal{O}(\log k)$*, where* k *is the separation number of the input graph.*

Proof. Feige [18] provides a polynomial time algorithm finding a balanced separator of size $\mathcal{O}(k\sqrt{k})$ provided the input graph has a balanced separator of size k. Using the approximation algorithm for finding our balanced separators and Lemma 8, we get an approximation guarantee of $\mathcal{O}(\log(k\sqrt{k})) = \mathcal{O}(\log k)$.

Each node of the divide-and-conquer recursion tree performs a polynomial amount of work, therefore the overall running time is polynomial. ◀

We close this section by noting that once a balanced separator X of G is found, it is not important how the recursively-computed layouts of each component of $G[V \setminus X]$ and X itself are sequenced – this sequencing order does not affect the analysis. An optimized implementation, would benefit from engineering a good heuristic for ordering the components: Ideally, want to place components C close to the X that have large $|E[C, X]|$ and small $|C|$; however, these two metrics may be at odds with one another, so the heuristic would have to balance those two objectives.

3.2 Related Algorithms

Let us discuss the consequences of the analysis of Section 3 to other algorithms.

Bisection

The state-of-the-art approach for MLogA uses recursive graph bisection [11, 31]. Start with a given graph, G, and find a small almost balanced edge-cut, that is, a collection of edges whose removal yields two almost-equal-sized subgraphs. Then recursively layout each of the two subgraphs, and then concatenate the resulting orders.

It is natural to wonder if this is a good heuristic provided the balanced cuts found by the algorithm are relatively small. This is indeed the case, since the endpoints of the edges in an almost-balanced cut form an almost-balanced separator. Using a similar analysis technique to Theorem 10, one can show that if the bisection algorithm always finds almost-balanced cuts whose size is at most k then the solution found is $\mathcal{O}(\log k)$-approximate.

Centroid decomposition

Chung [8] proposed an optimal algorithm for MLA on trees that is based on the idea of removing the centroid of the tree, recursively finding a layout of each subtree and carefully concatenating these subtrees.

A similar algorithm (but without the need to be careful about how the subproblems are combined) is an $\mathcal{O}(1)$-approximation for MLogA on trees since the centroid is an almost-balanced separator.

4 Conclusions and Open Problems

In this paper we tackled a practical problem arising in graph compression. We studied approximation algorithms for MLogA, which was posed as an open question by Chierichetti et al. [6] and Dhulipala et al. [11]. Our main result, an approximation based on balanced separators, partially explains why the state-of-the-art heuristic (that uses a similar scheme) works well in practice.

There are several interesting open questions related to the problem. First, the complexity of MLogA on simple graphs and graphs of bounded treewidth is open. We emphasize that the related problem, MLA, can be solved on trees in polynomial time [1, 8, 24]. These algorithms rely on certain properties of optimally embedded trees for the linear objective, and it is unclear whether similar properties hold for the logarithmic objective. The complexity status of MLA on 2-trees (series-parallel graphs) is unsettled [15].

Another natural question is to design a constant-factor approximation algorithm for general graphs. We stress that Theorem 10 provides such an algorithm for graphs with a constant separation number. At the same time, graphs without small separators (e.g., with a constant conductance) have cost $\Omega(m \log n)$; thus, any order of the vertices yields a cost that is within a constant factor of the optimum. The challenge is to analyze the scenario between the two extremes.

Finally, we would like to see some progress on designing practical exact approaches for MLogA. To the best of our knowledge, there is no algorithm that works faster than the naive exhaustive search of $n!$ combinations. Can we solve the problem (exactly) in $\mathcal{O}(c^n)$ time for some constant $c > 0$? Is there an efficient integer programming formulation of the problem? We emphasize that the two questions are interesting even when the input graph is a tree.

References

1 D Adolphson and T Ch Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973. doi:10.1137/0125042.

2 Maciej Besta and Torsten Hoefler. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *CoRR*, abs/1806.01799, 2018.

3 Avrim Blum, Prasad Chalasani, Don Coppersmith, William R. Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proc. of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–171, 1994. doi:10.1145/195058.195125.

4 Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is np-hard. *Inf. Process. Lett.*, 42(3):153–159, 1992. doi:10.1016/0020-0190(92)90140-Q.

5 Moses Charikar, Mohammad Taghi Hajiaghayi, Howard Karloff, and Satish Rao. L_2^2 spreading metrics for vertex ordering problems. *Algorithmica*, 56(4):577–604, 2010. doi:10.1007/s00453-008-9191-1.

6 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Knowledge Discovery and Data Mining*, pages 219–228, 2009. doi:10.1145/1557019.1557049.

7 Fan-Rong King Chung. A conjectured minimum valuation tree. *SIAM Review*, 20(3):601–604, 1978. doi:10.1137/1020084.

8 Fan-Rong King Chung. On optimal linear arrangements of trees. *Computers & mathematics with applications*, 10(1):43–60, 1984. doi:10.1016/0898-1221(84)90085-3.

9 Johanne Cohen, Fedor Fomin, Pinar Heggernes, Dieter Kratsch, and Gregory Kucherov. Optimal linear arrangement of interval graphs. In *International Symposium on Mathematical Foundations of Computer Science*, pages 267–279. Springer, 2006. doi:10.1007/11821069_24.

10 Nikhil R Devanur, Subhash A Khot, Rishi Saket, and Nisheeth K Vishnoi. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *Symposium on Theory of Computing*, pages 537–546, 2006. doi:10.1145/1132516.1132594.

11 Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1535–1544, 2016. doi:10.1145/2939672.2939862.

12 Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002. doi:10.1145/568522.568523.

13 Markus Sortland Dregi and Daniel Lokshtanov. Parameterized complexity of bandwidth on trees. In *International Colloquium on Automata, Languages, and Programming*, pages 405–416, 2014. doi:10.1007/978-3-662-43948-7_34.

14 Zdeněk Dvořák and Sergey Norin. Treewidth of graphs with balanced separations. *Journal of Combinatorial Theory, Series B*, 137:137–144, 2019. doi:10.1016/j.jctb.2018.12.007.

15 Martina Eikel, Christian Scheideler, and Alexander Setzer. Minimum linear arrangement of series-parallel graphs. In *Proc. of the 12th International Workshop on Approximation and Online Algorithms*, pages 168–180. Springer, 2014. doi:10.1007/978-3-319-18263-6_15.

16 Shimon Even and Yossi Shiloach. NP-completeness of several arrangement problems. Technical report 43, Dept. Computer Science, Technicon, Haifa, Isreal, 1975.

17 Uriel Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci.*, 60(3):510–539, 2000. doi:10.1006/jcss.1999.1682.

18 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.

19 Uriel Feige and James R. Lee. An improved approximation ratio for the minimum linear arrangement problem. *Inf. Process. Lett.*, 101(1):26–29, 2007. doi:10.1016/j.ipl.2006.07.009.

20 Uriel Feige and Kunal Talwar. Approximating the bandwidth of caterpillars. *Algorithmica*, 55(1):190–204, 2009. doi:10.1007/s00453-007-9002-0.

21 Peter Fishburn, Prasad Tetali, and Peter Winkler. Optimal linear arrangement of a rectangular grid. *Discrete Mathematics*, 213(1-3):123–139, 2000. doi:10.1016/S0012-365X(99)00173-9.

22 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

23 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

24 MK Goldberg and IA Klipker. Minimal placing of trees on a line. In *Physico-Technical Institute of Low Temperatures*. Academy of Sciences of Ukranian SSR, 1976.

25 Anupam Gupta. Improved bandwidth approximation for trees and chordal graphs. *J. Algorithms*, 40(1):24–36, 2001. doi:10.1006/jagm.2000.1118.

26 Lawrence Hueston Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964. doi:10.1137/0112012.

27 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Oper. Res.*, 18(6):1138–1162, 1970. doi:10.1287/opre.18.6.1138.

28 Martin Juvan and Bojan Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992. doi:10.1016/0166-218X(92)90229-4.

29 T. Kloks. *Treewidth: Computations and Approximations*. Springer-Verlag New York, 1994.

30 Yehuda Koren and David Harel. A multi-scale algorithm for the linear arrangement problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 296–309. Springer, 2002. doi:10.1007/3-540-36379-3_26.

31 Joel Mackenzie, Antonio Mallia, Matthias Petri, J. Shane Culpepper, and Torsten Suel. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Advances in Information Retrieval*, pages 339–352, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-15712-8_22.

32 Silviu Maniu, Pierre Senellart, and Suraj Jog. An experimental study of the treewidth of real-world graph data. In Pablo Barceló and Marco Calautti, editors, *Proc. of the 22nd International Conference on Database Theory*, volume 127, pages 12:1–12:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.12.

33 Julián Mestre, Sergey Pupyrev, and Seeun William Umboh. On the Extended TSP problem. In *Proc. of the 32nd International Symposium on Algorithms and Computation*, volume 212, pages 42:1–42:14, 2021. doi:10.4230/LIPIcs.ISAAC.2021.42.

34 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993. doi:10.1287/moor.18.1.1.

35 Satish Rao and Andréa W. Richa. New approximation techniques for some linear ordering problems. *SIAM Journal on Computing*, 34(2):388–404, 2005. doi:10.1137/S0097539702413197.

36 Neil Robertson and Paul D Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

37 James B Saxe. Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM Journal on Algebraic Discrete Methods*, 1(4):363–369, 1980. doi:10.1137/0601042.

38 Wikipedia contributors. Hilbert curve. URL: https://en.wikipedia.org/wiki/Hilbert_curve.

Bi-Criteria Approximation Algorithms for Bounded-Degree Subset TSP

Zachary Friggstad
Department of Computing Science, University of Alberta, Edmonton, Canada

Ramin Mousavi
Department of Computing Science, University of Alberta, Edmonton, Canada

Abstract

We initiate the study of the Bounded-Degree Subset Traveling Salesman problem (BDSTSP) in which we are given a graph $G = (V, E)$ with edge cost $c_e \geq 0$ on each edge e, degree bounds $b_v \geq 0$ on each vertex $v \in V$ and a subset of terminals $X \subseteq V$. The goal is to find a minimum-cost closed walk that spans all terminals and visits each vertex $v \in V$ at most $\frac{b_v}{2}$ times. In this paper, we study bi-criteria approximations that find tours whose cost is within a constant-factor of the optimum tour length while violating the bounds b_v at each vertex by additive quantities.

If $X = V$, an adaptation of the Christofides-Serdyukov algorithm yields a $(3/2, +4)$-approximation, that is the tour passes through each vertex at most $b_v/2 + 2$ times (the degree of v in the multiset of edges on the tour being at most $b_v + 4$). This is enabled through known results in bounded-degree Steiner trees and integrality of the bounded-degree Y-join polytope. The general case $X \neq V$ is more challenging since we cannot pass to the metric completion on X. However, it is at least simple to get a $(5/3, +4)$-bicriteria approximation by using ideas similar to Hoogeveen's TSP-Path algorithm.

Our main result is an improved approximation with marginally worse violations of the vertex bounds: a $(13/8, +6)$-approximation. We obtain this primarily through adapting the bounded-degree Steiner tree approximation to ensure certain "dangerous" nodes always have even degree in the resulting tree which allows us to bound the cost of the resulting degree-bounded Y-join. We also recover a $(3/2, +8)$-approximation for this general case.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases Linear programming, approximation algorithms, combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.8

Funding *Zachary Friggstad*: Supported by an NSERC Discovery Grant and Accelerator Supplement.

1 Introduction

Consider the problem of having a very disruptive vehicle travel about a road network to serve some locations X. As with classic TSP, one could be interested in minimizing the total distance of this tour. However, we may want to restrict the number of times the vehicle passes through a location due to its disruptive nature. Or perhaps the driver does not wish to pass through a location too many times, e.g. it is difficult to traverse. In this paper, we consider approximations for this problem that induce only mild violations on these restrictions.

Without these traversal restrictions, the problem is equivalent to classic Traveling Salesman Problem (TSP), e.g. by considering the metric completion of the underlying graph and then restricting it to X. In an instance of TSP, we are given a graph $G = (V, E)$ with edge cost $c_e \geq 0$ on each edge $e \in E$. The goal is to find a shortest tour that visits all the vertices at least once. The Chistofides-Serdyukov algorithm [2, 17] from 1976 gives a simple $\frac{3}{2}$-approximation for TSP; this was only recently improved to $(\frac{3}{2} - \delta)$-approximation for some constant $\delta > 0$ by Karlin, Klein, and Oveis Gharan [6]. On the other hand, it is NP-hard to approximate TSP within a constant factor smaller than $\frac{123}{122}$ [7].

In reality, TSP problems are concerned with visiting a subset of nodes of some larger graph. Thus, we consider the Bounded-Degree Subset Traveling Salesman problem (BDSTSP) in which we are given an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0, e \in E$, a subset of terminals $X \subseteq V$ we are to visit ($|X| \geq 2$), and even integer bounds $b_v \geq 0$ for all nodes $v \in V$. The goal is to find a minimum-cost closed walk $\mathcal{Q}$ spanning all terminals such that $d_{\mathcal{Q}}(v) \leq b_v$ (i.e., the number of edges in the multiset $\mathcal{Q}$ incident to v is at most b_v). Note b_v should be thought of as a degree bound, thus the tour should pass through v at most $b_v/2$ times. We call a special case of BDSTSP where $X = V$, Bounded-Degree Traveling Salesman problem (BDTSP).

To the best of our knowledge (and to our surprise), BDTSP or BDSTSP have not been studied before. However, finding special subgraphs whose vertices satisfy some degree bounds has been an active research area in computer science and operations research, e.g., Bounded-Degree Spanning Trees [5, 18], Bounded-Degree Steiner Networks [9, 11, 12, 13], Bounded-Degree Element-Connectivity and Bounded-Degree Vertex-Connectivity [4, 8].

Throughout this paper by a *tour* we mean a closed walk that spans all the required vertices, i.e., a tour in BDTSP is a closed walk that contains all the vertices and a tour in BDSTSP is a closed walk that contains all the terminals and possibly other vertices. The cost of a tour is the sum of the cost of the edges in the tour (counting with multiplicity). Note the edge set of a tour is potentially a multiset, we may be required to span an edge multiple times. All of our algorithms are based on linear-programming relaxations: if the LPs are not feasible then we report there is no feasible solution. Otherwise, we will find an $(\alpha, +d)$-approximate solution: the cost of the tour will be at most α times the value of the LP relaxation and will visit each node at most $(b_v + d)/2$ times (i.e. the degree of the tour at v will be at most $b_v + d$). We note that if there is a feasible solution, then the LP relaxations we use will be feasible and will have value at most the optimum solution value.

1.1 Our Results and Techniques

As a warm up, in Section 3 we present a simple $(3/2, +4)$-approximation algorithm for BDTSP (i.e. if $X = V$).

▶ **Theorem 1** (BDTSP). *There is a $(3/2, +4)$-approximation algorithm for BDTSP.*

Since a feasible solution is an Eulerian graph and b_v's are even, if there is an approximation algorithm whose degree violation is better than additive factor of 2, then this algorithm can decide the Hamiltonian cycle problem. Hence, assuming P $\neq$ NP, the additive factor of 2 violation on degree bounds is necessary. Furthermore, the same integrality gap example for Held-Karp relaxation where the degree bound on every vertex is 2, see Figure 1, shows the integrality gap of the natural LP formulation (**BDTSP-LP**) is at least $(4/3, +2)$, meaning any tour has cost at least 4/3 times the LP optimum and any tour violates the degree bound of at least one vertex by at least +2.

The proof is a straightforward adaptation of Wolsey's analysis [19] of the Christofides-Serdyukov algorithm for TSP so we sketch it here to discuss our techniques. Let x^* be an optimal solution for the natural LP formulation of BDTSP. Step (1): using the natural cut-based LP formulation (augmented with degree bounds) for spanning trees and the fact that $\frac{x^*}{2}$ is feasible for this LP, using the rounding technique in [11] one can obtain a spanning tree T of cost at most $\sum_{e \in E} c_e \cdot x^*_e$ whose degree on vertex v is at most $\frac{b_v}{2} + 3$. Step (2): fix the degree parities of T using Y-join polytope augmented with degree bounds at most $\frac{b_v}{2} + 1$ (depending on the parity of v's degree in the tour) and show $\frac{x^*}{2}$ is feasible for this LP.

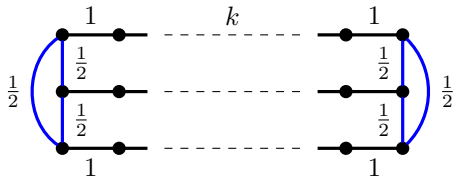


Figure 1 This is the graph $G = (V, E)$ that shows the $(\frac{4}{3}, +2)$ integrality gap of the natural LP for BDTSP (**BDTSP-LP**). All vertices are terminals, the cost of blue edges is zero and the cost of black edges is 1. The LP value on blue edges are $\frac{1}{2}$, and 1 on all the other edges. Also $b_v = 2$ for all $v \in V$. Note that the cost of the LP is $3 \cdot k$ and satisfies all the degree bounds. However, in any integer solution we must cross one of the path of length k at least twice. Therefore, any integer solution will violate the degree constraint by at least an additive factor of 2 and its cost is at least $4 \cdot k$ which give the desired integrality gap.

▶ Remark 2. Notice we did not use the $+1$ algorithm for degree-bounded spanning trees by [18]. This is because dividing x by 2 is only guaranteed to satisfy the weaker cut-based LP relaxation for spanning trees.

The main focus of this paper is on BDSTSP (i.e. $X \neq V$). Our results present different approximation/violation tradeoffs.

▶ **Theorem 3.** *There is a* $(5/3, +4)$*-approximation algorithm for BDSTSP.*

▶ **Theorem 4.** *There is a* $(13/8, +6)$*-approximation algorithm for BDSTSP.*

▶ **Theorem 5.** *There is a* $(3/2, +8)$*-approximation algorithm for BDSTSP.*

In each of these, we first adapt step (1) from BDSTP discussed above, compute a Steiner tree (instead of spanning tree) T using $\frac{x^*}{2}$ as a fractional solution to the Bounded-Degree Steiner Tree polytope. However, step (2) is not applicable since $\frac{x^*}{2}$ might not be feasible for the Y-join polytope.

To prove Theorem 3, it is easy to show that $\frac{1}{3} \cdot (\chi_T + x^*)$ is feasible for the degree-bounded Y-join polytope where χ_T is the characteristic vector of T which yields $(5/3, +4)$-approximation factor[1]. In order to improve the cost factor, we first augment the natural LP for BDSTSP with non-trivial constraints asserting the degree of Steiner cuts should be at least the degree of any Steiner node in the cut. Then, we modify the iterative rounding algorithm of [11] using splitting off techniques by Mader to obtain a more "structured" Steiner tree. Namely, some Steiner nodes are designated *dangerous* because they have low fractional degree in our LP solution: our modification ensures dangerous nodes will have even degree in the resulting tree. Finally, we show how this Steiner tree helps us to obtained a better bounded-degree Y-join to fix the degree parity of odd-degree vertices.

2 Preliminaries

In this section, we state definition and recall previous work that will be used throughout the paper. All graphs may be multigraphs and all subsets of edges may be multisubsets, we adopt this convention now so we do not have to use the prefix *multi* on every set or graph. In particular, when we discuss the degree of a vertex with respect to a set of edges or with

[1] Interestingly, this is basically the same fractional join from [1] that could be formed to analyze Hoogeveen's TSP-Path algorithm.

respect to a graph, we mean its degree if we count all edges with the same multiplicity that they appear in the set/graph. However, all subsets of vertices will be actual sets: each vertex will be in the set at most once.

Given a graph (or subgraph) $H = (V, E)$, denote by $d_H(v)$ the degree of vertex v in H. For a subset $S \subseteq V$, we denote by $\delta_H(S)$ the set of edges in $E(H)$ with exactly one endpoint in S while $E_H[S]$ is the set of all edges in $E(H)$ having both endpoints in S. We define $cost_c(H) := \sum_{e \in E(H)} c_e$. We may drop the subscripts in the above notation if the underlying graph/cost is clear from the context. Denote by $odd(H)$ the set of all vertices with odd degree in H, i.e., $odd(H) = \{v \in V(H) : \ d_H(v) \text{ is odd}\}$. For a subset of edges (or a subgraph) F, we denote by χ_F the characteristic vector of the edges in F (i.e., $\chi_F(e) = 1$ if $e \in F$ and zero otherwise). For a solution x of an LP by $cost(x)$ we mean the value of the objective function given solution x. We sometime use notation $|A| = odd$ which means $|A| \equiv 1 \pmod 2$, similarly we define the notation $|A| = even$.

We use the Bounded-Degree Steiner Tree problem (BDSTP) and the Bounded-Degree Y-Join problem (BD-Y-join) in our results. In BDSTP, the input is a tuple $\big(G = (V, E), X, c, b\big)$, where c is the edge cost, i.e., $c_e \geq 0$ on each edge $e \in E$, $X \subseteq V$ is a set of terminals, and b is a degree bound for a subset of vertices $W \subseteq V$, i.e., $b_v \in \mathbb{Z}_{\geq 0}$ for all $v \in W$. Non-terminal vertices are called *Steiner* nodes. The goal is to compute a minimum cost connected subgraph T that spans all the terminals and respects the degree bounds, i.e., $d_T(v) \leq b_v$ for all $v \in W$.

There is a natural cut-based LP relaxation for this problem. Let $W \subseteq V$:

$$\text{minimize:} \quad \sum_{e \in E} c_e \cdot x_e \qquad \textbf{(SNDP-LP)}$$

$$\begin{aligned}
\text{subject to:} \quad x(\delta(v)) &\leq b_v \quad \forall v \in W && (1)\\
x(\delta(S)) &\geq 1 \quad \forall S \neq X,\ S \cap X \neq \emptyset && (2)\\
x &\geq 0 && (3)
\end{aligned}$$

Lau and Singh [11] presented an iterative rounding algorithm for this LP.

▶ **Theorem 6** (Theorem 1.1 in [11]). *There exists a polynomial time algorithm for Bounded-Degree Steiner Tree which returns a Steiner tree of cost at most $2 \cdot opt$ with additive degree violation of at most 3, where opt is the cost of an optimal Steiner tree. In our notation, this is a $(2, +3)$-approximation algorithm.*

The above theorem is based on an iterative rounding method that rounds an extreme point of (**SNDP-LP**) to an integral solution. The following is an immediate consequence of the above theorem.

▶ **Corollary 7.** *Let $\bar{x}$ be a feasible solution to* (**SNDP-LP**). *Then, in polynomial time, one could get a Steiner tree T of cost at most $2 \cdot cost(\bar{x})$ and $d_T(v) \leq b_v + 3$ for all $v \in W$.*

In the BD-Y-join problem, we are given a graph $G = (V, E)$, a non-negative cost c_e on each edge $e \in E$, a subset $Y \subseteq V$ with even size, and a degree bound $b_v \in \mathbb{Z}_{\geq 0}$ on each vertex such that b_v is odd for all $v \in Y$ and even otherwise. The goal is to find a minimum cost subset of edges $J \subseteq E$ such that $|d_J(v)| = odd$ if and only if $v \in Y$. Furthermore, we want $|d_J(v)| \leq b_v$ for all $v \in V$. There is a natural LP relaxation for this problem as well, which is just augmenting the integral Y-join polyhedron with the degree constraints.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e\in E} c_e \cdot x_e && && \text{(BD-}Y\text{-join LP)}\\
\text{subject to:} \quad & x(\delta(v)) \leq && b_v \quad \forall v \in V && (4)\\
& x(\delta(S)) \geq && 1 \quad \forall S \subsetneq V,\ |S \cap Y| = odd && (5)\\
& x \geq && 0 && (6)
\end{aligned}$$

It is known that (**BD-Y-join LP**) is integral.

▶ **Theorem 8** (Theorem 36.8 in [16]). (**BD-Y-join LP**) *is integral, if and only if, b_v is odd if $v \in Y$ and even otherwise for all $v \in V$.*

The above result is a corollary of Theorem 36.8 in [16] but it is not trivial to see at the first sight. So for the sake of completeness, we present a self-contained proof of Theorem 8 in Appendix C. Our proof is based on the iterative rounding technique which is different and simpler than the proof stated in [16] for Theorem 36.8, as we are trying to prove a special case of Theorem 36.8.

Also note that (**BD-Y-join LP**) admits a polynomial-time separation oracle since the odd-cut constraints can be separated just like with the classic Y-join polyhedron (eg. by using Gomory-Hu trees). So we can find a minimum-cost degree-bounded Y-join in polynomial time or determine one does not exist.

3 Bounded-Degree TSP (Warm Up!)

We quickly present the simple result for BDTSP in order to warm the reader up to how we use the results cited in the last section. Fix an instance of BDTSP: $G = (V, E)$, edge cost $c_e \geq 0$ for all $e \in E$, even degree bound $b_v \geq 0$ for all $v \in V$. The following is a natrual LP formulation for this problem. For each edge e, there is a variable x_e indicating whether e is in the solution or not. Note that in an optimal solution for the problem, we might need to pick an edge twice but not more than twice since, otherwise, one can reduce its occurrence by two and retain connectivity.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e\in E} c_e \cdot x_e && && \text{(BDTSP-LP)}\\
\text{subject to:} \quad & x(\delta(S)) \geq && 2 \quad \forall\, \emptyset \neq S \subsetneq V && (7)\\
& x(\delta(v)) \leq && b_v \quad \forall\, v \in V && (8)\\
& 0 \leq x_e \leq && 2 \quad \forall\, e \in E && (9)
\end{aligned}$$

One can separate the constraints using a minimum-cut algorithm, so we can find an optimal solution (or determine (**BDTSP-LP**) is infeasible) in polynomial time. If the LP is infeasible, we report there is no feasible solution and terminate. Otherwise, we proceed as follows.

The algorithm is very similar to Wolsey's analysis of Christofides-Serdyukov algorithm. First we compute at spanning tree T using an optimal solution to (**BDTSP-LP**) and then we fix the degree parities using the $odd(T)$-join polytope. However, we need to respect (approximately) the degree bounds. For a node v, let $b'(v, T)$ be the smallest integer at least $\frac{b_v}{2}$ whose parity is the same as $|d_T(v)|$: note $b'(v, T) \in \{\frac{b_v}{2}, \frac{b_v}{2} + 1\}$.

Algorithm 1 $(3/2, +4)$-approximation algorithm for BDTSP.

Input: Graph $G = (V, E)$ with edge costs $c_e \geq 0$ for every $e \in E$ and even degree bounds b_v for every $v \in V$.

Output: A tour that spans V.

Let T be a Steiner tree (in this case spanning tree) obtained from applying Theorem 6 with input $G = (V, E)$, edge cost c_e for $e \in E$, $X := V$ and degree bounds $\frac{b_v}{2}$ for every $v \in V$.

Let $odd(T)$ be the set of vertices with odd degrees with respect to T. Compute an $odd(T)$-join J in G using Theorem 8 with degree bounds $b'(v, T)$ for all $v \in V$.

Output a closed spanning walk in $T \cup J$.

We show Algorithm 1 works correctly and this proves Theorem 1.

Proof of Theorem 1. Let x^* be an optimal solution to (**BDTSP-LP**). Note that $\bar{x} := \frac{x^*}{2}$ is feasible for (**SNDP-LP**) where $X := V$ and degree bounds $\frac{b_v}{2}$ for every $v \in V$. Hence, by Theorem 6 we have $cost(T) \leq 2 \cdot cost(\bar{x}) = cost(x^*) \leq opt$. Furthermore, $d_T(v) \leq \frac{b_v}{2} + 3$.

Consider vertex v in the graph, note that $\bar{x}(\delta(v)) \leq \frac{b_v}{2}$. By using degree bounds $b'(T, v)$ for $v \in V$, we ensure (**BD-Y-join LP**) is integral and that $\bar{x}$ is a feasible solution. Thus, a minimum-cost degree-bounded $odd(T)$-join J has cost at most $cost(\bar{x}) = cost(\frac{x^*}{2})$ and $d_J(v) \leq \frac{b_v}{2} + 1$ for every $v \in V$.

Putting the bounds on T and J together we have an Eulerian subgraph $T \cup J$ with cost $\frac{3}{2} \cdot cost(x^*)$ and $d_{T \cup J}(v) \leq b_v + 4$. ◀

4 Bounded-Degree Subset TSP

In this section, we prove our main results, i.e., Theorems 3, 4 and 5. Fix an instance of BDSTSP: $G = (V, E)$, a edge cost $c_e \geq 0$ for each $e \in E$, a set of terminals $X \subseteq V$, and an even integer degree bound $b_v \geq 0$ on each vertex $v \in V$. We refer to vertices in $V \setminus X$ as *Steiner* nodes.

The algorithm for BDSTSP is to find a "good" Steiner tree T that spans the terminals and then fix the degree parity using $odd(T)$-join. However, finding a "good" $odd(T)$-join is not as trivial as it was for BDTSP since $\frac{x^*}{2}$ may no longer be feasible for $odd(T)$-join polytope since some cuts S involving only Steiner nodes may have very low $x^*(\delta(S))$. Nevertheless, for all the approximation factors in this section, we show combining x^* and T itself with appropriate ratios is sufficient to construct a "good" (fractional) solution for the $odd(T)$-join polytope with degree constraints. We start with a simple application of this idea by proving Theorem 3.

We begin with the natural LP for BDSTSP. As before, we assume the LP has an optimal solution, otherwise the BTSTSP instance has no feasible solution.

$$\text{minimize:} \quad \sum_{e \in E} c_e \cdot x_e \qquad \text{(BDSTSP-Natural-LP)}$$

$$\text{subject to:} \quad x(\delta(S)) \geq 2 \quad \forall S \neq X,\ S \cap X \neq \emptyset \qquad (10)$$

$$x(\delta(v)) \leq b_v \quad \forall v \in V \qquad (11)$$

$$0 \leq x_e \leq 2 \quad \forall e \in E \qquad (12)$$

Proof of Theorem 3. Let x^* be an optimal solution to (**BDSTSP-Natural-LP**). Since $\frac{x^*}{2}$ is feasible for (**SNDP-LP**) where degree bounds are $\frac{b_v}{2}$ for every $v \in V$, we can obtain a Steiner tree T of cost at most $2 \cdot cost(\frac{x^*}{2}) \leq opt$ and $d_T(v) \leq \frac{b_v}{2} + 3$, see Corollary 7. Furthermore, we iteratively prune leaf nodes that are Steiner nodes so all leaves of T are terminals. With abuse of notation, we denote the resulting tree by T.

Next, we show that $\bar{y} := \frac{\chi_T}{3} + \frac{x^*}{3}$ is feasible for (**BD-Y-join LP**) when $odd(T)$ is the set of odd degree vertices and the RHS of degree constraint is either $\frac{b_v}{2} + 1$ or $\frac{b_v}{2} + 2$ (whichever has the same parity as $d_T(v)$). Note that by definition of $\bar{y}$, and the fact that $d_T(v) \leq \frac{b_v}{2} + 3$, $\bar{y}$ respects the degree constraints in (**BD-Y-join LP**). Now consider a cut S that contains a terminal. Then T crosses the cut at least once and $x^*(\delta(S)) \geq 2$ so $\bar{y}(\delta(S)) \geq 1$.

Now consider an odd cut S, i.e. $|S \cap odd(T)| = odd$, that contains only Steiner nodes. Since $\sum_{v \in S} d(v) = 2 \cdot |E_T[S]| + |\delta_T(S)|$ and $\sum_{v \in V} d(v)$ is odd, we must have $|\delta_T(S)| = odd$. We claim that $|\delta_T(S)| > 1$, otherwise $|\delta_T(S)| = 1$ means S contains a leaf node which is impossible since (the pruned version) of T has only terminals as leaf nodes. Therefore, $|\delta_T(S)| \geq 3$ and by definition of $\bar{y}$ we have $\bar{y}(\delta(S)) \geq 1$, as desired.

So we have proved $\bar{y}$ is feasible for (**BD-Y-join LP**). By Theorem 8, there is an $odd(T)$-join J of cost at most $cost(\bar{y}) = \frac{1}{3} \cdot cost(T) + \frac{1}{3} \cdot cost(x^*) \leq \frac{2}{3} \cdot opt$ and $d_J(v) \leq \frac{b_v}{2} + 2$ for all $v \in V$. Finally we output a closed walk in subgraph $\mathcal{Q} := T \cup J$. Note $cost(\mathcal{Q}) \leq (1 + \frac{2}{3}) \cdot opt = \frac{5}{3} \cdot opt$ and $d_{\mathcal{Q}}(v) \leq b_v + 5$ for all $v \in V$. Since $\mathcal{Q}$ is an Eulerian graph and b_v's are even, it must be that $d_{\mathcal{Q}}(v) \leq b_v + 4$ for all $v \in V$. This finishes the proof of Theorem 3. ◀

To improve on the approximation factor of Theorem 3, i.e. the results in Theorem 4 & 5, we consider a slight strengthening of (**BDSTSP-Natural-LP**) for BDSTSP. We first make an observation about the structure of an optimal solution and then we add a constraint based on this observation.

We use the following definition throughout this section. Let v be a Steiner node, we say S is a v, X-cut if $v \in S \subseteq V \setminus X$.

▶ **Lemma 9.** *There exists an optimal solution $\mathcal{Q}^*$ such that for any Steiner node $\bar{v}$ and any $\bar{v}, X$-cut S, we have $|\delta_{\mathcal{Q}^*}(S)| \geq d_{\mathcal{Q}^*}(\bar{v})$.*

Proof. Among all optimal solutions, let $\mathcal{Q}^*$ be one with the minimum number of edges. For this proof, every degree or cut is with respect to $\mathcal{Q}^*$ unless stated otherwise. We show $|\delta_{\mathcal{Q}^*}(S)| \geq d_{\mathcal{Q}^*}(\bar{v})$ for every $\bar{v}$ and any $\bar{v}, X$-cut S. Suppose otherwise, that for $\bar{v}$ there is some $\bar{v}, X$-cut S with $|\delta_{\mathcal{Q}^*}(S)| < d_{\mathcal{Q}^*}(\bar{v})$. We take S to be a minimum-cardinality $\bar{v}, X$-cut.

Let $k := |\delta(S)|$. Note $\sum_{v \in S} d(v) = 2 \cdot |E_{\mathcal{Q}^*}[S]| + |\delta(S)|$ and since all vertices have even degree, k must be even. We say $u \in S$ is a *boundary node* with respect to S if $\delta(u) \cap \delta(S) \neq \emptyset$.

Contract $V \setminus S$ to a single vertex and call it t. Since S is a minimum cardinality $\bar{v}, t$-cut, there are k edge-disjoint simple paths $P_1, ..., P_k$ from $\bar{v}$ to t, in particular, for every boundary node $u \in S$, $|\delta(u) \cap \delta(S)|$ of the paths $P_1, ..., P_k$ have u as their second-last node (just before t).

Construct a graph G' obtained from $\mathcal{Q}^*$ as follows: remove all the edges in $E_{\mathcal{Q}^*}[S] \setminus \cup_{i=1}^{k} P_i$ which is non-empty as we assumed $d_{\mathcal{Q}^*}(v) > k$ and then remove all the isolated vertices. Note that $d_{G'}(\bar{v}) = k$ which is even. Also for every boundary vertex $u \in S$, $|\delta(u) \cap \delta(S)|$ of P_i's contains u, so the degree of boundary vertices is even. The vertices inside S except $\bar{v}$ and the boundary vertices are internal vertices of some paths so their degrees are even. Finally, degree of the vertices outside of S did not change, so their degree is even as well. Hence, G' is Eulerian.

We claim G' connects all the terminals, which would contradict the minimality of $\mathcal{Q}^*$. Hence, $|\delta_{\mathcal{Q}^*}(S)| \geq d_{\mathcal{Q}^*}(\bar{v})$, as desired. It is easy to prove the claim. Consider two terminals x and x'. If a $x - x'$ path in $\mathcal{Q}^*$ does not use any vertices in S then this path exists in $E(G')$. So suppose every $x - x'$ paths in $\mathcal{Q}^*$ crosses S, let u and u' (possibly $u = u'$) be the two boundary vertices (with respect to S) on a $x - x'$ path in $\mathcal{Q}^*$. Note that there is a path between u and $\bar{v}$, and a path between u' and $\bar{v}$ in $E(G')$. Therefore, x and x' are connected in $E(G')$. ◀

We use Lemma 9 to get a slightly stronger LP relaxation for BDSTSP.

$$\text{minimize:} \quad \sum_{e \in E} c_e \cdot x_e \qquad \textbf{(BDSTSP-LP)}$$

$$\begin{aligned}
\text{subject to:} \quad x(\delta(S)) &\geq 2 && \forall S \neq X,\ S \cap X \neq \emptyset && (13)\\
x(\delta(v)) &\leq b_v && \forall v \in V && (14)\\
x(\delta(v)) &\leq x(\delta(S)) && \forall S \subseteq V \setminus X, \forall v \in S && (15)\\
0 \leq x_e &\leq 2 && \forall e \in E && (16)
\end{aligned}$$

Constraint (15) is valid because there is an optimal solution that has this property according to Lemma 9. Furthermore, this constraint can be separated by computing a minimum weight v, X-cut (with respect to weight x on the edges) for every Steiner node v. Let x^* be an optimal solution to this LP.

Now we are ready to discuss how to get a more well-structured Steiner "tree"[2] T using a slightly modified version of the algorithm in [11] for computing a degree-bounded Steiner tree, which in turn, results in a cheaper $odd(T)$-join.

The tweak in the algorithm of [11] for BDSTP is to completely "split off" a predetermined subset of Steiner nodes when the algorithm decides to drop the degree constraint corresponding to these Steiner nodes. This will ensure that we get a feasible solution for BDSTP (not necessarily a tree) such that all the Steiner nodes in the predetermined subset have even degree (counting with multiplicities). First we explain the splitting-off procedure and then present the tweak in the algorithm for BDSTP.

Splitting-off procedure. We begin with some definition. Fix a multigraph $G = (V, E)$. We say edge e is a *cut-edge* if there is a set S such that $\delta(S)$ contains only e. We denote the minimum cardinality of a u, v-cut by $\lambda_G(u, v)$. We say a pair of edges (su, sv) is an *splittable pair* if in $G' = \big(V \cup \{s\}, (E \setminus \{su, sv\}) \cup \{uv\}\big)$, we have $\lambda_G(u, v) = \lambda_{G'}(u, v)$ for all $u, v \in V$. In other words, removing su, sv and adding an edge between u and v preserves the minimum u, v-cut value for all $u, v \in V$. This process is called *splitting off* pair (su, sv). Recall a classical splitting-off result by Mader.

▶ **Theorem 10** (Mader, [14, 3]). *Let $G = (V \cup \{s\}, E)$ be a connected graph, perhaps with parallel edges. Assume there is no cut-edge incident to s and $d(s)$ is even. Then, there is a splittable pair (su, sv) for some u, v (possibly $u = v$) adjacent to s.*

When we remove all the splittable pairs so that there is no incident edge to s, we say we *completely split off* s. Since after applying Mader's theorem, the conditions of the theorem still holds for s, one can repeatedly apply Mader's theorem to completely split off s while preserving the local connectivities between any pair of vertices in V.

[2] We put tree in the quotation as we will see later that T might contain cycles to allow the degree parity of some Steiner nodes to be even, so T might not be a proper tree.

The following is the result that allows us to tweak the iterative rounding algorithm of [11] for BDSTP to ensure Steiner nodes with small fractional degree have even degree in the resulting tree. The proof uses Mader theorem as the subroutine. However, just applying Mader theorem repeatedly when the connectivities are based on weighted edges does not run in polynomial time. Here we use the idea used by Post and Swamy [15] to make the complete splitting off procedure polytime. In [15] they work with the directed version of Mader theorem. Although everything will be translated to our setting in a straightforward fashion, we present the proof in Appendix A for completeness.

▶ **Lemma 11.** *Let $\mathcal{I} = \big(G = (V, E), X, c, b\big)$ be an instance of the BDSTP and let $\bar{x}$ be a feasible solution for* (**SNDP-LP**) *of this instance. Let $s \in V$ be a Steiner node. Then, in polynomial time, one can obtain an instance of BDSTP $\mathcal{I}' = \big(G' = (V \setminus \{s\}, E'), X, c', b\big)$ and a feasible solution x' for* (**SNDP-LP**) *of this instance such that*

1. *$cost_{c'}(x') \leq cost_c(x)$.*
2. *An integral solution T' for $\mathcal{I}'$ can be transformed to an integral solution T for $\mathcal{I}$ whose cost is at most $cost_{c'}(T')$, $d_T(s)$ is even, and $d_T(v) = d_{T'}(v)$ for all $v \in V \setminus s$.*

Next, we modify one step of $(2, +3)$-approximation of [11] for BDSTP. For the sake of space, we only sketch the adaptation and defer the full description to Appendix B.

The iterative rounding algorithm for BDSTP. Consider an instance $\big(G = (V, E), X, c, b\big)$ of BDSTP and a subset of Steiner nodes $A \subseteq V \setminus X$ where $b_v = 1$ for all $v \in A$. The goal is to find a minimum cost solution T for this instance such that $d_T(v)$ is even for all $v \in A$ while violating the degree bounds by a small additive constant for these vertices, and $d_T(v) \leq b_v$ for all $v \in V \setminus A$ (see Lemma below). In the context of our BDSTSP, A will be the set of all Steiner nodes with "small" fractional degree (with respect to an optimal solution of (**BDSTSP-LP**)). Then, this more structured Steiner tree T helps us to find a cheaper $odd(T)$-join.

In the iterative rounding algorithm for BDSTP whenever we drop the degree constraints corresponding to a Steiner node v in A, we completely split off v as well. After the tree is constructed, we restore any edges produced by the splitting off procedure to the original set of edges and further prune some edges if necessary (i.e., if their multiplicity is > 2 after this restoration step)[3]. Again, see Appendix B for the full description of this algorithm.

Putting together Corollary 7 and Lemma 11 we have the following result for BDSTP. It is important to note that while we use the letter T to denote the solution, it is not necessarily a tree because we cannot necessarily drop edges from cycles while preserving the parity of nodes in A.

▶ **Lemma 12.** *Given an instance $\big(G = (V, E), X, c, b\big)$ of BDSTP, a feasible solution $\bar{x}$ of* (**SNDP-LP**) *and a set $A \subseteq V \setminus X$ where $b_v = 1$ for all $v \in A$, there is a polynomial time algorithm that computes a feasible solution T of BDSTP such that*

1. *$cost(T) \leq 2 \cdot cost(\bar{x})$.*
2. *$d_T(v) \leq b_v + 3$ for all $v \in V \setminus A$.*
3. *$d_T(v) \leq b_v + 7$ for all $v \in A$.*
4. *$d_T(v)$ is even for all $v \in A$.*
5. *$|\delta_T(S)|$ is even for any $S \subseteq A$.*

[3] Note we might have edges with multiplicity 2 in this solution but nevertheless we call the solution a Steiner tree.

6. *Let $T^{(m)}$ be a minimal (inclusion-wise) subset of edges of T such that $T^{(m)}$ is still a feasible solution for the BDSTP instance. Then, $T^{(m)}$ satisfies property 1 and $d_T(v) \leq b_v + 3$ for all $v \in V$.*

Proof. We run the $(2, +3)$-approximation of [11] with the above mentioned tweak, see Algorithm 3 and apply Lemma 11 (property 2) repeatedly to obtain a feasible solution T for BDSTP on G.

Properties 1 and 2 are trivial because of the approximation factor of Algorithm 3 and Lemma 11.

Property 3: we completely split off a Steiner node $s \in A$ in Algorithm 3 when the algorithm decides to drop the degree constraint corresponding to s (see step b in Algorithm 3). Hence, we must have $d(s) \leq b_s + 3 \leq 4$ in the current graph in that iteration. Once we apply Lemma 11 (property 2) to obtain a solution for the current instance of BDSTP (i.e., by putting s back) we might use these (up to) four edges incident to s multiple times; however, we do not need to use any edge more than twice (otherwise we can drop two copies of the edge and preserve both connectivity and parities) so the degree of s in the solution will be at most $8 = b_v + 7$ since $b_v = 1$ for $v \in A$. This proves property 3.

Property 4: the degree of a node $v \in A$ that was split off in our iterative rounding algorithm is even simply because each edge used in T that was produced in the splitting off procedure is then subdivided to re-integrate v so the degree of v is even. Further, any parallel edges that were pruned maintain the parity of the degree of v.

Property 5: for any set $S \subseteq V$ we have

$$\sum_{v \in S} d_T(v) = 2 \cdot |E_T[S]| + |\delta_T(S)|. \tag{17}$$

If $S \subseteq A$, by property 4 the LHS of (17) is even and therefore $|\delta_T(S)|$ must be even.

Property 6: by property 2, for all $v \in V \setminus A$ we have $d_T(v) \leq b_v + 3$. In $T^{(m)}$ we keep only one copy of each parallel edge so $d_T(v) \leq b_v + 3$ for all $v \in A$ (see the argument for property 3) and the resulting solution is still feasible which implies the last property for $T^{(m)}$. ◀

Finally, we can present our $(13/8, +6)$-approximation for BDSTSP, see Algorithm 2.

Algorithm 2 $(13/8, +6)$-approximation algorithm for BDSTSP.

Input: Graph $\big(G = (V, E), X, c, b\big)$.
Output: A closed walk $\mathcal{Q}$ in G that spans X.

(a) Compute an optimal solution x^* of (**BDSTSP-LP**). Let $A := \{v \in V \setminus X : x^*(\delta(v)) < 2\}$, and let $b'_v := \frac{b_v}{2}$ if $v \notin A$ and $b'_v := 1$ if $v \in A$.
(b) Apply Lemma 12 with instance $\big(G = (V, E), X, c, b'\big)$, feasible solution $\bar{x} := \frac{x^*}{2}$, and set A to obtain a Steiner tree T with properties 1-5 in the lemma.
(c) Compute a Steiner tree $T^{(m)}$ using T according to property 6 of Lemma 12.
(d) Compute a minimum cost $odd(T^{(m)})$-join J such that $d_J(v) \leq b'_v + 3$ for all $v \in V$ (cf. Lemma 13).
(e) Output a closed walk $\mathcal{Q}$ in $T^{(m)} \cup J$ that spans all the terminals.

Analysis. For the rest of this section, let x^* be an optimal solution for (**BDSTSP-LP**) computed in step (a) and $\bar{x} := \frac{x^*}{2}$. Let A be the subset of Steiner nodes and b'_v's the degree bounds constructed based on x^* in step (a). Also let T and $T^{(m)}$ be the Steiner trees

computed in steps (b) and (c) of the algorithm, respectively. Note that $\bar{x}$ is feasible for (**SNDP-LP**) when the degree bounds are according to b'. Combining this fact and Lemma 12 we have:

$$cost(T^{(m)}) \leq cost(T) \leq 2 \cdot cost(\bar{x}) = cost(x^*). \tag{18}$$

▶ **Lemma 13.** *There is an $odd(T^{(m)})$-join J with cost at most $\frac{5}{8} \cdot opt$ and $d_J(v) \leq \frac{b_v}{2} + 3$ for all $v \in V$.*

Proof. We claim $y := \frac{\chi_T}{4} + \frac{3 \cdot x^*}{8}$ is feasible for (**BD-Y-join LP**) when the set of odd degree vertices is $odd(T^{(m)})$ and the RHS of degree constraint is at most $b'_v + 3$ for all $v \in V$ (in fact this fractional solution is feasible when degree bounds are $\frac{b_v}{2} + 2$ but similar to the proof of Theorems 1 and 3, we might need to consider $\frac{b_v}{2} + 3$ for some vertices as (**BD-Y-join LP**) is integral if and only if the parity of the degree bounds match the parity of the degree of vertices in an integral solution). We prove the claim after we show how the lemma follows from this claim. By Theorem 8, there is an integral $odd(T^{(m)})$-join J whose cost is at most

$$cost(y) = cost(\frac{1}{4} \cdot \chi_T) + cost(\frac{3}{8} \cdot x^*) \leq \frac{1}{4} \cdot cost(x^*) + \frac{3}{8} \cdot cost(x^*) \leq \frac{5}{8} \cdot opt,$$

where the first inequality follows from (18). Furthermore, $d_J(v) \leq b'_v + 3 \leq \frac{b_v}{2} + 3$.

So it remains to prove the claim. As (19) shows, y satisfies the degree constraints of (**BD-Y-join LP**) when the degree bound is at most $b'_v + 3$ for all $v \in V$.

$$y(\delta(v)) = \frac{1}{4} \cdot d_T(v) + \frac{3}{8} \cdot x^*(\delta(v)) \leq \frac{1}{4} \cdot (b'_v + 7) + \frac{3}{8} \cdot b_v \leq \frac{b_v}{2} + 3, \tag{19}$$

where the first inequality follows from properties 2 & 3 of Lemma 12.

Next we show cut constraints (5) in (**BD-Y-join LP**) hold under solution y. Consider a subset $S \subseteq V$ such that $|S \cap odd(T^{(m)})| = odd$. There are three cases to consider:

- Case 1: If $S \cap X \neq \emptyset$. Then, $x^*(\delta(S)) \geq 2$ and since T is connected we have $|\delta_T(S)| \geq 1$ which implies $y(\delta(S)) \geq 1$.
- Case 2: If $S \cap X = \emptyset$ and $S \cap (V \setminus A) \neq \emptyset$. Then, there is a Steiner node $s \in S$ such that $s \notin A$. By definition of set A we have $x^*(\delta(s)) \geq 2$. By constraint (15) in (**BDSTSP-LP**) this implies $x^*(\delta(S)) \geq 2$ as well. Again since T is a connected so $|\delta_T(S)| \geq 1$ and this implies $y(\delta(S)) \geq 1$.
- Case 3: If $S \subseteq A$. Since (17) holds for any subset of vertices and $|S \cap odd(T^{(m)})| = odd$, the LHS of (17) is odd and so is $|\delta_{T^{(m)}}(S)|$. Furthermore, if $|\delta_{T^{(m)}}(S)| = 1$ then we can remove S from $T^{(m)}$ and still have a feasible solution, contradicting the inclusion-wise minimality of $T^{(m)}$. Hence $|\delta_{T^{(m)}}(S)| \geq 3$. So we have

  $$|\delta_T(S)| \geq |\delta_{T^{(m)}}(S)| \geq 3.$$

  On the other hand, since $S \subseteq A$, by property 5 Lemma 12 we have $|\delta_T(S)| = even$; together with above inequality we get $|\delta_T(S)| \geq 4$. Therefore, $y(\delta(S)) \geq 1$ in this case as well. ◀

Now the proof of Theorem 4 follows easily.

Proof of Theorem 4. Since $T^{(m)} \cup J$ is an Eulerian subgraph, there is a closed walk Q in it that spans X. By (18) we have $cost(T^{(m)}) \leq cost(x^*)$. By Lemma 13 $cost(J) \leq \frac{5}{8} \cdot opt$ and this implies $cost(T^{(m)} \cup J) \leq (1 + \frac{5}{8}) \cdot opt = \frac{13}{8} \cdot opt$, as desired.

By property 6 of Lemma 12, $d_{T^{(m)}}(v) \leq b'_v + 3 \leq \frac{b_v}{2} + 3$ and by Lemma 13 we have $d_J(v) \leq \frac{b_v}{2} + 3$. So $d_{T^{(m)} \cup J}(v) \leq b_v + 6$ for all $v \in V$, as desired. ◀

To prove Theorem 5, we modify Algorithm 2 slightly as follows: remove step (c), in step (d) compute a minimum cost $odd(T)$-join J (instead of $odd(T^{(m)})$-join) and in step (e) output a closed walk in $T \cup J$ that spans all the terminals. Similar to Lemma 13, we have the following bound on the bounded-degree $odd(T)$-join LP.

▶ **Lemma 14.** *There is an $odd(T)$-join J with cost at most $\frac{opt}{2}$ and $d_J(v) \leq \frac{b_v}{2} + 1$ for all $v \in V$.*

Proof. We just need to show $\frac{x^*}{2}$ is feasible for degree-bounded $odd(T)$-join polytope when the degree bound is $\frac{b_v}{2}$ for all $v \in V$. Then, whenever necessary we replace $\frac{b_v}{2}$ by $\frac{b_v}{2} + 1$ for $v \in V$ so that Theorem 8 to be applicable. It is trivial that the degree constraints hold. So we show constraints (5) in (**BD-Y-join LP**) holds. Let $S \subsetneq V$ such that $|S \cap odd(T)| = odd$.

- Case 1: If $S \cap X \neq \emptyset$. Then, $x^*(\delta(S)) \geq 2$ which implies $\frac{x^*}{2}(\delta(S)) \geq 1$.
- Case 2: If $S \cap X = \emptyset$ and $S \cap (V \setminus A) \neq \emptyset$. Then, there is a Steiner node $s \in S$ such that $s \notin A$. By definition of set A we have $x^*(\delta(s)) \geq 2$. By constraint (15) in (**BDSTSP-LP**) this implies $x^*(\delta(S)) \geq 2$ as well which implies $\frac{x^*}{2}(\delta(S)) \geq 1$.
- Case 3: If $S \subseteq A$. Thus, we have $S \subseteq A$ and note that $d_T(v)$ is even for all $v \in A$ by property 4 of Lemma 12. This is a contradiction because we assumed $|S \cap odd(T)|$ is odd. So the constraints for $S \subseteq A$ are not present in the LP. ◀

Proof of Theorem 5. By (18) we have $cost(T) \leq cost(x^*) \leq opt$ and by Lemma 14 we have $cost(J) \leq \frac{cost(x^*)}{2} \leq \frac{opt}{2}$ which implies $cost(T \cup J) \leq \frac{3}{2} \cdot opt$.

By properties 2 and 3 of Lemma 12, we have $d_T(v) \leq \frac{b_v}{2} + 7$ and by Lemma 14 we have $d_J(v) \leq \frac{b_v}{2} + 1$ for all $v \in V$. Thus, $d_{T \cup J}(v) \leq b_v + 8$, as desired. ◀

5 Conclusion

We gave a $(5/3, +4)$, a $(13/8, +6)$ and a $(3/2, +8)$ approximations for BDSTSP. It would be interesting to see if there is any $O(1)$-approximation that violates the degree bounds by at most $+2$. On the other hand, a demonstration that any LP-based $O(1)$-approximation requires a $+4$ violation would be interesting as well. Though, we suspect an $O(1)$-approximation with $+2$ violation is possible. Further improvements to our $\frac{13}{8}$-approximation for low violations (e.g. $+4$ or $+6$) would also be interesting. For example, perhaps a best-of-many Christofides approach could help to show that, on average, the Y-join can be constructed more cheaply than in our analysis. As a starting point, we note it is possible to extend the bounded-degree Steiner tree iterated rounding result to show that for feasible BDST solution x^*, $2 \cdot x^*$ dominates a convex combination of trees each of which violates degree bounds by $+3$.

Finally, we did not consider edge bounds b_e because if we even allow a $+1$ violation it becomes quite trivial: no BDSTSP solution will use any edge more than twice because we could remove two copies of any edge used more than twice. So what about satisfying edge bounds exactly? Obviously cut edges in the original graph that have Steiner nodes on both sides of the cut must be used twice. Even if there are no cut edges it is still hard to find a spanning Eulerian tour that uses each edge at most once: This would still model the Hamiltonian cycle problem in cubic graphs. Still, is there some interesting extent to which we could bound the use of some edges and still expect some approximation algorithm to respect most of these bounds?

References

1 Hyung-Chan An, Robert Kleinberg, and David B Shmoys. Improving christofides' algorithm for the st path tsp. *Journal of the ACM (JACM)*, 62(5):1–28, 2015.
2 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.
3 András Frank. On a theorem of mader. *Discret. Math.*, 101:49–57, 1992.
4 Takuro Fukunaga, Zeev Nutov, and R Ravi. Iterative rounding approximation algorithms for degree-bounded node-connectivity network design. *SIAM Journal on Computing*, 44(5):1202–1229, 2015.
5 Michel X Goemans. Minimum bounded degree spanning trees. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 273–282. IEEE, 2006.
6 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.
7 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for tsp. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.
8 Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. On some network design problems with degree constraints. *Journal of Computer and System Sciences*, 79(5):725–736, 2013.
9 Lap Chi Lau, Joseph Naor, Mohammad R Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM Journal on Computing*, 39(3):1062–1087, 2009.
10 Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.
11 Lap Chi Lau and Mohit Singh. Additive approximation for bounded degree survivable network design. *SIAM Journal on Computing*, 42(6):2217–2242, 2013.
12 Lap Chi Lau and Hong Zhou. A unified algorithm for degree bounded survivable network design. *Mathematical Programming*, 154(1):515–532, 2015.
13 Anand Louis and Nisheeth K Vishnoi. Improved algorithm for degree bounded survivable network design problem. In *Scandinavian Workshop on Algorithm Theory*, pages 408–419. Springer, 2010.
14 Wolfgang Mader. A reduction method for edge-connectivity in graphs. *Annals of discrete mathematics*, 3:145–164, 1978.
15 Ian Post and Chaitanya Swamy. Linear programming-based approximation algorithms for multi-vehicle minimum latency problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 512–531. SIAM, 2014.
16 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.
17 AI Serdyukov. O nekotorykh ekstremal'nykh obkhodakh v grafakh. *Upravlyayemyye sistemy*, 17:76–79, 1978.
18 Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *Journal of the ACM (JACM)*, 62(1):1–19, 2015.
19 Laurence A Wolsey. Heuristic analysis, linear programming and branch and bound. In *Combinatorial Optimization II*, pages 121–134. Springer, 1980.

A Proof of Lemma 11

Here we present the proof of Lemma 11.

Recall $\bar{x}$ is an optimal solution to (**SNDP-LP**) and s is a Steiner node such that $\bar{x}(\delta(s)) \leq 1$. First we show how to obtain x' and c' with an easy application of Mader's theorem but the running time of this procedure might be exponential and then at the end we show how to turn this procedure to run in polynomial time.

Since $\bar{x}$ is rational and the number bits needed to represent it is polynomial in the size of the input, there is a positive integer Δ such that $\Delta \cdot \bar{x}_e$ is an even integer for all $e \in E$. We replace each edge e, with $\Delta \cdot x_e$ copies of parallel edges. Note that degree of s is even and there is no cut-edge in the graph. By applying Mader's theorem repeatedly, we can split off s completely. Denote the resulting graph by $G' = (V \setminus \{s\}, E')$. Finally, we define a new solution x' for the resulting graph as follows: $x'_e = \frac{\#\ copies\ of\ e}{\Delta}$ for all $e \in E'$. Note that by construction, x' respects the degree bounds and we preserve the connectivity between each pairs of nodes (except s), hence x' is feasible for (**SNDP-LP**) corresponding to G'. Next, we define the cost function c'. It is defined naturally, i.e., for the existed edges in G, c and c' agree with each other and for a new introduced edge uv we set $c'_{uv} := c_{su} + c_{sv}$. Let $\mathcal{I}' := (G', X, c', b)$ be the resulting instance.

We show that $cost_{c'}(x') \leq cost_c(\bar{x})$. This follows from the fact that if we introduce a new edge uv, then its cost is $c_{su} + c_{sv}$ and we decrease the LP weight on su and sv by the same amount we increase the LP value on uv. If uv exists in G we claim that $c_{su} + c_{sv} > c_{uv}$ because otherwise one can increase the LP value on su and sv and decrease the LP value on uv by the same amount and get a cheaper feasible solution, contradicting the fact that $\bar{x}$ is optimal.

Finally, for any integral solution T' for $\mathcal{I}'$ we construct an integral solution T for $\mathcal{I}$ by replacing every edge uv in T' that is not in G with the two edges su and sv (keep the edges with multiplicities). Note we might have more than 2 copies of an edge e incident to s in T. In this case, we reduce the occurrences of e as much as possible so that the resulting solution is feasible for $\mathcal{I}$ AND the parity of the degree of the endpoints of e does not change. By definition of c' we have $cost_c(T) \leq cost_{c'}(T')$.

Here we show by a straightforward adaptation of techniques in [15], we can construct x' and c' that satisfy the property of the lemma efficiently.

Let $\bar{x}$ be the LP weight on the edges. Splitting off a pair (su, sv) to the extent of $\alpha > 0$ means reducing $\bar{x}_{sv}$ and $\bar{x}_{sv}$ by α, and increasing $\bar{x}_{uv}$ by α such that all the connectivities between any pair of nodes (except pairs involving s) are preserved (the connectivity between two nodes u and v is defined as the minimum weight $u - v$ cut when edges have weight according to $\bar{x}$). We say we split of (su, sv) to the *maximum extent* if value α above is the maximum value possible.

Note that by the first procedure we explained earlier, there exists always a pair (su, sv) and value $\alpha > 0$ such that we can split of (su, sv) to the extent of α. We can find such pairs by brute force ($O(n^2)$ pairs) and find α by binary search. Note that it is possible $u = v$ which in that case it means reducing x_{su} by α.

The algorithm is simple, we find a pair of edges (su, sv) and a value $\alpha > 0$ and split off (su, sv) to the extent of α and repeat this procedure.

In the next claim, we show if we split off a pair of edges to the maximum extent, that pair never becomes splittable again which in turn implies a polynomial running time of above procedure.

▷ Claim 15. Consider an splittable pair (su, sv) according to Mader theorem (Theorem 10). If we split off (su, sv) to the maximum extent, then (su, sv) will not become splittable again.

Proof of Claim 15. This follows from Claim 3.1 in [3] which states the following:

▷ Claim 16 (Claim 3.1 in [3]). A pair (su, sv) is splittable if and only if there is no set Y such that $u, v \in Y$, $s \notin Y$, and there are two nodes $w \in Y$, $s \neq z \notin Y$ such that $|\delta_G(Y)| \leq \lambda_G(w, z) + 1$.

Multiply $\bar{x}$ by a suitable integer Δ such that $\Delta \cdot \bar{x}_e$ is even for all $e \in E$. Replace each edge e by $\Delta \cdot \bar{x}_e$ many parallel edges in G.

Suppose we split as much copy of (su, sv) as possible. Then, there must be a set Y that satisfies the properties of Claim 16. Now assume we split of a pair of edges (e, f) incident to s and let G' be the resulting graph. We show that Y still satisfies the properties of Claim 16; hence, non of the copy of (su, sv) are splittable after splitting off (e, f).

Note $s \notin Y$ so $|\delta_{G'}(Y)| \leq |\delta_G(Y)|$. Furthermore, since (e, f) is a splittable pair we have $\lambda_{G'}(w, z) = \lambda_G(w, z)$. Therefore, $|\delta_{G'}(Y)| \leq |\delta_G(Y)| \leq \lambda_G(w, z) = \lambda_{G'}(w, z) + 1$. Hence, Y satisfies the properties of Claim 16 in G' as well so (e, f) is not splittable in G'. ◁

B Iterative Rounding Algorithm for BDSTP

Algorithm 3 Iterative rounding algorithm of [11] for BDTSP with small change in step (b).

Input: Graph $\big(G = (V, E), X, c, b\big)$, a subset of Steiner nodes A where $b_v = 1$ for all $v \in A$.
Output: A connected subgraph T of G that spans X such that $d_T(v) \leq b_v + 7$ is even for all $v \in A$ and $d_T(v) \leq b_v + 3$ for all $v \in V \setminus A$.

Initialize $T' \leftarrow \emptyset$ and $W \leftarrow V$ (W is the set of vertices with degree constraints present in the LP formulation).

while T' is not a feasible solution for BDSTP **do**

(a) Compute an optimal extreme point solution x of (**SNDP-LP**) and remove every edge e with $x_e = 0$.

(**b, with modification**). Removing a degree constraint: For every $v \in W$ with degree at most $b_v + 3$ in G, remove v from W. Furthermore, if $v \in A$, completely split off v and compute an optimal solution for the resulting instance (cf. Lemma 11). Redefine G to be this new graph and x to be the new optimal solution after splitting off procedure.

(c) Picking 1-edge: For each edge $e = uv$ with $x_e = 1$, add e to T, remove e from G, and decrease b_u, b_v by 1.

(d) Picking a heavy edge with no degree constraints: For each edge $e = uv$ with $x_e \geq \frac{1}{2}$ and $u, v \notin W$, add e to T' and remove e from G.

Updating the connectivity requirements: For every set $S \neq X$, $S \cap X \neq \emptyset$ update the RHS of constraint (2) in (**SNDP-LP**) to be $\max\{1 - |\delta'_T(S)|, 0\}$.

Let T be the resulting Steiner tree obtained from T' by applying Lemma 11 repeatedly.

Note that the algorithm works correctly for any subset A of Steiner nodes (i.e., A does not need to contain only Steiner nodes v with $b_v = 1$). The performance guarantee of this algorithm follows from [11] and the fact that after splitting off a Steiner node, by Lemma 11 we still have a feasible solution for the (**SNDP-LP**) of the resulting instance with cost at most the cost of the original LP. So T' is a Steiner tree for the instance $\big(G' = (V \setminus A, E'), X, c', b\big)$. The fact that the Steiner tree T for the original instance obtained from T' has the desired properties of Lemma 12 follows from Lemma 11.

C Iterative Rounding Algorithm for BD Y-join

In this section, we prove Theorem 8 using an iterative rounding algorithm. The algorithm and its analysis are a simple adaptation of the iterative rounding algorithm for maximum matching in [10].

Algorithm 4 An iterative algorithm for bounded-degree Y-join problem.

Input: Undirected graph $G = (V, E)$ with edge costs $c_e \geq 0, e \in E$, degree bounds $b_v, v \in V$, and $Y \subseteq V$ where $|Y| = even$.
Output: A Y-join.

$J \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
Find an optimal extreme point solution x to **BD-Y-join LP** defined on $G = (V, E)$.
Find an edge $e = uv$ with either $x_e = 0$ or $x_e = 1$ (cf. Lemma 18). In the former case remove e, and in the latter case add e to J and do the following:
Update $Y \leftarrow Y \Delta \{u, v\}$. Update $E \leftarrow E \setminus \{e\}$. Update $b(v) \leftarrow b(v) - 1$, and $b(u) \leftarrow b(u) - 1$.
output J.

We only need to show that there is always an edge with LP value 0 or 1. This shows the LP is integral. To do this, we state some properties of an extreme point solution to **BD-Y-join LP** which follows from rank lemma (Lemma 2.1.4 in [10]) and standard uncrossing techniques.

▶ **Lemma 17** (Properties of an extreme point). *Consider an extreme point x and suppose $0 < x_e < 1$ for all $e \in E$. Then, there exists a laminar family $\mathcal{L}$ of Y-odd sets and $W \subseteq V$ such that*

(i) *For any $S \in \mathcal{L}$ we have $x(\delta(S)) = 1$ and for any $v \in W$ we have $x(\delta(v)) = b(v)$.*
(ii) *The vectors in $\{\chi(\delta(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in W\}$ are linearly independent.*
(iii) $|\mathcal{L}| + |W| = |E|$.
(iv) *$E[S]$ is connected for each $S \in \mathcal{L}$.*

Proof. Properties (i), (ii), and (iii) follow from a standard application of uncrossing technique and Rank Lemma in polyhedral theory.

We show that one can further modify the laminar family to obtain (iv). Consider $S \in \mathcal{L}$ and suppose $E[S]$ is not connected. Since $|S \cap Y| = odd$, there must be a connected component C of $E[S]$ such that $|C \cap Y| = odd$. Since $x(\delta(C)) \geq 1$ and the fact that S is a tight set we have $\chi(\delta(C)) = \chi(\delta(S))$ and C is a tight set. By property (ii) $C \notin \mathcal{L}$. Now consider the laminar family $(\mathcal{L} \setminus \{S\}) \cup \{C\}$. Repeating this procedure until there is no set in the laminar family that violates (iv) finishes the proof. ◀

▶ **Lemma 18.** *Given any extreme point x of* **BD-Y-join LP** *there must exist an edge e with $x_e = 0$ or $x_e = 1$.*

Proof. Suppose not. So we have $0 < x_e < 1$ for each $e \in E$. Let $\mathcal{L}$ be a laminar family and let W be a subset of V that satisfy properties (i)-(iv) in Lemma 17. We show a contradiction with property (iii) using a token-based argument. Let $\mathcal{L}' := \mathcal{L} \cup W$ be the extended laminar family. We assign one token to each edge in the support of x. Then we distribute the tokens inductively among the sets in the laminar family such that each member of $\mathcal{L}'$ receives at least one token and we show there are some extra tokens left which shows the contradiction with the fact that $|E| = |\mathcal{L}'|$.

We use the natural forest structure that the laminar family $\mathcal{L}'$ imposes, recall that each component of this forest is a rooted tree. We use the following claim to redistribute the tokens of $E[S]$ among the laminar sets inside S. We use G_S to denote the graph after contracting the children of S in $G[S]$. We label the contracted vertices with the same label as the contracted set, i.e., if R_i is a child of S then we call the contracted vertex of R_i by R_i as well.

▷ Claim 19. For any rooted subtree of $\mathcal{L}'$ with root S, the tokens of edges in $E[S]$ can be distributed such that for every $S' \in \mathcal{L}'$ where $S' \subsetneq S$ receives at least one token.

Proof. We prove this by induction. If $S \in W$ the claim holds vacuously. So let $S \in \mathcal{L}$ and let $R_1, ..., R_k$ be the children of S. We call a child of S *good* if a token is already (before considering S) assigned to it and call it *bad* otherwise. Note that none of the tokens in $E(G_S)$ is assigned to any sets yet because of the way the inductive procedure works.

By property (iv) of Lemma 17, G_S is a connected graph and if it is not a tree then $E(G_S) \geq k$ and we can assign the tokens of these (at least) k edges in $E(G_S)$ to $R_1, ..., R_k$ and we are done. So suppose G_S is a tree. If one of $R_1, ..., R_k$ is a good child then again we have enough tokens in $E(G_S)$ to assign to bad children of S. So from now on we assume all the children of S are bad.

Since $|S \cap Y| = odd$ and the fact that if $x(\delta(R_i)) = odd$ then $|R_i \cap Y| = odd$, we conclude the number of children of S that have an odd fractional degree is odd. Therefore, $\sum_{i=1}^{k} x(\delta(R_i)) = odd$.

Since G_S is a tree so it is a bipartite graph. Let V_1 and V_2 be the two parts of G_S and since $\sum_{i=1}^{k} x(\delta(R_i)) = odd$, wlog, we can assume $\sum_{R_i \in V_1} x(\delta(R_i)) \leq \frac{\sum_{i=1}^{k} x(\delta(R_i))-1}{2}$. From this inequality and the fact that $x(\delta(S)) = 1$ we get

$$\frac{\sum_{i=1}^{k} x(\delta(R_i)) - 1}{2} = x(E(G_S)) \leq \sum_{R_i \in V_1} x(\delta(R_i)) \leq \frac{\sum_{i=1}^{k} x(\delta(R_i)) - 1}{2}. \tag{20}$$

So all the inequalities in (20) must hold as equality. Therefore, $x(E(G_S)) = \frac{\sum_{i=1}^{k} x(\delta(R_i))-1}{2}$ which implies $\chi(\delta(S)) = \sum_{R_i \in V_2} \chi(\delta(R_i)) - \sum_{R_i \in V_1} \chi(\delta(R_i))$ and this contradicts the linear independence of the characteristic vectors in $\mathcal{L}'$. This finishes the proof of the claim. ◁

We continue the proof of Lemma 18. By Claim 19 every non-root vertices of the forest (obtained from $\mathcal{L}'$) is assigned a token. Let $S_1, ..., S_k$ be the root nodes of the forest. Denote by G_R the graph obtained from contracting S_i's. Note that none of the tokens of edges in $E(G_R)$ is assigned to any member of $\mathcal{L}'$ by our token assignment given in Claim 19. Since $x_e < 1$, $|\delta(S_i)| \geq 2$. We show that at least one root node has degree at least 3. But first let us show if this holds then the lemma follows. Since one non-root node has degree at least 3, we have $|E(G_R)| \geq k + 1$ which implies we can assign one token to each root node and still have at least one token left unassigned. This implies $|E(G)| > |\mathcal{L}'|$, a contradiction with property (iii) of Lemma 17. Therefore, there must be an edge e with $x_e = 0$ or $x_e = 1$.

Now it remains to prove at least one root node has degree at least 3. Suppose not. Then, every vertex in G_R has degree 2 and so G_R is a collection of cycles. Consider a cycle component C of G_R. Note $|C|$ cannot be odd otherwise there should be at lest fractionally one edge going outside of the nodes in C. So let $S_1, ..., S_l$ be the vertices of C and note l is even. Then we have $\sum_{i=1}^{l} (-1)^i \chi(\delta(S_i)) = 0$ which contradicts the linear independence of characteristic vectors in $\mathcal{L}'$.

We finish the proof by showing G_R itself cannot be a cycle. Note $x(\delta(S_i)) = 1$ for $1 \leq i \leq k$ which implies $|S_i \cap Y| = odd$ and so the number of vertices in G_R is even. And if G_R forms a cycle, then the same argument as the previous paragraph shows a linear dependence of characteristic vectors in $\mathcal{L}'$. ◀

Budgeted Out-Tree Maximization with Submodular Prizes

Gianlorenzo D'Angelo
Gran Sasso Science Institute, L'Aquila, Italy

Esmaeil Delfaraz
Gran Sasso Science Institute, L'Aquila, Italy

Hugo Gilbert
Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016 Paris, France

Abstract

We consider a variant of the prize collecting Steiner tree problem in which we are given a *directed graph* $D = (V, A)$, a monotone submodular prize function $p : 2^V \to \mathbb{R}^+ \cup \{0\}$, a cost function $c : V \to \mathbb{Z}^+$, a root vertex $r \in V$, and a budget B. The aim is to find an out-subtree T of D rooted at r that costs at most B and maximizes the prize function. We call this problem *Directed Rooted Submodular Tree* (**DRST**).

For the case of undirected graphs and additive prize functions, Moss and Rabani [SIAM J. Comput. 2007] gave an algorithm that guarantees an $O(\log |V|)$-approximation factor if a violation by a factor 2 of the budget constraint is allowed. Bateni et al. [SIAM J. Comput. 2018] improved the budget violation factor to $1 + \varepsilon$ at the cost of an additional approximation factor of $O(1/\varepsilon^2)$, for any $\varepsilon \in (0, 1]$. For directed graphs, Ghuge and Nagarajan [SODA 2020] gave an optimal quasi-polynomial time $O\left(\frac{\log n'}{\log \log n'}\right)$-approximation algorithm, where n' is the number of vertices in an optimal solution, for the case in which the costs are associated to the edges.

In this paper, we give a polynomial time algorithm for **DRST** that guarantees an approximation factor of $O(\sqrt{B}/\varepsilon^3)$ at the cost of a budget violation of a factor $1 + \varepsilon$, for any $\varepsilon \in (0, 1]$. The same result holds for the edge-cost case, to the best of our knowledge this is the first polynomial time approximation algorithm for this case. We further show that the unrooted version of **DRST** can be approximated to a factor of $O(\sqrt{B})$ without budget violation, which is an improvement over the factor $O(\Delta\sqrt{B})$ given in [Kuo et al. IEEE/ACM Trans. Netw. 2015] for the undirected and unrooted case, where Δ is the maximum degree of the graph. Finally, we provide some new/improved approximation bounds for several related problems, including the additive-prize version of **DRST**, the maximum budgeted connected set cover problem, and the budgeted sensor cover problem.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Prize Collecting Steiner Tree, Directed graphs, Approximation Algorithms, Budgeted Problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.9

Related Version *Full Version*: `https://arxiv.org/abs/2204.12162`

Funding *Gianlorenzo D'Angelo*: Partially supported by the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets".

1 Introduction

Prize collecting Steiner tree problems (**PCSTP**) have been extensively studied due to their applications in designing computer and telecommunication networks, VLSI design, computational geometry, wireless mesh networks, and cancer genome studies [5, 8, 15, 23, 32]. Very interesting polynomial-time constant/poly-logarithmic approximation algorithms have been proposed for many variants of **PCSTP** when the graph is undirected [1, 2, 10, 12, 18, 21, 29]. However, these problems are usually much harder on directed graphs. For instance,

there is a simple polynomial-time 2-approximation algorithm for the *undirected Steiner tree* problem, but no quasi-polynomial-time algorithm for the *directed Steiner tree* problem achieving an approximation ratio of $o\left(\frac{\log^2 k}{\log\log k}\right)$ exists, unless $NP \subseteq \bigcap_{0<\varepsilon<1} \text{ZPTIME}(2^{n^\varepsilon})$ or the Projection Game Conjecture is false [13], where k is the number of terminal nodes.

Some of the most relevant variants of **PCSTP** are represented by prize collecting problems with budget constraints. In such problems, we are usually given a graph with prizes and costs on the nodes and the goal is to find a tree that maximizes the sum of the prize of its nodes, while keeping the total cost bounded by a given budget. Guha et al. [14] introduced the case in which the graph is undirected and the goal is to find a tree that contains a distinguished vertex, called root, respects the budget constraint, and maximizes the prize, we call this problem *Undirected Rooted Additive Tree* (**URAT**). They gave an algorithm that achieves an $O(\log^2 n)$-approximation factor, where n is the number of nodes in the graph, but the computed solution requires a factor-2 violation of the budget constraint. Moss and Rabani [27] and Bateni et al. [2] further investigated **URAT** and improved the results from the approximability point of view. The former paper improved the approximation factor to $O(\log n)$, with the same budget violation, and the latter one improved the budget violation factor to $1+\varepsilon$ to obtain an approximation factor of $O\left(\frac{1}{\varepsilon^2}\log n\right)$, for any $\varepsilon \in (0,1]$. Kortsarz and Nutov [22] showed that the unrooted version of **URAT**, so does **URAT**, admits no $o(\log\log n)$-approximation algorithm, unless $NP \subseteq DTIME(n^{\text{polylog}(n)})$, even if the algorithm is allowed to violate the budget constraint by a factor equal to a universal constant.

In this paper, we consider a generalization of **URAT** on directed graphs. We are given a directed graph, where each node is associated with a cost, and the prize is defined by a monotone submodular function on the subsets of nodes, and the goal is to find an *out-tree* (a.k.a. *out-arborescence*) rooted at a specific vertex r with the maximum prize such that the total cost of all vertices in the out-tree is no more than a given budget. We term this problem *Directed Rooted Submodular Tree* (**DRST**). A closely related problem, called *Submodular Tree Orienteering* (**STO**), has been recently introduced by Ghuge and Nagarajan [11]. **STO** is the same problem as **DRST** except that edges and not nodes have costs. They provided a tight quasi-polynomial-time $O(\frac{\log n'}{\log\log n'})$-approximation algorithm that requires $(n\log B)^{O(\log^{1+\epsilon} n')}$ time, where n' is the number of vertices in an optimal solution and B is the budget constraint.

Contribution. By extending some ideas of Kuo et al. [23] and Bateni et al. [2], we design a polynomial-time $O(\sqrt{B}/\varepsilon^3)$-approximation algorithm for **DRST**, violating the budget constraint B by a factor of at most $1+\varepsilon$, for any $\varepsilon \in (0,1]$ (Section 4). Our technique can be used to obtain the same result for **STO** (Section 6). To our knowledge, this is the first polynomial-time approximation algorithm for **STO**. We also show that, for any $1+\varepsilon$ budget violation, with $\varepsilon \in (0,1]$, our approach provides an $O(\sqrt{B}/\varepsilon^2)$-approximation algorithm for the special cases of **DRST** and **STO** where the prize function is additive (Section 7). We also consider the unrooted version of **DRST** and give an $O(\sqrt{B})$-approximation algorithm without budget violation (Section 5), which is an improvement over the factor $O(\Delta\sqrt{B})$ [23] for the undirected and unrooted version of **DRST**, where Δ is the maximum node-degree.

Finally, we study some variants of **DRST** on undirected graphs. We show that, for any $1+\varepsilon$ budget violation, **URAT** admits an $O(\Delta/\varepsilon^2)$-approximation algorithm, while its quota version admits a 2Δ-approximation algorithm. Next, we present some approximation results for some variants of the connected maximum coverage problem, which improve over the bounds given by Ran et al. [30]. Finally, we provide two approximation algorithms for the Budgeted Sensor Cover problem, which result in an improvement to the literature [23, 30, 33, 34]. We discuss these results in Section 7.

Related Work. Many variants of Prize collecting Steiner Tree problems have been investigated. Here we list those that are more closely related to our study. Further related work is reported in the Appendix.

Kuo et al. [23] studied the unrooted version of **DRST** on undirected graphs called *Maximum Connected Submodular function with Budget constraint* (**MCSB**). They provided an $O(\Delta\sqrt{B})$-approximation algorithm for **MCSB**, where Δ is the maximum degree of the graph. Vandin et al. [32] provided a $(\frac{2e-1}{e-1}r)$-approximation algorithm for a special case of the same problem, where r is the radius of an optimal solution. This problem coincides with the connected maximum coverage problem in which each set has cost one. Ran et al. [30] presented an $O(\Delta \log n)$-approximation algorithm for a special case of the connected maximum coverage problem. Hochbaum and Rao [15] investigated **MCSB** in which each vertex costs 1 and provided an approximation algorithm with factor $\min\{1/((1-1/e)(1/R-1/B)), B\}$, where R is the radius of the graph. Chen et al. [4] investigated the edge-cost version of **MCSB**. One of the applications of **MCSB** is a problem in wireless sensor networks called *Budgeted Sensor Cover Problem* (**BSCP**), where the goal is to find a set of B connected sensors to maximize the number of covered users, for a given B. Kuo et al. [23] provided a $5(\sqrt{B}+1)/(1-1/e)$-approximation algorithm for **BSCP**, which was improved by Xu et al. [33] to $\lfloor\sqrt{B}\rfloor/(1-1/e)$. Huang et al. [17] proposed a $8(\lceil 2\sqrt{2C}\rceil+1)^2/(1-1/e)$-approximation algorithm for **BSCP**, where $C = O(1)$.

Johnson et al. [18] introduced an edge-cost variant of **DRST** on undirected graphs, where the prize function is additive, called **E-URAT**. They showed that there exists a $(5+\varepsilon)$-approximation algorithm for the unrooted version of **E-URAT** using Garg's 3-approximation algorithm [9] for the k-MST problem, and observed that a 2-approximation for k-MST would lead to a 3-approximation for **E-URAT**. This observation along with the Garg's 2-approximation algorithm [10] for k-MST yield a 3-approximation algorithm for the unrooted version of **E-URAT**. Recently, Paul et al. [29] provided a polynomial-time 2-approximation algorithm for **E-URAT**.

2 Notation and problem statement

For an integer k, let $[k] := \{1, \ldots, k\}$. A directed *path* is a directed graph made of a sequence of distinct vertices $(v_1, \ldots, v_k)$ and a sequence of directed edges (v_i, v_{i+1}), $i \in [k-1]$. An *out-tree* (a.k.a. out-arborescence) is a directed graph in which there is exactly one directed path from a specific vertex r, called *root*, to each other vertex. If a subgraph T of a directed graph D is an out-tree, then we say that T is an out-tree of D.

Let $D = (V, A)$ be a directed graph with n nodes, $c : V \to \mathbb{Z}^+$ be a cost function on nodes, $p : 2^V \to \mathbb{R}^+ \cup \{0\}$ be a monotone submodular prize function on the subsets of nodes, $r \in V$ be a root vertex, and B be an integer budget. For any subgraph D' of D, we denote by $V(D')$ and $A(D')$ the set of nodes and edges in D', respectively. Given $S \subseteq V$, we denote the cost of S by $c(S) = \sum_{v\in S} c(v)$ and we use shortcuts $c(D') = c(V(D'))$ and $p(D') = p(V(D'))$ for a subgraph D' of D. In the Directed Rooted Submodular Tree problem (**DRST**), the goal is to find an out-tree T of D rooted at r such that $c(T) \le B$ and $p(T)$ is maximum. Throughout the paper, we denote an optimal solution to **DRST** by T^*.

Given two nodes u and v in V, a path in D from u to v with the minimum cost is called a *shortest path* and its cost, denoted by $dist(u, v)$, is called the *distance* from u to v in D.

An algorithm is a bicriteria (β, α)-approximation algorithm for **DRST** if, for any instance I of the problem, it returns a solution Sol_I such that $p(Sol_I) \ge \frac{OPT_I}{\alpha}$ and $c(Sol_I) \le \beta B$, where OPT_I is the optimum for I.

3 Results and Techniques

Our main result is given in the next theorem.

▶ **Theorem 1.** ***DRST*** *admits a polynomial-time bicriteria* $\left(1+\varepsilon, O\left(\frac{\sqrt{B}}{\varepsilon^3}\right)\right)$*-approximation algorithm, for any* $\varepsilon \in (0,1]$.

Our approach combines and extends techniques given by Kuo et al. [23] and Bateni et al. [2]. To illustrate our techniques, we now consider the case in which costs are unitary, i.e. $c(v) = 1$, for each $v \in V$, and the prize function is additive, i.e. $p(S) = \sum_{v \in S} p(\{v\})$, for any $S \subseteq V$. In this case, the distance from a node u to a node v is equal to the minimum number of nodes in a path from u to v and the cost of a tree T is equal to its size, $c(T) = |V(T)|$. W.lo.g. we also assume that the distance from r to any node is at most B. We will give the proof for the general case in Section 4.

The algorithm works as follows. For any vertex u, we denote as V_u the set of all nodes that are at a distance no more than $\lfloor\sqrt{B}\rfloor$ from u, $V_u := \{v \mid dist(u,v) \leq \lfloor\sqrt{B}\rfloor\}$. We first select a subset S_u of V_u of at most $\lfloor\sqrt{B}\rfloor$ nodes with the maximum prize, $S_u := \arg\max\{p(S) \ : \ S \subseteq V_u, |S| \leq \lfloor\sqrt{B}\rfloor\}$.[1] We then compute a minimal inclusion-wise out-tree T_u rooted at u that spans all nodes in S_u. Note that $|V(T_u)| \leq B$ since the distance from u to any node in S_u is at most $\lfloor\sqrt{B}\rfloor$. Let z be a node such that $p(T_z)$ is maximum. If $z = r$, then we take T_z as our solution, otherwise we compute a solution by adding to T_z a shortest path P from r to z and removing the edges in $A(T_z) \setminus A(P)$ incoming the nodes in $V(T_z) \cap V(P)$. Let T be our solution and T^* be an optimal solution.

We will prove (Lemma 6) that any out-tree $\hat{T}$ can be covered by at most $N = O(|\hat{T}|/m)$ out-subtrees $\{\hat{T}_i\}_{i=1}^N$ with at most m nodes each, where m is any positive integer less than $|\hat{T}|$. By applying this claim to an optimal solution T^* and by setting $m = \lfloor\sqrt{B}\rfloor$, we obtain

$$p(T^*) = p\left(\bigcup_{i=1}^{N} V(T_i^*)\right) \leq N p(T'),$$

where $p(T') = \max\{p(T_i^*) \mid i \in [N]\}$, $|T'| \leq \lfloor\sqrt{B}\rfloor$, and $N = O(|T^*|/m) = O(\sqrt{B})$. Let w be the root of T'. Recall that S_w is a set of at most $\lfloor\sqrt{B}\rfloor$ nodes that are at a distance no more than $\lfloor\sqrt{B}\rfloor$ from w and have the maximum prize and T_w contains all the nodes in S_w. Since $|T'| \leq \lfloor\sqrt{B}\rfloor$, we have

$$p(T') \leq p(S_w) \leq p(T_w) \leq p(T_z) \leq p(T).$$

Since $N = O(\sqrt{B})$, we conclude that $p(T^*) = O(\sqrt{B})p(T)$.

Note that the cost of T is upper-bounded by $2B$, as both the cost of T_z and that of a shortest path from r to z are at most B. We can use the trimming procedure introduced by Bateni et al. [2] to obtain an out-subtree of T with cost at most $(1+\varepsilon)B$ by loosing an approximation factor of $O(1/\varepsilon^2)$, for any $\varepsilon \in (0,1]$ (see Lemma 2). This shows Theorem 1 for the unit-cost, additive-prize case. In the case in which the prize is a general monotone submodular function, the trimming procedure by Bateni et al. cannot be applied. We show how to generalize this procedure to the case of any monotone submodular prize function by loosing an extra approximation factor of $O(1/\varepsilon)$.

[1] This step can be done in polynomial time since function p is additive. If p is monotone and submodular, this step consists in solving the submodular maximization problem. See Section 4 for more details.

We can use the same approach to obtain a polynomial-time bicriteria $\left(1+\varepsilon, O\left(\frac{\sqrt{B}}{\varepsilon^2}\right)\right)$-approximation algorithm for the case of additive prize function and edge-cost. More importantly, we can obtain a polynomial-time bicriteria $\left(1+\varepsilon, O\left(\frac{\sqrt{B}}{\varepsilon^3}\right)\right)$-approximation algorithm for **STO**, i.e. for the edge-cost case where the prize function is monotone submodular. To the best of our knowledge, this is the first polynomial-time approximation algorithm for **STO**.

Finally, for the unrooted version the same approach with some minor changes achieves an $O(\sqrt{B})$-approximation with no budget violation.

4 Approximation Algorithm for DRST

We now introduce our polynomial-time approximation algorithm for **DRST**. We start by defining a procedure that takes as input an out-tree of a directed graph D and returns another out-tree of D which has a smaller cost but preserves the same prize-to-cost ratio (up to a bounded multiplicative factor).

Bateni et al. [2] introduced a similar procedure for the case of undirected graphs and additive prize function. In their case, we are given an undirected graph $G = (V, E)$, a distinguished vertex $r \in V$ and a budget B, where each vertex $v \in V$ is assigned with a prize $p'(v)$ and a cost $c'(v)$. For a tree T, the prize and cost of T are the sum of the prizes and costs of the nodes of T and are denoted by $p'(T)$ and $c'(T)$, respectively. A graph G is called *B-proper* for the vertex r if the cost of reaching any vertex from r is at most B. Bateni et al. proposed a trimming process that leads to the following lemma.

▶ **Lemma 2** (Lemma 3 in [2]). *Let T be a tree rooted at r with the prize-to-cost ratio $\gamma = \frac{p'(T)}{c'(T)}$. Suppose the underlying graph is B-proper for r and for $\varepsilon \in (0,1]$ the cost of the tree is at least $\frac{\varepsilon B}{2}$. One can find a tree T' containing r with the prize-to-cost ratio at least $\frac{\varepsilon\gamma}{4}$ such that $\varepsilon B/2 \le c'(T') \le (1+\varepsilon)B$.*

We now generalize this trimming process to the case in which the underlying graph is directed and the prize function is monotone and submodular by borrowing ideas from [2].

We introduce some additional definitions. Let T be an out-tree rooted at r. A *full* out-subtree of T rooted at some node v is an out-subtree of T containing all the vertices that are reachable from r through v in T. The set of *strict* out-subtrees of T is the set of all full out-subtrees of T other than T itself. The set of *immediate* out-subtrees of T is the set of all full out-subtrees rooted at the children of r in T. A directed graph $D = (V, A)$ is *B-appropriate* for a node r if $dist(r, v) \le B$ for any node $v \in V$.

▶ **Lemma 3.** *Let $D = (V, A)$ be a B-appropriate graph for a node r. Let T be an out-tree of D rooted at r with the prize-to-cost ratio $\gamma = \frac{p(T)}{c(T)}$, where p is a monotone submodular function. Suppose that $\frac{\varepsilon B}{2} \le c(T) \le hB$, where $h \in (1, n]$ and $\varepsilon \in (0,1]$. One can find an out-subtree $\hat{T}$ rooted at r with the prize-to-cost ratio at least $\frac{\varepsilon^2\gamma}{32h}$ such that $\varepsilon B/2 \le c(\hat{T}) \le (1+\varepsilon)B$.*

Proof. We run the following initial trimming procedure. We iteratively remove a strict out-subtree T' from T that satisfies two conditions: (i) the prize-to-cost ratio of $T \setminus T'$ is at least γ, and (ii) $c(T \setminus T') \ge \frac{\varepsilon}{2}B$. We repeat this process until no such strict out-subtree exists. Let T_- be the remaining out-tree after applying this process on T.

Now if $c(T_-) \le (1+\varepsilon)B$, the desired out-subtree is obtained. Suppose it is not the case. A full out-subtree T' is called *rich* if $c(T') \ge \frac{\varepsilon}{2}B$ and the prize-to-cost ratio of T' and all its strict out-subtrees are at least γ. We claim that if there exists a rich out-subtree, then we can find the desired out-subtree $\hat{T}$.

▷ Claim 4. Given a rich out-subtree T', the desired out-subtree $\hat{T}$ can be found.

Proof. We first find a rich out-subtree T'' of T' such that the strict out-subtrees of T'' are not rich, i.e., $c(T'') \geq \frac{\varepsilon}{2}B$ while the cost of any strict out-subtree of T'' (if any exist) is less than $\frac{\varepsilon}{2}B$. Let C be the total cost of the immediate out-subtrees of T''. We distinguish between two cases:

1. If $C < \frac{\varepsilon}{2}B$, then let $\hat{T}$ be the union of T'' and a shortest path P from r to the root r'' of T''. $\hat{T}$ has cost at most $C + B \leq (1+\varepsilon)B$ and prize at least $\gamma(\frac{\varepsilon}{2}B)$. This implies that $\hat{T}$ has ratio at least $\frac{\gamma\varepsilon}{2(1+\varepsilon)} \geq \frac{\gamma\varepsilon}{4} \geq \frac{\gamma\varepsilon^2}{32h}$.
2. If $C \geq \frac{\varepsilon}{2}B$, we proceed as follows. Since each immediate out-subtree of T'' has a cost strictly smaller than $\frac{\varepsilon}{2}B$, we can partition all the immediate out-subtrees of T'' into M groups $S_1, \ldots, S_M$ in such a way that for each $i \in [M-1]$ the total cost of immediate out-subtrees in S_i is at least $\frac{\varepsilon}{2}B$, and for each $i \in [M]$ it is at most εB. We can always group in this way since the cost of each immediate out-subtree of T'' is less than $\frac{\varepsilon}{2}B$ while $C \geq \frac{\varepsilon}{2}B$. Since the total cost of all the immediate out-subtrees of T'' is upper bounded by hB, then the number of selected groups M is at most

$$M \leq \left\lceil \frac{hB}{\frac{\varepsilon}{2}B} \right\rceil = \left\lceil \frac{2h}{\varepsilon} \right\rceil \leq \left\lfloor \frac{2h}{\varepsilon} \right\rfloor + 1 \leq \left\lfloor \frac{4h}{\varepsilon} \right\rfloor \leq \frac{4h}{\varepsilon}.$$

We now add the root r'' of T'' to each group S_i and denote the new group by S'_i, i.e., $S'_i = S_i \cup \{r''\}$, for any $i \in [M]$. By the monotonicity and submodularity of p, we have $\sum_{i=1}^{M} p(S'_i) \geq p(S'_1) + \sum_{i=2}^{M} p(S_i) \geq p(S'_1 \cup \bigcup_{i=2}^{M} S_i) = p(T'')$. Now among $S'_1, \ldots, S'_M$, we select the group S'_z that maximizes the prize, i.e., $z = \arg\max_{i \in [M]} p(S'_i)$. We know that

$$p(S'_z) \geq \frac{1}{M}\sum_{i=1}^{M} p(S'_i) \geq \frac{p(T'')}{M} \geq \frac{\varepsilon}{4h}p(T'') \geq \frac{\varepsilon}{4h} \cdot \frac{\gamma\varepsilon}{2}B = \frac{\gamma\varepsilon^2}{8h}B.$$

In case $z = M$ and $c(S'_M) < \frac{\varepsilon}{2}B$, we select a subset of immediate out-subtrees from $\bigcup_{i=1}^{M-1} S_i$ with the total cost of at least $\frac{\varepsilon}{2}B$ and at most $\varepsilon B - c(S'_M)$, and add it to S'_z.
Finally, let $\hat{T}$ be the union of a shortest path P from r to r'', S'_z, and the edges from r'' to the roots of the out-subtrees in S_z (see Figure 1). By monotonicity, $\hat{T}$ has the total prize at least $p(\hat{T}) \geq p(S'_z) \geq \frac{\gamma\varepsilon^2}{8h}B$. Note that $c(\hat{T}) \leq (1+\varepsilon)B$ as $c(S'_z \setminus \{r''\}) = c(S_z) \leq \varepsilon B$ and the shortest path from r to r'' costs at most B (since the graph is B-appropriate). This implies that the prize-to-cost ratio of $\hat{T}$ is at least $\frac{\gamma\varepsilon^2}{8h(1+\varepsilon)} \geq \frac{\gamma\varepsilon^2}{16h} \geq \frac{\gamma\varepsilon^2}{32h}$. ◁

It only remains to consider the case when there is no rich out-subtree. Since T_- is not rich and $c(T_-) \geq \frac{\varepsilon}{2}B$, the ratio of at least one strict out-subtree of T_- is less than γ. Now we find a strict out-subtree T' with ratio less than γ such that the ratio of all of its strict out-subtrees (if any exist) is at least γ. We first need to show that $c(T_- \setminus T') < \frac{\varepsilon}{2}B$.

▷ Claim 5. $c(T_- \setminus T') < \frac{\varepsilon}{2}B$.

Proof. By the submodularity of p, we know that $p(T_- \setminus T') + p(T') \geq p(T_-)$. This implies that

$$\frac{p(T_- \setminus T')}{c(T_- \setminus T')} \geq \frac{p(T_-) - p(T')}{c(T_-) - c(T')}. \tag{1}$$

Let $\gamma' = \frac{p(T')}{c(T')}$ be the prize-to-cost ratio of T'. We know that

$$p(T_-) - p(T') = c(T_-)\gamma - c(T')\gamma' > c(T_-)\gamma - c(T')\gamma, \tag{2}$$

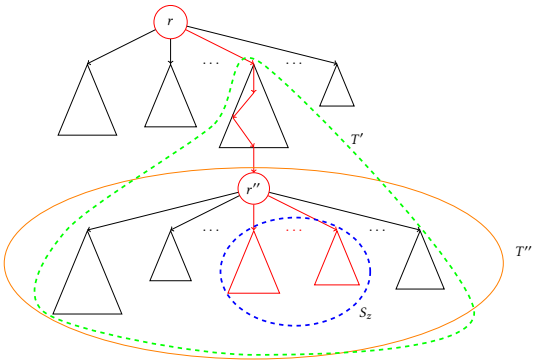


Figure 1 T_- is rooted at r, which is the whole out-tree. The green dashed closed curve represents the rich out-subtree T'. The orange circle represents T'' rooted at r'', where its strict out-subtrees are not rich, i.e., the cost of any strict out-subtree of T'' is less than $\frac{\varepsilon}{2}B$. The blue dashed circle represents the partition S_z, which maximizes the prize and costs at most εB. The red out-subtree represents $\hat{T}$, which is the union of a shortest path from r to r'', S_z and the edges from r'' to the immediate out-subtrees of T'' in S_z. Note that for the sake of simplicity, in this figure we suppose that the shortest path from r to r'' is included in T_-.

where the inequality holds because $\gamma' < \gamma$. By Equations (1) and (2), we have $\frac{p(T_- \setminus T')}{c(T_- \setminus T')} > \gamma$. As the prize-to-cost ratio of $T_- \setminus T'$ is more than γ but T' has not been removed from T during the initial phase, then $c(T_- \setminus T') < \frac{\varepsilon}{2}B$. This concludes the proof of the claim. ◁

We know that $c(T_-) > (1+\varepsilon)B$ and the cost from r to the root of T' is at most B. Then by Claim 5, the total cost of immediate out-subtrees of T' is at least $\frac{\varepsilon}{2}B$. Also, the cost of an immediate out-subtree of T' is less than $\frac{\varepsilon}{2}B$, otherwise, we have a rich out-subtree. As the ratio and cost of T_- are at least γ and $\frac{\varepsilon}{2}B$, respectively, then $p(T_-) \geq \frac{\gamma\varepsilon}{2}B$. Now we distinguish between two cases:

1. If $p(T') \geq \frac{\gamma\varepsilon}{4}B$, by similar reasoning as above, we group the immediate out-subtrees of T' into M groups $S_1, \ldots, S_M$ in such a way that for each $i \in [M-1]$ the total cost of immediate out-subtrees in S_i is at least $\frac{\varepsilon}{2}B$, and for each $i \in [M]$ it is at most εB. Now define a new group $S'_i = S_i \cup \{r'\}$, for any $i \in [M]$. Let $z = \arg\max_{i\in[M]} p(S'_i)$. Then the group S'_z, which maximizes the prize is selected. We know that $M \leq \frac{4h}{\varepsilon}$. Hence, $p(S'_z) \geq \frac{\varepsilon}{4h}p(T') \geq \frac{\varepsilon}{4h} \cdot \frac{\gamma\varepsilon}{4}B = \frac{\gamma\varepsilon^2}{16h}B$.
   Note that in case $z = M$ and $c(S'_M) < \frac{\varepsilon}{2}B$, we select a subset of immediate out-subtrees from $\bigcup_{i=1}^{M-1} S_i$ with the total cost of at least $\frac{\varepsilon}{2}B$ and at most $\varepsilon B - c(S'_M)$, and add it to S'_z. Let $\hat{T}$ be the union of a shortest path P from r to r', S'_z, and the edges from r' to the roots of the out-subtrees in S_z. The cost of $\hat{T}$ is at most $(1+\varepsilon)B$ and the prize-to-cost ratio is at least $\frac{\gamma\varepsilon^2}{16h(1+\varepsilon)} \geq \frac{\gamma\varepsilon^2}{32h}$.
2. If $p(T') < \frac{\gamma\varepsilon}{4}B$, we proceed as follows. Consider the out-subtree $T'' = T_- \setminus T'$, which is rooted at r. Recall that by Claim 5, we have $c(T'') < \frac{\varepsilon}{2}B$. We connect a subset of immediate out-subtrees $T'_1, \ldots, T'_q$ of T' with cost $\frac{\varepsilon}{2}B - c(T'') \leq c(\bigcup_{i=1}^{q} T'_i) \leq \varepsilon B - c(T'')$ to the root of T'' through the root of T'. Since the cost of each immediate out-subtree of T' is less than $\frac{\varepsilon}{2}B$ (otherwise, we have a rich out-subtree) and $c(T') > (1+\frac{\varepsilon}{2})B$, a subset of immediate out-subtrees $T'_1, \ldots, T'_q$ of T' with such a cost can be found. We call the resulting out-subtree $\hat{T}$ and observe that $c(\hat{T}) \geq \frac{\varepsilon}{2}B$ (see Figure 2). We now bound the prize-to-cost ratio of $\hat{T}$. First note that by the submodularity of p, $p(T'') + p(T') \geq p(T_-)$. Thus by the subcase assumption and the monotonicity of p, we have $p(\hat{T}) \geq p(T'') \geq \frac{\gamma\varepsilon}{4}B$. Since $\frac{\varepsilon}{2}B - c(T'') \leq c(\bigcup_{i=1}^{q} T'_i) \leq \varepsilon B - c(T'')$ and the graph is B-appropriate, $c(\hat{T}) \leq (1+\varepsilon)B$.

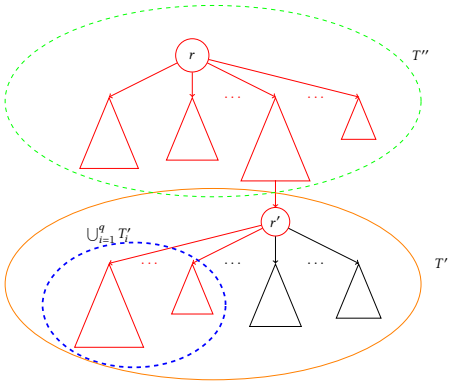


Figure 2 T_- is rooted at r, which is the whole out-tree. The green dashed circle represents T'' rooted at r. The orange circle represents T' rooted at r', where its cost is more than $(1+\frac{\varepsilon}{2})B$. The blue dashed circle represents a subset of immediate out-subtrees $T'_1,\ldots,T'_q$ of T' with cost $\frac{\varepsilon}{2}B - c(T'') \leq c(\bigcup_{i=1}^{q} T'_i) \leq \varepsilon B - c(T'')$. The red out-subtree represents $\hat{T}$, which is the union of T'', the edge from T'' to r', $\bigcup_{i=1}^{q} T'_i$ and the edges from r' to $T'_1,\ldots,T'_q$.

Therefore, the prize-to-cost ratio of the resulting out-subtree $\hat{T}$ is $\frac{\gamma\varepsilon}{4(1+\varepsilon)} \geq \frac{\gamma\varepsilon}{8} \geq \frac{\gamma\varepsilon^2}{32h}$. The proof is complete. ◀

To propose our algorithm, we need a last element. Let $U = \{x_1,\ldots,x_n\}$ be a ground set, $c : U \to Z^+$ be a cost function, $f : 2^U \to \mathbb{R}^+ \cup \{0\}$ be a monotone submodular function, and K be an integer budget. In the Submodular Maximization problem (**SM**), we are looking for a subset $S \subseteq U$ such that $|S| \leq K$ and $f(S)$ is maximum. Nemhauser et al. [28] provided a greedy algorithm that starts from $S := \emptyset$ and runs K iterations in which, at each iteration, it adds to S the element x which maximizes $f(S \cup \{x\}) - f(S)$. This algorithm guarantees a $(1-e^{-1})$-approximation for **SM**. We denote by **RSM** the rooted variant of **SM** in which, additionally, a specific element $v \in U$ is required to be included in the solution, that is we are looking for a subset $S \subseteq U$ such that $|S| \leq K$, $v \in S$ and $f(S)$ is maximum. We can run Nemhauser et al. [28]'s approach for **RSM** with a minor change: we initialize $S := \{v\}$ and run $K-1$ greedy iterations. We call this approach **Greedy**. It can be shown that **Greedy** guarantees a $(1-e^{-1})$-approximation algorithm for **RSM** (see e.g. [23]).

Now we can propose our approximation algorithm for **DRST**, which is reported in Algorithm 1. In words, Algorithm 1 first computes the maximal inclusion-wise B-appropriate subgraph for r of a given graph D by removing all the nodes at a distance larger than B from r. Let $D = (V, A)$ be the resulting directed graph. For each node u, it computes the set V_u of all nodes that are at a distance no more than $c(u) + \lfloor\sqrt{B}\rfloor$ from u. Let S^*_u be a subset of V_u such that $|S^*_u| \leq \lfloor\sqrt{B}\rfloor + 1$, $u \in S^*_u$, and $p(S^*_u)$ is maximum. Finding S^*_u requires to solve an instance I^{RSM}_u of **RSM** where the elements are V_u, the budget is $\lfloor\sqrt{B}\rfloor + 1$, the specific element is u, and profits are defined by function $p(\cdot)$. Using **Greedy**, Algorithm 1 computes in polynomial time an approximate solution S_u to I^{RSM}_u with $u \in S_u$, $|S_u| \leq \lfloor\sqrt{B}\rfloor + 1$ and $p(S_u) \geq (1-e^{-1})p(S^*_u)$. Finally, for each $u \in V$, Algorithm 1 computes a spanning out-tree T_u rooted in u that spans all the nodes in S_u. Let z be a node such that $p(T_z)$ is maximum, i.e., $z = \arg\max_{u \in V} p(T_u)$. Then, we have $c(T_z \setminus \{z\}) \leq B$ as $|S_z \setminus \{z\}| \leq \lfloor\sqrt{B}\rfloor$ and $dist(z, v) \leq c(z) + \lfloor\sqrt{B}\rfloor$ for any $v \in S_z$. If $z = r$, then Algorithm 1 defines $T = T_z$. Otherwise, it computes a shortest path P from r to z and defines T as the union of T_z and P. Since the obtained graph might not be an out-tree, Algorithm 1 removes the possible edges incoming the nodes in $V(T_z) \cap V(P)$ that belong only to T_z. The obtained out-tree T has a cost of at

Algorithm 1 DRST-Algo.

Input: Directed graph $D = (V, A)$; monotone submodular prize function $p : 2^V \to \mathbb{R}^+ \cup \{0\}$; cost function $c : V \to \mathbb{Z}^+$; root $r \in V$; budget B; and $\varepsilon' \in (0, 1]$.
Output: Out-tree T of D rooted at r such that $c(T) \leq (1 + \varepsilon')B$.

1: Remove from D all the nodes at a distance more than B from r;
2: **for** $u \in V$ **do**
3: $V_u := \{v \mid dist(u, v) \leq c(u) + \lfloor \sqrt{B} \rfloor\}$;
4: Define an instance I_u^{RSM} of **RSM** with elements V_u, specific element u, budget $\lfloor \sqrt{B} \rfloor + 1$, profits $p(S)$, for each $S \subseteq V_u$;
5: Let S_u be a $(1 - e^{-1})$-approximate solution to I_u^{RSM}, computed by using **Greedy**;
6: Let T_u be a minimal inclusion-wise out-tree rooted at u spanning all nodes in S_u;
7: **end for**
8: $z := \arg\max_{u \in V} p(T_u)$;
9: Let P be a shortest path from r to z;
10: $T := P \cup T_z$;
11: $A(T) := A(T) \setminus \{(v, w) \in A(T_z) \setminus A(P) \ : \ w \in V(T_z) \cap V(P)\}$;
12: Apply the trimming process in Lemma 3 with $\varepsilon = \varepsilon'$ to T;
13: **return** T.

most $2B$ as $dist(r, z) \leq B$ and $c(T_z \setminus \{z\}) \leq B$. Therefore, Algorithm 1 applies the trimming process in Lemma 3 to T to reduce the cost to $(1 + \varepsilon)B$, where $\varepsilon \in (0, 1]$ and outputs the resulting out-tree.

In the next theorem, we show that Algorithm 1 guarantees a bicriteria approximation.

▶ **Theorem 1.** ***DRST*** *admits a polynomial-time bicriteria* $\left(1 + \varepsilon, O\left(\frac{\sqrt{B}}{\varepsilon^3}\right)\right)$*-approximation algorithm, for any* $\varepsilon \in (0, 1]$.

For our analysis, we need to decompose an optimal out-tree into a bounded number of out-subtrees of bounded cost as in the following lemma, which is similar to Claim 3 in Kuo et al. [23] on the unrooted problem and undirected graphs.

▶ **Lemma 6.** *For any out-tree* $\hat{T} = (V, A)$ *rooted at* r *with cost* $c(\hat{T})$ *and any* $m \leq c(\hat{T})$*, there exist* $N \leq 5\lfloor \frac{c(\hat{T})}{m} \rfloor$ *out-subtrees* $T^i = (V^i, A^i)$ *of* $\hat{T}$*, for* $i \in [N]$*, where* $V^i \subseteq V$*,* $A^i = (V^i \times V^i) \cap A$*,* $c(V^i) \leq m + c(r_i)$*,* r_i *is the root of* T^i*, and* $\bigcup_{i=1}^{N} V^i = V$.

Proof. An out-subtree T' of $\hat{T}$ rooted at r' is called *feasible* if $c(V(T') \setminus \{r'\}) \leq m$; it is called *infeasible* otherwise.

Let us consider the following procedure called **Proc**. **Proc** takes as input an out-tree T', **Proc**(T'), and visits the vertices on T' from the leaves to the root. In this visiting process when **Proc** encounters a vertex v such that T'_v is the first infeasible full out-subtree, it removes T'_v from T', i.e., $T' = T' \setminus T'_v$. **Proc** iteratively repeats this process for the new tree T'. Finally, **Proc** returns all infeasible full out-subtrees that have been found in the visit.

Let $I_1, \ldots, I_s$ be the set of all infeasible full out-subtrees that have been returned after running **Proc**$(\hat{T})$ and let I_{s+1} be the possible feasible out-subtree rooted at r that remains after the visit of **Proc**$(\hat{T})$. We have $\cup_{i \in [s+1]} V(I_i) = V(\hat{T})$ and $V(I_i) \cap V(I_j) = \emptyset$, for $i \neq j$.

For each $i \in [s]$, let us consider the infeasible out-subtree I_i, let v_i be the root of I_i, and let I_u be the full out-subtree of I_i rooted at u, for each child u of v_i. Each out-subtree I_i is further divided into out-subtrees as follows:

- for all children u of v_i such that $c(I_u) \geq m/2$, we generate an out-subtree I_u, observe that $c(I_u) \leq m + c(u)$ because I_u is feasible. If all the children of v_i are in this category, we generate a further out-subtree made of only node v_i.
- All children u of v_i such that $c(I_u) < m/2$ are partitioned into groups of cost between $m/2$ and m, plus a possible group of cost smaller than $m/2$. It is always possible to partition the nodes in this way since $c(I_u) < m/2$ for all such nodes. Then, for each of these groups, we generate an out-subtree by connecting v_i to the roots of the out-subtrees in the group. All the generated out-subtrees have the same root v_i and cost at most $m + c(v_i)$.

The generated out-subtrees cover all the nodes in I_i. We add I_{s+1} to the set of generated out-subtrees, if it exists. Let $T^1, \ldots, T^N$ be the set of generated out-subtrees. Since $I_1, \ldots, I_{s+1}$ cover all the nodes of $\hat{T}$, then so do $T^1, \ldots, T^N$. Moreover, each generated out-subtree T^j costs at most $m + c(r_j)$, where r_j is the root of T^j.

We now bound the number N of generated out-subtrees. Given an infeasible out-subtree I_i, for some $i \in [s]$, each out-subtree generated from I_i costs at least $m/2$, except for the possible out-subtree made of only the root node of I_i and a possible out-subtree of cost smaller than $m/2$. Note that, by construction, at most one of these two additional out-subtrees can be generated. Hence, for each $i \in [s]$, the number s_i of out-subtrees generated from I_i is

$$s_i \leq \left\lfloor \frac{c(I_i)}{m/2} \right\rfloor + 1 \leq 2 \left\lfloor \frac{c(I_i)}{m} \right\rfloor + 2 \leq 4 \left\lfloor \frac{c(I_i)}{m} \right\rfloor.$$

Since $I_1, \ldots, I_{s+1}$ are disjoint, then the overall number of generated out-subtrees is at most $N \leq 1 + \sum_{i \in [s]} s_i \leq 1 + \sum_{i \in [s]} 4 \left\lfloor \frac{c(I_i)}{m} \right\rfloor \leq 5 \left\lfloor \frac{c(\hat{T})}{m} \right\rfloor$. ◀

Now we are ready to prove Theorem 1.

Proof of Theorem 1. By applying Lemma 6 to an optimal solution T^* and by setting $m = \lfloor \sqrt{B} \rfloor$, we obtain $N \leq 5\lfloor \sqrt{B} \rfloor$ out-subtrees $T^i = (V^i, A^i)$ of T^*, for $i \in [N]$, where $V^i \subseteq V(T^*)$, $A^i = (V^i \times V^i) \cap A(T^*)$, $c(V^i) \leq c(r_i) + \lfloor \sqrt{B} \rfloor$, r_i is the root of T^i, and $\bigcup_{i=1}^N V^i = T^*$. Let $p(T') = \max\{p(T^i) : i \in [N]\}$ and w be the root of T'. The submodularity of p implies $p(T^*) = p\left(\bigcup_{i=1}^N V(T^i)\right) \leq N p(T')$, which implies

$$p(T) \geq p(T_z) \geq p(S_w) \geq (1 - e^{-1}) p(S_w^*) \geq (1 - e^{-1}) p(T') \geq \frac{1 - e^{-1}}{N} p(T^*) \geq \frac{1 - e^{-1}}{5\lfloor \sqrt{B} \rfloor} p(T^*), \quad (3)$$

where the first two inequalities hold by the definitions of z and S_w and by the monotonicity of function p; The Third inequality holds because S_w is a $(1 - e^{-1})$-approximate solution for instance I_w^{RSM}; The fourth inequality holds as (i) T' contains nodes at a distance no more than $c(w) + \lfloor \sqrt{B} \rfloor$ from w and contains at most $1 + \lfloor \sqrt{B} \rfloor$ nodes (since the minimum cost of a node is at least 1) and (ii) $p(S_w^*) = \max\{p(S) : |S| \leq 1 + \lfloor \sqrt{B} \rfloor$ and $dist(w, v) \leq c(w) + \lfloor \sqrt{B} \rfloor$, for all $v \in S\}$.

Before the trimming process in Lemma 3, the ratio between the prize and the cost of T is at least $\gamma = \frac{1 - e^{-1}}{10\sqrt{B}B} p(T^*)$ as $c(T) \leq 2B$. After applying the trimming process in Lemma 3 (with $h = 2$) to T, the cost of T is at most $(1 + \varepsilon)B$ and its prize-to-cost ratio is:

$$\frac{p(T)}{c(T)} \geq \frac{\varepsilon^2 \gamma}{64} = \alpha \frac{\varepsilon^2}{\sqrt{B}B} p(T^*),$$

where $\alpha = \frac{1 - e^{-1}}{640}$. As $c(T) \geq \varepsilon B/2$, we have $p(T) \geq \frac{\alpha \varepsilon^3}{2\sqrt{B}} p(T^*)$, which concludes the proof. ◀

5 The unrooted version of DRST

Here we consider the unrooted version of **DRST**, denoted by **DUST**, in which the goal is to find an out-tree T of D such that $c(T) \leq B$ and $p(T)$ is maximum. Note that T can be rooted at any vertex. By guessing the root of an optimal solution, we can apply the algorithm in the previous section to obtain a bicriteria $(1+\varepsilon, O(\frac{\sqrt{B}}{\varepsilon^3}))$ approximation. We now show that **DUST** admits an $O(\sqrt{B})$-approximation algorithm with no budget violations. To do this, we first provide an unrooted version of Lemma 3 in which it is not necessary to violate the budget constraint when each vertex costs at most half of the budget. This trimming process follows the same procedure as that of Lemma 3, but we include it for the sake of completeness.

▶ **Lemma 7.** *Let T be an out-tree with the prize-to-cost ratio $\gamma = \frac{p(T)}{c(T)}$, where p is a monotone submodular function. Suppose $\frac{B}{2} \leq c(T) \leq hB$, where $h \in (1, n]$ and the cost of each vertex is at most $\frac{B}{2}$. One can find an out-subtree $\hat{T} \subseteq T$ with the prize-to-cost ratio at least $\frac{\gamma}{32h+8}$ such that $B/4 \leq c(\hat{T}) \leq B$.*

Proof. In the initial step, we remove a strict out-subtree T' of T if (i) the prize-to-cost ratio of $T \setminus T'$ is at least γ, and (ii) $c(T \setminus T') \geq \frac{B}{4}$. This process is performed iteratively, until no such out-subtree exists. Let T_- be the remaining out-subtree after applying this iterative process on T.

If $c(T_-) \leq B$, the desired out-subtree is obtained and we are done. Suppose it is not the case. A full out-subtree T' is called *rich* if $c(T') \geq \frac{B}{4}$ and the prize-to-cost ratio of T' and all its strict out-subtrees are at least γ. As in Lemma 3, we claim that the lemma follows from the existence of a rich out-subtree.

▷ Claim 8. Given a rich out-subtree T', the desired out-subtree $\hat{T}$ can be found.

Proof. Let T'' be the lowest rich out-subtree of T' such that the strict out-subtrees of T'' are not rich, i.e., $c(T'') \geq \frac{B}{4}$ while the cost of strict out-subtrees of T'' (if any exist) is less than $\frac{B}{4}$. Let C be the total cost of the immediate out-subtrees of T''. We distinguish between two cases:

1. If $C < \frac{B}{4}$, then $c(T'') \leq \frac{3B}{4}$ as the root of T'' costs at most $\frac{B}{2}$. Since T'' has the prize-to-cost ratio at least γ and cost at least $\frac{B}{4}$ (as it is rich), $\hat{T} = T''$ is the desired out-subtree.
2. If $C \geq \frac{B}{4}$, we first group the immediate out-subtrees of T'' into M groups $S_1, \ldots, S_M$ in such a way that for each $i \in [M-1]$ the total cost of immediate out-subtrees in S_i is at least $\frac{B}{4}$, and for each $i \in [M]$ it is at most $\frac{B}{2}$. As $c(T_-) \leq hB$, we have

$$M \leq \left\lceil \frac{hB}{B/4} \right\rceil = \lceil 4h \rceil \leq 4h+1.$$

For each $i \in [M]$, let $S'_i = S_i \cup \{r''\}$, where r'' is the root of T''. Let $z = \arg\max_{i \in [M]} p(S'_i)$. Hence by the submodularity and monotonicity of p, we have

$$p(S'_z) \geq \frac{\sum_{i=1}^{M} p(S'_i)}{4h+1} \geq \frac{p(S'_1) + \sum_{i=2}^{M} p(S_i)}{4h+1} \geq \frac{p(S'_1 \cup \bigcup_{i=2}^{M} S_i)}{4h+1} = \frac{p(T'')}{4h+1} \geq \frac{\gamma}{16h+4} B,$$

where the last inequality holds as $p(T'') \geq \gamma \frac{B}{4}$ (since T'' is rich).
In case $z = M$ and $c(S'_M) < \frac{B}{4}$, we select a subset of immediate out-subtrees from $\bigcup_{i=1}^{M-1} S_i$ with the total cost of at least $\frac{B}{4}$ and at most $\frac{B}{2} - c(S_M)$, and add it to S_z.
Let $\hat{T}$ be the union of r'', the edges from r'' to the roots of the out-subtrees in S_z, and S_z. The cost of $\hat{T}$ is at most B. Hence the prize-to-cost ratio of $\hat{T}$ is at least $\frac{\gamma}{16h+4} \geq \frac{\gamma}{32h+8}$. ◀

It only remains to consider the case when there is no rich out-subtree. Since T_- is not rich and $c(T_-) \geq \frac{B}{4}$, the ratio of at least one of the strict out-subtrees of T_- is less than γ. Now we find an out-subtree T' with ratio less than γ such that the ratio of all of its strict out-subtrees (if any exist) is at least γ. Since the ratio of T' is less than γ and T' is not removed in the initial process, $c(T_- \setminus T') < \frac{B}{4}$ (this can be shown by the same argument as that of Claim 5). As $c(T_-) > B$ and the cost of the root of T' is at most $\frac{B}{2}$, the total cost of the immediate out-subtrees of T' is at least $\frac{B}{4}$. Also, the cost of an immediate out-subtree of T' is less than $\frac{B}{4}$, otherwise we have a rich out-subtree. As the ratio and cost of T_- are at least γ and $\frac{B}{4}$, respectively, then $p(T_-) \geq \frac{\gamma}{4}B$. We distinguish between two cases.

1. If $p(T') \geq \frac{\gamma}{8}B$, by the similar reasoning as above, we partition the immediate out-subtrees of T' into M groups $S_1, \ldots, S_M$ in such a way that for each $i \in [M-1]$ the total cost of immediate out-subtrees in S_i is at least $\frac{B}{4}$, and for each $i \in [M]$ it is at most $\frac{B}{2}$. For each $i \in [M]$, let $S'_i = S_i \cup \{r'\}$ where r' is the root of T'. Let $z = \arg\max_{i\in[M]} p(S'_i)$. As $M \leq 4h+1$, by the submodularity and monotonicity of p we have:
$$p(S'_z) \geq \frac{\sum_{i=1}^{M} p(S'_i)}{4h+1} \geq \frac{p(S'_1) + \sum_{i=2}^{M} p(S_i)}{4h+1} \geq \frac{p(S'_1 \cup \bigcup_{i=2}^{M} S_i)}{4h+1} = \frac{p(T')}{4h+1} \geq \frac{\gamma}{32h+8}B,$$
where the last inequality holds as $p(T') \geq \frac{\gamma}{8}B$.
Note that in case $z = M$ and $c(S'_M) < \frac{B}{4}$, we select a subset of immediate out-subtrees from $\bigcup_{i=1}^{M-1} S_i$ with the total cost of at least $\frac{B}{4}$ and at most $\frac{B}{2} - c(S_M)$, and add it to S_z. Let $\hat{T}$ be the union of r', the edges from r' to the roots of the out-subtrees in S_z and S_z. The cost of $\hat{T}$ is at most B and its prize-to-cost ratio is at least $\frac{\gamma}{32h+8}$.
2. If $p(T') < \frac{\gamma}{8}B$, we proceed as follows. Consider the out-subtree $T'' = T_- \setminus T'$. Recall that by the above discussion we have $c(T'') < \frac{B}{4}$. Thus we find a subset S of the immediate out-subtrees of T' with cost between $\frac{B}{4} - c(T'') \leq c(S) \leq \frac{B}{2} - c(T'')$. Note that such set S can be found as each immediate out-subtree of T' costs less than $\frac{B}{4}$ (otherwise we have a rich subtree) and $c(T'') > \frac{3B}{4}$ (as $c(T_-) > B$ and $c(T'') < \frac{B}{4}$). Then let $\hat{T}$ be the union of T'', the edge from T'' to r' in T_-, S, and the edges from r' to the roots of the out-subtrees in S, where r' is the root of T'. We now bound the prize-to-cost ratio of $\hat{T}$. Recall that $T'' = T_- \setminus T'$. First note that by the submodularity' properties $p(T'') + p(T') \geq f(T_-)$. Thus by the case assumption and monotonicity, we have $f(\hat{T}) \geq p(T'') \geq \frac{\gamma}{8}B$. Since $\frac{B}{4}B - c(T'') \leq c(S) \leq \frac{B}{2} - c(T'')$ and $c(r') \leq \frac{B}{2}$, $c(\hat{T}) \leq B$. Therefore, the prize-to-cost ratio of $\hat{T}$ is at least $\frac{\gamma}{8} \geq \frac{\gamma}{32h+8}$.

The proof is complete. ◀

▶ **Theorem 9.** ***DUST*** *admits a polynomial-time $O(\sqrt{B})$-approximation algorithm.*

Proof. We follow arguments similar to those in Theorem 4 from Bateni et al. [2], but for the sake of completeness the proof is provided here.

An out-tree is called *flat* if each vertex of the out-tree costs no more than $\frac{B}{2}$. Let x be a vertex of an out-tree with the largest cost. An out-tree is called *saddled* if $c(x) > \frac{B}{2}$ and the cost of every other vertex of the out-tree is no more than $\frac{B-c(x)}{2}$. Let T_f^* (resp. T_s^*) be the optimal flat (resp. saddled) out-tree, i.e, a flat (resp. saddled) out-tree with cost at most B maximizing the prize. We first show that given an optimal solution T^* to **DUST**, then either $p(T_f^*) \geq \frac{p(T^*)}{2}$ or $p(T_s^*) \geq \frac{p(T^*)}{2}$.

▷ Claim 10. Either $p(T_f^*) \geq \frac{p(T^*)}{2}$ or $p(T_s^*) \geq \frac{p(T^*)}{2}$, where T^* is an optimal solution to **DUST**.

Proof. If T^* has only one vertex, then it is either flat or saddled and we are done. If T^* has more than one vertex and it is neither flat nor saddled, then we proceed as follows. Let x and y be two vertices in T^* with the maximum cost and the second maximum cost, respectively. Since T^* is not flat then $c(x) > \frac{B}{2}$ and $c(y) \leq \frac{B}{2}$. Also as T^* is not saddled, $c(y) > \frac{B-c(x)}{2}$, and, since the cost of T^* is at most B, y is the only node with a cost higher than $\frac{B-c(x)}{2}$. By removing the edge e adjacent to y on the path between x and y, we can partition T^* into two out-subtrees T^*_x and T^*_y that contain x and y, respectively. Clearly, each vertex in T^*_y costs no more than $\frac{B}{2}$, then T^*_y is flat. Also, each vertex in T^*_x except x costs at most $\frac{B-c(x)}{2}$, implying that T^*_x is saddled. By the submodularity of p, $p(T^*_x) + p(T^*_y) \geq p(T^*)$, meaning that one of T^*_x and T^*_y has at least half of the optimum prize $p(T^*)$, which concludes the claim. ◁

Now we restrict Algorithm 1 to only flat and saddled out-trees. Indeed, we can reduce the case of saddled out-trees to flat out-trees as follows. We first find a vertex x with the maximum cost. We then set the cost of x to zero and define a new budget $B' = B - c(x)$. Note that the cost of any other vertex in the optimal saddled out-tree T^*_s is at most half of the remaining budget. This means that we only need to find an approximation solution when restricted to flat out-trees.

Since for the new instance no other vertex except x with cost more than $\frac{B}{2}$ can be contained in the final solution, we remove all vertices with cost more than $\frac{B}{2}$ and run Lines 1-8 of Algorithm 1 on the new resulting graph to achieve an out-tree T with cost $c(T) \leq 2B$ (as $c(T \setminus \{z\}) = B$ and $c(z) \leq B$) and prize $p(T) \geq \frac{1-e^{-1}}{5\sqrt{B}} p(T^*_f) \geq \frac{1-e^{-1}}{10\sqrt{B}} p(T^*)$. So, the prize-to-cost ratio of T is $\gamma \geq \frac{p(T)}{2B}$. As T is flat, we can apply Lemma 7 to achieve an out-subtree $\hat{T}$ of T with the cost $B/4 \leq c(\hat{T}) \leq B$ and the prize-to-cost ratio $\frac{p(\hat{T})}{c(\hat{T})} \geq \frac{\gamma}{32h+8} = \frac{\gamma}{72}$ as $h \leq 2$. This implies that

$$p(\hat{T}) \geq \frac{\gamma}{72} \cdot \frac{B}{4} = \frac{\gamma}{288} B \geq \frac{p(T)}{576} \geq \frac{1-e^{-1}}{5760\sqrt{B}} p(T^*). \quad \blacktriangleleft$$

6 Submodular Tree Orienteering

Recently, Ghuge and Nagarajan [11] studied the *Submodular Tree Orienteering* problem (**STO**), which is similar to **DRST** with the only difference that the costs are associated to the edges of a directed graph instead of the nodes. In particular, in **STO**, we are given a directed graph $D = (V, A)$, a vertex $r \in V$, a budget B, a monotone submodular function $p : 2^V \to \mathbb{R}^+$, and a cost $c : A \to \mathbb{Z}^+$, and the goal is to find an out-subtree T of D rooted at r such that $\sum_{e \in A(T)} c(e) \leq B$ and $p(T) = p(V(T))$ is maximum. Ghuge and Nagarajan [11] proposed an $O\left(\frac{\log k}{\log \log k}\right)$-approximation algorithm for **STO** that runs in $(n \log B)^{O(\log^{1+\varepsilon} k)}$ time for any constant $\varepsilon > 0$, where $k \leq |V|$ is the number of vertices in an optimal solution.

Here we first show that **DRST** can be reduced to **STO**, preserving the approximation factor, by assigning the cost of each node v to all edges entering v.

▶ **Theorem 11.** *There is an $O(\frac{\log k}{\log \log k})$-approximation algorithm for **DRST** that runs in $(n \log B)^{O(\log^{1+\varepsilon} k)}$ time for any constant $\varepsilon > 0$, where $k \leq n = |V|$ is the number of vertices in an optimal solution.*

Proof. To prove the theorem, we show that one can transform an instance $J = \langle D' = (V', A'), p', c', r', B' \rangle$ of **DRST** to an instance $I = \langle D = (V, A), p, c, r, B \rangle$ of **STO** as follows. We set $V = V'$, $r = r'$, $A = A' \setminus \{(v, r') | (v, r') \in A'\}$, $B = B' - c'(r')$ and for any subset

$S \subseteq V$, $p(S) = p'(S)$. For any $e = (i,j) \in A$ in I, we set $c(e) = c'(j)$. The theorem follows by observing that any out-subtree T of D is an out-subtree for D', $c'(T) = \sum_{v \in V(T)} c'(v) = \sum_{e=(u,v) \in A(T)} c(e) + c'(r') = c(T) + c'(r')$, and $p'(T) = p(T)$. ◀

Moreover, we show that Algorithm 1 can be used to approximate **STO**. To our knowledge, this is the first polynomial-time bicriteria approximation algorithm for **STO**.

▶ **Theorem 12.** *There exists a polynomial-time bicriteria* $(1+\varepsilon, O(\frac{\sqrt{B}}{\varepsilon^3}))$*-approximation algorithm for* ***STO****, for any* $\varepsilon \in (0,1]$.

Proof. We first transform an instance $I_S = \langle D_S = (V_S, A_S), p_S, c_S, r, B\rangle$ of **STO** to an instance $I_D = \langle D_D = (V_D, A_D), p_D, c_D, r, B\rangle$ of **DRST**, where $V_D = V_S \cup V_A$, $V_A = \{v_e : e \in A_S\}$, $A_D = \{(i, v_e), (v_e, j) : e = (i,j) \in A_S\}$, $p_D(S) = p_S(S \cap V_S)$, for each $S \subseteq V_D$, $c_D(v) = 0$ for each $v \in V_S$, and $c_D(v_e) = c_S(e)$ for each $v_e \in V_A$.

Let T_S^* be an optimal solution for I_S and let T_D^* be the out-subtree of D' corresponding to T_S^* (i.e. $V(T_D^*) = V(T_S^*) \cup \{v_e : e \in A(T_S^*)\}$, $A(T_D^*) = \{(i, v_e), (v_e, j) : e = (i,j) \in A(T_S^*)\}$). We observe that $p_D(T_D^*) = p_S(T_S^*)$, $c_D(T_D^*) = c_S(T_S^*)$, and T_D^* is an optimal solution for I_D, since if there exists an out-subtree $\bar{T}_D$ of D_D with $p_D(\bar{T}_D) > p_D(T_D^*)$, then we can construct an out-subtree $\bar{T}_S = (V(\bar{T}_D) \cap V_S, \{e \in A_S : v_e \in V(\bar{T}_D) \cap V_A\})$ of D_S such that $p_S(\bar{T}_S) > p_S(T_S^*)$.

We decompose T_D^* as in Lemma 6,[2] with $m = \lfloor\sqrt{B}\rfloor$; let T_D' be the out-subtree that maximizes the prize among those returned by the lemma, and let w be the root of T_D'. We have that $p_D(T_D') \geq \frac{1}{5\lfloor\sqrt{B}\rfloor} p_D(T_D^*)$ and $c(T_D') \leq c(w) + \lfloor\sqrt{B}\rfloor$. It follows that the distance from w to any other node in T_D' is at most $c(w) + \lfloor\sqrt{B}\rfloor$.

We now show that $|V(T_D') \cap V_S| \leq \lfloor\sqrt{B}\rfloor + 1$. Since the cost of nodes in V_S is equal to 0, then $c((V(T_D') \cap V_A) \setminus \{w\}) = c(V(T_D') \setminus \{w\}) \leq \lfloor\sqrt{B}\rfloor$. Therefore, as the cost of each edge in A_S is at least 1, $|(V(T_D') \cap V_A) \setminus \{w\}| \leq \lfloor\sqrt{B}\rfloor$. For every node in $(V(T_D') \cap V_S) \setminus \{w\}$, there exists a distinct node in $V(T_D') \cap V_A$, which means that $|(V(T_D') \cap V_S) \setminus \{w\}| = |V(T_D') \cap V_A|$. If $w \in V_S$, then $|V(T_D') \cap V_A| = |(V(T_D') \cap V_A) \setminus \{w\}| \leq \lfloor\sqrt{B}\rfloor$. If $w \in V_A$, then $|(V(T_D') \cap V_A)| \leq \lfloor\sqrt{B}\rfloor + 1$. In both cases $|V(T_D') \cap V_S| \leq \lfloor\sqrt{B}\rfloor + 1$.

Let T_D be the output of lines 1-11 of Algorithm 1 for instance I_D. We have that

$$p_D(T_D) \geq (1-e^{-1}) p_D(S_w^*) \geq (1-e^{-1}) p_D(T_D') \geq \frac{1-e^{-1}}{5\lfloor\sqrt{B}\rfloor} p_D(T_D^*) = \frac{1-e^{-1}}{5\lfloor\sqrt{B}\rfloor} p_S(T_S^*),$$

where the second inequality is due to the fact that (i) T_D' contains nodes at a distance no more than $c(w)+\lfloor\sqrt{B}\rfloor$ from w and contains at most $\lfloor\sqrt{B}\rfloor+1$ nodes in V_S, and (ii) $p_D(S) = p_D(S \cap V_S)$, for each $S \subseteq V_D$, and therefore $p_D(S_w^*) = \max\{p_D(S) : |S \cap V_S| \leq \lfloor\sqrt{B}\rfloor + 1 \text{ and } dist(w,v) \leq c(w) + \lfloor\sqrt{B}\rfloor, \text{ for all } v \in S\}$. The other inequalities are analogous to those in (3).

The cost of T_D is at most $2B$, as in Theorem 1 we can trim T_D to reduce its cost to $(1+\varepsilon)B$ and maintaining a prize of $p_D(T_D) = \frac{\alpha\varepsilon^2}{\sqrt{B}} p(T_S^*)$, for some constant α and any arbitrary $\varepsilon > 0$.

Let us consider the out-subtree T_S of D_S corresponding to T_D, $T_S = (V(T_D) \cap V_S, \{e \in A_S : v_e \in V(T_D) \cap V_A\})$, then $p_S(T_S) = p_D(T_D)$ and $c_S(T_S) = c_D(T_D)$, which concludes the proof. ◀

7 Further Results on Some Variants of DRST

In this section, we provide approximation results on some variants of **DRST**. Due to space constraints, here we only state our results, all the details are given in a long version of the paper [6].

[2] Note that the Lemma 3 and 6 hold even if node costs are allowed to be equal to 0.

Additive prize function and Directed Tree Orienteering (DTO). We consider the special case of **DRST** in which the prize function is additive, i.e., for any $S \subseteq V$, $p(S) = \sum_{v \in S} p(\{v\})$, called **DRAT**. We show that there exists a polynomial-time bicriteria $(1+\varepsilon, O(\sqrt{B}/\varepsilon^2))$-approximation algorithm for **DRAT** (Theorem B.1 in [6]). By using the reduction in Theorem 12, it follows that this result also holds for **DTO**, which is the special case of **STO** in which the prize function is additive [11].

Undirected graphs. All our results hold also in the case in which the input graph is undirected and the output graph is a tree. In particular, our $O(\sqrt{B})$-approximation algorithm for the unrooted case improves over the factors $O((\Delta+1)\sqrt{B})$ [23] and $\min\{1/((1-1/e)(1/R-1/B)), B\}$ [15], where R is the radius of the input graph G. For the case in which the prize function is additive, we show that there exists a polynomial-time bicriteria approximation algorithm whose approximation factor only depends on the the maximum degree Δ of the given graph. In particular, it guarantees a bicriteria $(1+\varepsilon, 16\Delta/\varepsilon^2)$-approximation (Theorem B.2 in [6]).

Quota problem. We consider the problem in which we are given an undirected graph $G = (V, E)$, a cost function $c : V \to \mathbb{R}^+$, a prize function $p : V \to \mathbb{R}^+$, a quota $Q \in \mathbb{R}^+$, and a vertex r, and the goal is to find a tree T such that $p(T) \geq Q$, $r \in V(T)$ and $c(T)$ is minimum. We prove that this problem admits a 2Δ-approximation algorithm (Theorem B.3 in [6]).

Maximum Weighted Budgeted Connected Set Cover (MWBCSC). Let X be a set of elements, $\mathcal{S} \subseteq 2^X$ be a collection of sets, $p : X \to \mathbb{R}^+$ be a prize function, $c : \mathcal{S} \to \mathcal{R}^+$ be a cost function, $G_{\mathcal{S}}$ be a graph on vertex set $\mathcal{S}$, and B be a budget. In **MWBCSC**, the goal is to find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that $c(\mathcal{S}') = \sum_{S \in \mathcal{S}'} c(S) \leq B$, the subgraph induced by $\mathcal{S}'$ is connected and $p(\mathcal{S}') = \sum_{x \in X_{\mathcal{S}'}} p(x)$ is maximum, where $X_{\mathcal{S}'} = \bigcup_{S \in \mathcal{S}'} S$. We show that **MWBCSC** admits a polynomial-time αf-approximation algorithm, where f is the maximum frequency of an element and α is the performance ratio of an algorithm for the unrooted version of **MCSB** with additive prize function (Theorem B.4 in [6]). Moreover, one can have a polynomial-time $O(\log n)$-approximation algorithm for **MWBCSC** under the assumption that if two sets have an element in common, then they are adjacent in $G_{\mathcal{S}}$ (Corollary B.2 in [6]). This last result is an improvement over the factor $2(\Delta+1)\alpha/(1-e^{-1})$ by Ran et al. [30].

Budgeted Sensor Cover Problem (BSCP). In **BSCP**, we are given a set $\mathcal{S}$ of sensors, a set $\mathcal{P}$ of target points in a metric space, a sensing range R_s, a communication range R_c, and a budget B. A target point is covered by a sensor if it is within distance R_s from it. Two sensors are connected if they are at a distance at most R_c. The goal is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq B$, the number of covered target points by $\mathcal{S}'$ is maximized and $\mathcal{S}'$ induces a connected subgraph. We give a $2f$-approximation algorithm for **BSCP** (Theorem B.5 in [6]), where f is the maximum number of sensors that cover a target point. We also show that, under the assumption that $R_s \leq R_c/2$, **BSCP** admits a polynomial-time $8/(1-e^{-1})$-approximation algorithm (Theorem B.8 in [6]), which improves the factors $8(\lceil 2\sqrt{2}C \rceil + 1)^2/(1-1/e)$ [17] and $8(\lceil 4C/\sqrt{3} \rceil + 1)^2/(1-1/e)$ [34], where $C = R_s/R_c$. Note that Huang et al. [17] do not assume that $R_s/R_c \leq R_c/2$, however, our technique improves over their result if $R_s \leq R_c/2$.

References

1. Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for prize-collecting steiner tree and TSP. *SIAM J. Comput.*, 40(2):309–332, 2011.
2. Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi, and Vahid Liaghat. Improved approximation algorithms for (budgeted) node-weighted steiner problems. *SIAM J. Comput.*, 47(4):1275–1293, 2018.
3. Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.
4. Xuefeng Chen, Xin Cao, Yifeng Zeng, Yixiang Fang, and Bin Yao. Optimal region search with submodular maximization. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1216–1222, 2020.
5. Xiuzhen Cheng, Yingshu Li, Ding-Zhu Du, and Hung Q Ngo. Steiner trees in industry. In *Handbook of combinatorial optimization*, pages 193–216. Springer, 2004.
6. Gianlorenzo D'Angelo, Esmaeil Delfaraz, and Hugo Gilbert. Budgeted out-tree maximization with submodular prizes. *CoRR*, abs/2204.12162, 2022.
7. Kiril Danilchenko, Michael Segal, and Zeev Nutov. Covering users by a connected swarm efficiently. In *Algorithms for Sensor Systems - 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2020, , Revised Selected Papers*, volume 12503 of *Lecture Notes in Computer Science*, pages 32–44. Springer, 2020.
8. Xiaofeng Gao, Junwei Lu, Haotian Wang, Fan Wu, and Guihai Chen. Algorithm design and analysis for wireless relay network deployment problem. *IEEE Trans. Mob. Comput.*, 18(10):2257–2269, 2019.
9. N Garg. A 3 factor approximation algorithm for the minimum tree spanning k vertices. In *Proc of 37th Symp. on Foundations of Computer Science*, pages 302–309, 1996.
10. Naveen Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 396–402. ACM, 2005.
11. Rohan Ghuge and Viswanath Nagarajan. Quasi-polynomial algorithms for submodular tree orienteering and other directed network design problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1039–1048. SIAM, 2020.
12. Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.
13. Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k/\log\log k)$-approximation algorithm for directed steiner tree: A tight quasi-polynomial-time algorithm. *CoRR*, abs/1811.03020, 2018.
14. Sudipto Guha, Anna Moss, Joseph Naor, and Baruch Schieber. Efficient recovery from power outage (extended abstract). In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 574–582. ACM, 1999.
15. Dorit S. Hochbaum and Xu Rao. Approximation algorithms for connected maximum coverage problem for the discovery of mutated driver pathways in cancer. *Inf. Process. Lett.*, 158:105940, 2020.
16. Chien-Chung Huang, Mathieu Mari, Claire Mathieu, Joseph S. B. Mitchell, and Nabil H. Mustafa. Maximizing covered area in the euclidean plane with connectivity constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019*, volume 145 of *LIPIcs*, pages 32:1–32:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
17. Lingxiao Huang, Jian Li, and Qicai Shi. Approximation algorithms for the connected sensor cover problem. *Theor. Comput. Sci.*, 809:563–574, 2020.

18 David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: theory and practice. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769. ACM/SIAM, 2000.

19 Samir Khuller, Manish Purohit, and Kanthi K. Sarpatwar. Analyzing the optimal neighborhood: Algorithms for partial and budgeted connected dominating set problems. *SIAM J. Discret. Math.*, 34(1):251–270, 2020.

20 Philip N. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

21 Jochen Könemann, Sina Sadeghian Sadeghabad, and Laura Sanità. An LMP o(log n)-approximation algorithm for node weighted prize collecting steiner tree. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 568–577. IEEE Computer Society, 2013.

22 Guy Kortsarz and Zeev Nutov. Approximating some network design problems with node costs. *Theor. Comput. Sci.*, 412(35):4482–4492, 2011.

23 Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai. Maximizing submodular set function with connectivity constraint: Theory and application to networks. *IEEE/ACM Trans. Netw.*, 23(2):533–546, 2015.

24 Ioannis Lamprou, Ioannis Sigalas, and Vassilis Zissimopoulos. Improved budgeted connected domination and budgeted edge-vertex domination. *Theor. Comput. Sci.*, 858:1–12, 2021.

25 Heungsoon Felix Lee and Daniel R Dooly. Algorithms for the constrained maximum-weight connected graph problem. *Naval Research Logistics (NRL)*, 43(7):985–1008, 1996.

26 Shi Li and Bundit Laekhanukit. Polynomial integrality gap of flow LP for directed steiner tree. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 3230–3236. SIAM, 2022.

27 Anna Moss and Yuval Rabani. Approximation algorithms for constrained node weighted steiner tree problems. *SIAM J. Comput.*, 37(2):460–481, 2007.

28 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

29 Alice Paul, Daniel Freund, Aaron M. Ferber, David B. Shmoys, and David P. Williamson. Budgeted prize-collecting traveling salesman and minimum spanning tree problems. *Math. Oper. Res.*, 45(2):576–590, 2020.

30 Yingli Ran, Zhao Zhang, Ker-I Ko, and Jun Liang. An approximation algorithm for maximum weight budgeted connected set cover. *J. Comb. Optim.*, 31(4):1505–1517, 2016.

31 Stephan Seufert, Srikanta J. Bedathur, Julián Mestre, and Gerhard Weikum. Bonsai: Growing interesting small trees. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM 2010, The 10th IEEE International Conference on Data Mining*, pages 1013–1018. IEEE Computer Society, 2010.

32 Fabio Vandin, Eli Upfal, and Benjamin J. Raphael. Algorithms for detecting significantly mutated pathways in cancer. *J. Comput. Biol.*, 18(3):507–522, 2011.

33 Wenzheng Xu, Yueying Sun, Rui Zou, Weifa Liang, Qiufen Xia, Feng Shan, Tian Wang, Xiaohua Jia, and Zheng Li. Throughput maximization of UAV networks. *IEEE/ACM Transactions on Networking*, 30(2):881–895, 2022. doi:10.1109/TNET.2021.3125982.

34 Nan Yu, Haipeng Dai, Guihai Chen, Alex X. Liu, Bingchuan Tian, and Tian He. Connectivity-constrained placement of wireless chargers. *IEEE Trans. Mob. Comput.*, 20(3):909–927, 2021.

35 Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

36 Chenyang Zhou, Anisha Mazumder, Arun Das, Kaustav Basu, Navid Matin-Moghaddam, Saharnaz Mehrani, and Arunabha Sen. Relay node placement under budget constraint. In Paolo Bellavista and Vijay K. Garg, editors, *Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN*, pages 35:1–35:11. ACM, 2018.

A Further Related Work

Zelikovsky [35] provided the first approximation algorithm for Directed Steiner Tree Problem (**DSTP**) with factor $O(k^{\epsilon}(\log^{1/\epsilon} k))$ that runs in $O(n^{1/\epsilon})$, where $|U| = k$. Charikar et al. [3] proposed a better approximation algorithm for **DSTP** with a factor $O(\log^3 k)$ in quasi-polynomial time. Grandoni et al. [13] improved this factor and provided a randomized $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm $n^{O(\log^5 k)}$ time. Also, they showed that, unless $NP \subseteq \bigcap_{0<\varepsilon<1} \text{ZPTIME}(2^{n^{\varepsilon}})$ or the Projection Game Conjecture is false, there is no quasi-polynomial time algorithm for **DSTP** that achieves an approximation ratio of $o(\frac{\log^2 k}{\log\log k})$. Ghuge and Nagarajan [11] showed that their approximation algorithm results in a deterministic $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for **DSTP** in $n^{O(\log^{1+\epsilon} k)}$ time. Very recently, Li and Laekhanukit [26] showed that the lower bound on the integrality gap of the flow LP is polynomial in the number of vertices.

Danilchenko et al. [7] investigated a closely related problem to **BSCP**, where the goal is to place a set of connected disks (or squares) such that the total weight of target points in the plane is maximized. They provided a polynomial-time $O(1)$-approximation algorithm for this problem. **MCSB** is also closely related to the budgeted connected dominating set problem, where the goal is to select at most B connected vertices in a given undirected graph to maximize the profit function on the set of selected vertices. Khuller et al. [19] investigated this problem in which the profit function is a *special submodular function*. Khuller et al. [19] designed a $\frac{12}{1-1/e}$-approximation algorithm. By generalizing the analysis of Khuller et al. [19], Lamprou et al. [24] showed that there is a $\frac{11}{1-e^{-7/8}}$-approximation algorithm for the budgeted connected dominating set problem. They also showed that for this problem we cannot achieve in polynomial time an approximation factor better than $\left(\frac{1}{1-1/e}\right)$, unless $P = NP$.

Lee and Dooly [25] provided a $(B-2)$-approximation algorithm for **URAT**, where each vertex costs 1. Zhou et al. [36] studied a variant of **E-URAT** in the wireless sensor networks and provided a 10-approximation algorithm. Seufert et al. [31] investigated a special case of the unrooted version of **URAT**, where each vertex has cost 1 and we aim to find a tree with at most B nodes maximizing the accumulated prize. This coincides with the unrooted version of **E-URAT** when the cost of each edge is 1 and we are looking for a tree containing at most $B-1$ edges to maximize the accumulated prize. Seufert et al. [31] provided a $(5+\varepsilon)$-approximation algorithm for this problem. Similarly, Huang et al. [16] investigated this variant of **E-URAT** (or **URAT**) in the plane and proposed a 2-approximation algorithm.

The quota variant of **URAT** also has been studied, which is called **Q-URAT**. Here we wish to find a tree including a vertex r in a way that the total cost of the tree is minimized and its prize is no less than some *quota*. By using Moss and Rabbani [27]'black box and the ideas of Könemann et al. [21], and Bateni et al. [2], we have an $O(\log n)$-approximation algorithm for **Q-URAT**. This bound is tight [27]. The edge cost variant of **Q-URAT**, called **EQ-URAT**, has been investigated by Johnson et al. [18]. They showed that by adapting an α-approximation algorithm for the k-MST problem, one can have an α-approximation algorithm for **EQ-URAT**. Hence, the 2-approximation algorithm of Garg [10] for the k-MST problem results in a 2-approximation algorithm for **EQ-URAT**.

The prize collecting variants of **URAT** have also been studied. Könemann et al. [21] provided a Lagrangian multiplier preserving $O(\ln n)$-approximation algorithm for **NW-PCST**, where the goal is to minimize the cost of the nodes in the resulting tree plus the penalties of vertices not in the tree. Bateni et al. [2] considered a more general case of **NW-PCST** and provided an $O(\log n)$-approximation algorithm. There exists no $o(\ln n)$-approximation algorithm for **NW-PCST**, unless $NP \subseteq \text{DTIME}(n^{\text{Polylog}(n)})$ [20]. The edge cost variant

of **NW-PCST** has been investigated by Goemans and Williamson [12]. They provided a 2-approximation algorithm for **EW-PCST**. Later, Archer et al. [1] proposed a $(2-\varepsilon)$-approximation algorithm for **EW-PCST** which was an improvement upon the long standing bound of 2.

Table 1 A summary of the best bounds on some variants of prize collecting problems.

Problem	Best Bound
STO	$O(\frac{\log n}{\log\log n})$ [11] (tight)
DTO	$O(\frac{\log n}{\log\log n})$ [11] (tight)
DSTP	$O(\frac{\log^2 k}{\log\log k})$ [11, 13] (tight)
NW-PCST	$O(\log n)$ [2, 21] (tight)
EW-PCST	$2-\varepsilon$ [1]
URAT	$(1+\varepsilon, O(\frac{\log n}{\varepsilon^2}))$ [2, 21, 27]
E-URAT	2 [29]
Q-URAT	$O(\log n)$ [2, 21, 27] (tight)
EQ-URAT	2 [10, 18]

Clustering with Faulty Centers

Kyle Fox
University of Texas at Dallas, TX, USA

Hongyao Huang
University of Texas at Dallas, TX, USA

Benjamin Raichel
University of Texas at Dallas, TX, USA

Abstract

In this paper we introduce and formally study the problem of k-clustering with faulty centers. Specifically, we study the faulty versions of k-center, k-median, and k-means clustering, where centers have some probability of not existing, as opposed to prior work where clients had some probability of not existing. For all three problems we provide fixed parameter tractable algorithms, in the parameters k, d, and ε, that $(1+\varepsilon)$-approximate the minimum expected cost solutions for points in d dimensional Euclidean space. For Faulty k-center we additionally provide a 5-approximation for general metrics. Significantly, all of our algorithms have a small dependence on n. Specifically, our Faulty k-center algorithms have only linear dependence on n, while for our algorithms for Faulty k-median and Faulty k-means the dependence is still only $n^{1+o(1)}$.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Computational geometry

Keywords and phrases clustering, approximation, probabilistic input, uncertain input

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.10

Funding *Kyle Fox*: Research partially supported by NSF grant CCF-1942597.
Hongyao Huang: Research partially supported by NSF grant CCF-1750780.
Benjamin Raichel: Research partially supported by NSF grant CCF-1750780.

1 Introduction

There is a vast body of computational geometry literature which considers input points that are certain, that is they always exist and their location is known. However, uncertainty naturally arises when we are dealing with real world inputs. To model uncertain inputs, several works have considered the notion of probabilistic points. Two models for probabilistic points are commonly used: (i) the *existential* model [17, 19, 20], and (ii) the *locational* model [7, 11]. In the *existential* model, each probabilistic point has a certain fixed location if it exists, but it has a given probability of not existing. In the *locational* model, each probabilistic point always exists but its location is uncertain, and is instead specified by a probability density function over some region.

In this paper, we consider variants of the k-clustering problem under the *existential* model for the cluster centers. Specifically, we consider the k-center, k-median, and k-means problems, where the input points that must be covered are certain to exist, but each one of the k selected centers has an independent probability of existing, i.e. of being open to cover points. Our goal is then to select centers so as to minimize the expected furthest distance, sum of distances, or sum of squared distances, that points must travel to their nearest open center. We denote this as the *Faulty k-Clustering* problem. Prior papers have considered k-clustering in probabilistic input models, but where the centers are certain and the points needing to be covered are probabilistic (see for example [7]). To the best of our knowledge we are the first to consider probabilistic k-clustering, where the uncertainty is on

the cluster centers. Our variant of k-clustering is quite natural, as real world facilities can often have some probability of failure. Indeed, this real world motivation of faulty centers has inspired other previous work, though not in our probabilistic setting. Specifically, in the Fault Tolerant Clustering Problem (see for example [22]), the centers are certain to exists, though each point must travel to its l-th closest center. This objective attempts to provide robustness (i.e. failure tolerance) in the chosen centers. However, it less faithfully models the case where individual centers fail with some probability, since for example while one point's closest center may be closed, a different point's closest center may be open.

Related Work

The k-clustering problem with certain (i.e. non-probabilistic) input points is a classic and fundamental topic in computational geometry. The three most common variants are k-center, k-median, and k-means clustering. All three problems are know to be NP-hard. k-center is NP-hard to approximate within any factor less than 2 in general metric spaces [16] and hard to approximate within a factor of roughly 1.82 in the plane [8]. k-means is known to be NP-hard even when $k = 2$ [2], while k-median is known to be hard to approximate within a factor of $(1 + 2/e)$ [18]. Despite the hardness of these problems, there are many well known approximation algorithms. The standard greedy algorithm for k-center by Gonzalez [10] achieves an optimal 2-approximation to the optimal radius r_{opt}. By an alternative method, Hochbaum and Shmoys [15] also achieved a 2-approximation for k-center. For k-median and k-means it is known that local search achieves a constant factor approximation in polynomial time. (See discussion in [12] and references therein.) In Euclidean space, PTAS's exists for these problems when k, d, and ε are bounded. Specifically, Agarwal and Procopiuc [1] achieve a $(1 + \varepsilon)$-approximation for k-center in $O(n \log k) + (k/\varepsilon)^{O(k^{1-1/d})}$ time. For k-median and k-means a number of corset based $(1 + \varepsilon)$-approximation algorithms have been given which run in linear time in n, including Har-Peled and Mazumdar [13], and subsequent papers improving the time dependency on k, d, and ε [5, 9].

A number of prior works have considered variants of k-clustering where the client points that need to be covered are probabilistic, as opposed to our model where the centers are probabilistic. Perhaps most notably is the work of Cormode and McGregor [7]. They consider client points under the locational model, though as they also allow clients to have non-zero probability to not exist, their model also captures the existential model. For k-median and k-means they achieve $(1 + \varepsilon)$-approximations to the minimum expected cost solution in Euclidean space, and a constant factor approximation for k-median in general metrics. Their main focus, however, is on the more challenging case of k-center, for which they provide bi-criteria approximations for general metrics. That is, in addition to approximating the radius, they are allowed to exceed the requested number centers, and they provide different tradeoffs between the two kinds of approximation. Guha and Munagala [11] subsequently provided a non-bi-criteria approximation for k-center, obtaining an $O(1)$-approximation on only the expected radius. For points in $\mathbb{R}^d$, Huang and Li [17] later achieved the first PTAS for k-center, when k, d are fixed constants.

There have been a number of other follow up results to [7], again for the case of probabilistic clients not centers. For k-center on the real line, $\mathbb{R}^1$, Wang and Zhang [26] showed that the problem can be solved exactly in polynomial time. Munteanu *et al.* [24] considered the special case where $k = 1$ for both the k-center and the k-median objectives, and achieved polynomial time $(1 + \varepsilon)$-approximations. Moreover, in the data mining community, previous works have considered variations of probabilistic k-median [23] and k-means clustering [3, 25].

The idea of modeling faulty centers in clustering problems has also been considered in previous works under the setting of fault tolerant k-clustering [4, 21, 22]. In fault tolerant clustering centers are certain rather than probabilistic, and one assigns each point to its l nearest center for some integer $l \geq 1$. This is unlike our probabilistic model where each point is assigned to its nearest center that happens to be open.

1.1 Preliminaries

In the standard k-clustering problem, we are given a set P of n points in a metric space, called *clients*, and an integer parameter $k > 0$. The task is to select k points from P, called *centers*, so as to minimize some cost function of the distances of the client points in P to their nearest centers. In *k-center clustering* one minimizes the maximum distance from a client to its nearest center. In *k-median clustering* one minimizes the sum of distances from each client to their nearest center. Finally, in *k-means clustering*, one minimizes the sum of squared distances from each client to their nearest center. Note that for a given set of centers, the distances from the clients to their nearest centers, defines a vector of length n. The goal of k-center, k-median, or k-means, is then to minimize the ℓ_∞, ℓ_1, or ℓ_2 norm of this vector, respectively.

For these three problems we now formally define their respective cost functions for any given subset $C \subseteq P$. Specifically, for k-center, k-median, and k-means we respectively define the functions f, g, and h, as follows.

$$f_P(C) = \max_{p \in P} ||p - C|| = \max_{p \in P} \min_{c \in C} ||p - c||, \qquad g_P(C) = \sum_{p \in P} ||p - C|| = \sum_{p \in P} \min_{c \in C} ||p - c||,$$

$$h_P(C) = \sum_{p \in P} ||p - C||^2 = \sum_{p \in P} \min_{c \in C} ||p - c||^2.$$

The goal of k-center, k-median, or k-means is then to select the subset $C \subseteq P$ of size k that minimizes f_P, g_P, or h_P respectively. Note that for k-center clustering specifically, we often refer to $r = f_P(C)$ as the radius of the solution C, as r can be viewed as the radius of k equal radius balls covering the points in P.

Here we consider a variation of the standard k-clustering problem where there is uncertainty on whether any one of the chosen centers will be open, i.e. uncertainty on whether points in P can be covered by that center. Specifically, in addition to the point set P, as part of the input we are also given a vector V whose ith entry v_i is the probability that p_i will be open if it is chosen as a center. For any point p_i, when convenient we use $prob(p_i) = v_i$ to denote its associated probability.

▶ **Definition 1.** *Let P be a set of n points in a metric space, $V \in [0, 1]^n$ be a corresponding vector of probabilities, and $C \subseteq P$ be any subset. Any subset $R \subseteq C$ is called a* realization *of C, and let $Real(C)$ denote the set of all realizations (i.e. the power set of C). For a given realization R, a center $p \in C$ is said to be* open *(resp.* closed*) is $p \in R$ (resp. $p \notin R$). Each center $p \in C$ is open independently with probability $prob(p)$, thus the probability $R \in Real(C)$ occurs (i.e. is the set of open centers) is $Prob(R) = (\Pi_{p \in R}\ prob(p))(\Pi_{p \in C \setminus R}\ (1 - prob(p)))$.*

For this probabilistic version of k-clustering, our cost functions $f_{P,V}(C)$, $g_{P,V}(C)$, and $h_{P,V}(C)$ are now random variables, which for a given realization $R \subseteq C$ are equal to $f_P(R)$, $g_P(R)$, and $h_P(R)$, respectively. (Note that throughout we use the single subscript f_P to denote the non-probabilistic cost function, and the double subscript $f_{P,V}$ to denote the corresponding random variable version.) Our goal is now to find the subset C minimizing the expected value $E[f_{P,V}(C)]$, $E[g_{P,V}(C)]$, or $E[h_{P,V}(C)]$, where the expectation is taken over the distribution of $Real(C)$ determined by V.

▶ **Problem 2** (Faulty k-Center Clustering). *As input you are given a set P of n points in a metric space, a corresponding vector $V \in [0,1]^n$ of independent probabilities, and a positive integer parameter k. Find the subset C_{opt} of k centers which minimizes $E[f_{P,V}(C)]$. That is, $C_{opt} = \arg\min_{C \subseteq P, |C|=k} E[f_{P,V}(C)]$.*

▶ **Problem 3** (Faulty k-Median Clustering). *As input you are given a set P of n points in a metric space, a corresponding vector $V \in [0,1]^n$ of independent probabilities, and a positive integer parameter k. Find the subset C_{opt} of k centers which minimizes $E[g_{P,V}(C)]$. That is, $C_{opt} = \arg\min_{C \subseteq P, |C|=k} E[g_{P,V}(C)]$.*

▶ **Problem 4** (Faulty k-Means Clustering). *As input you are given a set P of n points in a metric space, a corresponding vector $V \in [0,1]^n$ of independent probabilities, and a positive integer parameter k. Find the subset C_{opt} of k centers which minimizes $E[h_{P,V}(C)]$. That is, $C_{opt} = \arg\min_{C \subseteq P, |C|=k} E[h_{P,V}(C)]$.*

▶ Remark 5. It is possible that all selected centers are closed, i.e. $R = \emptyset$. Thus to make sure the problem is well defined, we set $f_P(\emptyset)$, $g_P(\emptyset)$, and $h_P(\emptyset)$ equal to different specified values. Natural choices for these would depend on the input set P. For example, setting them equal to the optimal (non-probabilistic) 1-center, 1-median, or 1-mean solution, respectively. An alternative, though related approach, is to fix some center which is open with probability 1 and always included in the solution (i.e. not a part of the k selected centers).

Note that if all entries in V are the same then the probability that $R = \emptyset$ is the same, regardless of which subset C of size k is chosen, and thus the choice of $f_P(\emptyset)$ (resp. $g_P(\emptyset)$ and $h_P(\emptyset)$) does not affect the relative ordering of $E[f_{P,V}(C)]$ for different C. Furthermore, even in the case when the entries in V differ, for the solution our algorithm returns (as described below) the probability that $R = \emptyset$ is less than or equal to that for the optimal solution.

1.2 Our Contribution

To the best of our knowledge, we are the first to formally study the faulty k-clustering problem, where the probabilities are on the centers rather than the clients. As stated above, this is a natural setting, as centers may have some probability of failure.

For the three most common k-clustering variants, k-center, k-median, and k-means, we provide fixed parameter tractable approximation algorithms for their faulty versions. Specifically, for Faulty k-Center we provide an $O(8^k kn)$ time 5-approximation in general metrics, and an $O(dn\log(k)) + (1/\varepsilon)^{O(kd\log d)}$ time $(1+\varepsilon)$-approximation in the Euclidean case. For Faulty k-Median we provide an $(2^{O(k\log k)}/\varepsilon^{dk})n^{1+o(1)}$ time $(1+\varepsilon)$-approximation in the Euclidean case. Finally, for Faulty k-Means we provide an $(2^{O(k\log k)}/\varepsilon^{(2d+1)k})n^{1+o(1)}$ time $(1+\varepsilon)$-approximation in the Euclidean case.

It is important to note that all of our algorithms have a small dependence on n. Specifically, our Faulty k-Center algorithms have only linear dependence on n, while for our algorithms for Faulty k-Median and Faulty k-Means, the dependence is still only $n^{1+o(1)}$. Moreover, for all three problems, in the Euclidean case we are providing a $(1+\varepsilon)$-approximation, that is an EPTAS for fixed k and d.

Recall that the standard (non-faulty) versions of these problems are NP-hard, even in Euclidean settings, with additional results on hardness of approximation or for special cases, depending on which one of the three problems is considered. As the non-faulty versions are a special case of the faulty versions (where probabilities are all 1), these hardness results immediately apply to our problems. Moreover, our problems have the additional challenge that each choice of centers has an exponential number of possible realizations.

2 FPT Approximation Algorithms for k-Center

In this section we develop fixed parameter tractable (in the parameter k) approximation algorithms for Faulty k-Center Clustering. Specifically, for general metric spaces we achieve a 5-approximation to the optimal radius and in fixed dimensional Euclidean space we achieve a $(1+\varepsilon)$-approximation, i.e. a PTAS. For comparison, recall that for the standard non-probabilistic version of k-center clustering it is hard to approximate the radius within any constant factor less than 2 in general metric spaces [16], and there is a PTAS in Euclidean space. First, we present definitions and a core lemma, which are common to both algorithms.

Consider any instance P, k of Faulty k-Center, and let $\rho = \{\rho_1, \ldots, \rho_m\}$ be a partition of P into m disjoint subsets. Define the diameter of ρ as,

$$diam(\rho) = \max_i diam(\rho_i) = \max_i \max_{p,q \in \rho_i} ||p - q||.$$

For any subset $Z \subseteq P$, let $\rho(Z) = \{\rho_1(Z), \ldots, \rho_m(Z)\}$, where $\rho_i(Z) = Z \cap \rho_i$, denote the partition of Z induced by ρ. We then define the *characteristic vector* of Z with respect to ρ, denoted $char(Z, \rho)$ as the m dimensional integer vector whose ith entry is $|\rho_i(Z)|$. Define the *canonical subset*, $Canon(char(Z, \rho))$, of a characteristic vector $char(Z, \rho) = (w_1, \ldots, w_m)$, as the subset $S \subseteq P$ consisting of the w_i points with highest probability from ρ_i, for all i.[1] We then have the following key lemma.

▶ **Lemma 6.** *Let $\rho = \{\rho_1, \ldots, \rho_m\}$ be a partition of P into m subsets. Let $Q \subseteq P$ be any subset, and let $S = Canon(char(Q, \rho))$. Then we have,*

$$E[f_{P,V}(S)] \leq diam(\rho) + E[f_{P,V}(Q)].$$

Proof. Observe that for all i, $|\rho_i(S)| = |\rho_i(Q)|$, since S and Q have the same characteristic vector. For any i, label the points in $\rho_i(S) = \{s_1^i, \ldots, s_{w_i}^i\}$ and similarly in $\rho_i(Q) = \{q_1^i, \ldots, q_{w_i}^i\}$ in decreasing order of their probability. We define a bijection $b : S \to Q$, such that $b(s_j^i) = q_j^i$. Observe that the bijection b defines a bijection between $Real(S)$ and $Real(Q)$. Abusing notation slightly, for any realization R_S of S we use $b(R_S)$ to denote the corresponding realization in Q. Observe that by construction, $char(R_S, \rho) = char(b(R_S), \rho)$.

Let $R_S \in Real(S)$ be any realization of S. Consider any point $p \in P$. Let s be the closest point in R_S to p, and let q be the closest point in $b(R_S)$ to p. Let i be the index such that $q \in \rho_i$. Since $char(R_S, \rho) = char(b(R_S), \rho)$, there must be some point $s' \in R_S$ such that $s' \in \rho_i(S) \subseteq \rho_i$. Therefore, by the triangle inequality,

$$||p - s|| \leq ||p - s'|| \leq ||p - q|| + ||q - s'|| \leq ||p - q|| + diam(\rho).$$

This implies $f_P(R_S) = \max_{p \in P} ||p - R_S|| \leq \max_{p \in P} ||p - b(R_S)|| + diam(\rho) = f_P(b(R_S)) + diam(\rho)$.

For two vectors u, v of the same dimension, let $u \leq v$ denote that u is coordinate-wise smaller than v, i.e. $u_i \leq v_i$ for all i. So let V' be a probability vector such that $V' \leq V$. Then observe that $E[f_{P,V}(X)] \leq E[f_{P,V'}(X)]$ for any subset $X \subseteq P$. In other words, if each center in X has a smaller or equal probability to be open under V', then the expected cost cannot decrease when replacing V with V'.

[1] For points of equal probability, let there be an arbitrary but fixed ordering.

For any point $p \in P$, let $prob_V(p)$ denote the corresponding probability from the vector V. We define a new probability vector V' such that for any $p \in P$, if $p \in S$ then $prob_{V'}(p) = prob_V(b(p))$, and if $p \notin S$ then $prob_{V'}(p) = prob_V(p)$. Observe that since $\rho_i(S)$ consists of the w_i points with highest probability in ρ_i, we have that $prob_V(s^i_j) \geq prob_V(q^i_j)$ for any i, j. Therefore $V' \leq V$, and so by the above discussion, $E[f_{P,V}(S)] \leq E[f_{P,V'}(S)]$.

Let $prob_V(R_S)$ and $prob_{V'}(R_S)$ denote the probability that R_S is realized under V and V', respectively. Observe that $prob_{V'}(R_S) = prob_V(b(R_S))$. Therefore, we have the following,

$$\begin{aligned} E[f_{P,V}(S)] \leq E[f_{P,V'}(S)] &= \sum_{R_S \in Real(S)} prob_{V'}(R_S) \cdot f_P(R_S) \\ &= \sum_{b(R_S) \in Real(Q)} prob_V(b(R_S)) \cdot f_P(R_S) \\ &\leq \sum_{b(R_S) \in Real(Q)} prob_V(b(R_S)) \cdot (f_P(b(R_S)) + diam(\rho)) \\ &= E[f_{P,V}(Q)] + diam(\rho) \end{aligned}$$ ◀

2.1 General Metrics

Here we develop a fixed parameter tractable algorithm for Faulty k-Center Clustering in general metric spaces, which achieves a 5-approximation to the optimal radius.

Consider a subset $C = \{c_1, \ldots, c_k\}$ of k centers from P. For any other subset $Z \subseteq P$, let $Vor_i(Z, C)$ denote the subset of points in Z whose nearest center in C is c_i, e.g. when P is a point set in Euclidean space then this is the subset of points of Z in the Voronoi cell of c_i. Observe that $Vor(Z, C) = \{Vor_1(Z, C), \ldots, Vor_k(Z, C)\}$ defines a partition $\rho(Z) = \{\rho_1(Z), \ldots, \rho_k(Z)\}$ of Z where $\rho_i(Z) = Vor_i(Z, C)$. Thus the definitions of characteristic vectors and canonical subsets from above apply, where to simplify notation we write $char(Z, C) = char(Z, Vor(P, C))$. Moreover, by the triangle inequality

$$\begin{aligned} diam(Vor(P, C)) &= \max_i \max_{p,q \in Vor_i(P,C)} ||p - q|| \leq 2 \max_i \max_{p \in Vor_i(P,C)} ||p - c_i|| \\ &= 2 \max_{p \in P} ||p - C|| = 2 f_P(C), \end{aligned}$$

and thus we immediately have the following corollary of Lemma 6.

▶ **Corollary 7.** *Let $C = \{c_1, \ldots, c_k\}$ be a subset of k centers from P. Let $Q \subseteq P$ be any subset, and let $S = Canon(char(Q, C))$. Then we have,*

$$E[f_{P,V}(S)] \leq 2 f_P(C) + E[f_{P,V}(Q)].$$

Observe that for two different subsets $Z, Z' \subseteq P$ such that $|Z| = |Z'|$ it is possible to have $char(Z, C) = char(Z', C)$, however, we have the following bound on the total number of characteristic vectors for subsets of size k.

▶ **Observation 8.** *Let $C \subseteq P$ be a subset of k points. Then there are $O(4^k)$ possible characteristic vectors for all subsets of size k with respect to C. That is,*

$$\left| \bigcup_{C' \subseteq P, |C'| = k} char(C', C) \right| = O(4^k).$$

Proof. Recall $char(C', C)$ is a vector of length k whose (non-negative) entries sum to k. Any such vector can be represented as a binary vector of length $2k-1$ by writing each entry from the original vector in unary, and then separating entries with a single zero. The number of binary vectors of length $2k-1$ is $2^{2k-1} = O(4^k)$.[2] ◀

Before we present our algorithm, we make one more simple observation.

▶ **Observation 9.** *Given a point set P, probability vector V, and a subset C of k centers, $E[f_{P,V}(C)]$ can be computed in $O(2^k kn)$ time. Specifically, enumerate all possible $O(2^k)$ realizations in $Real(C)$. Then for a given realization R, the probability of R occurring is $(\Pi_{p_i \in R}\ v_i)(\Pi_{p_i \in C \setminus R}\ (1-v_i))$, and thus is computable in $O(k)$ time. Moreover, $f_P(R)$ can be computed in $O(kn)$ time, by checking for each point of P what is the closest point in R.*

▶ **Theorem 10.** *Let C_{opt} denote an optimal solution to Faulty k-Center. Then in $O(8^k kn)$ time one can compute a set $C \subseteq P$ of k centers such that $E[f_{P,V}(C)] \leq 5E[f_{P,V}(C_{opt})]$.*

Proof. The first step of our algorithm is to compute a set of k centers $D \subseteq P$, which is a 2-approximation to the optimal solution to the standard k-center instance on P, that is $f_P(D) \leq 2\min_{C' \subseteq P, |C'|=k} f_P(C')$. Note that D can be computed in $O(kn)$ time using the standard k-center algorithm of Gonzalez [10]. Next we guess the characteristic vector of C_{opt} with respect to D, $char(C_{opt}, D)$. This is done by enumerating all binary vectors of length $2k-1$ which have k 1's, as discussed in Observation 8. Let $W = (w_1, \ldots, w_k)$ denote the current guess for $char(C_{opt}, D)$. We construct the canonical subset $Canon(W)$, by taking the w_i points with highest probability from $Vor_i(P, D)$ for all i. Next we compute the expected cost of this subset $E[f_{P,V}(Canon(W))]$. After computing this value for all possible guesses of W, we then return as our solution $C = Canon(W)$ with minimum expected cost. (Note if W is not realizable, i.e. if there are fewer than w_i points in $Vor_i(P, D)$, then we simply record $E[f_{P,V}(Canon(W))] = \infty$.)

For the running time, computing D takes $O(kn)$ time. Next, for each guess $W = (w_1, \ldots, w_k)$ of $char(C_{opt}, D)$, computing $Canon(W)$ takes $O(kn)$ time, since finding the w_i points with highest probability from $Vor_i(P, D)$ can easily be done in $O(w_i n)$ time, and hence $O(kn)$ time over all i since $\sum w_i = k$. (Note this step can be performed faster by preprocessing the points, though ultimately it does not affect the asymptotic running time.) Next we must compute $E[f_{P,V}(Canon(W))]$, which can be done in $O(2^k kn)$ time by Observation 9. Thus since there are $O(4^k)$ possible guesses by Observation 8, the total time is $O(4^k(2^k kn + kn) + kn) = O(8^k kn)$

As for correctness, first observe that

$$f_P(D) \leq 2 \min_{C' \subseteq P, |C'|=k} f_P(C') \leq 2 \min_{C' \subseteq P, |C'|=k} E[f_{P,V}(C')] = 2E[f_{P,V}(C_{opt})].$$

Next, for $C = Canon(char(C_{opt}, D))$, by Corollary 7 we have $E[f_{P,V}(C)] \leq 2f_P(D) + E[f_{P,V}(C_{opt})]$. Combining these inequalities thus gives,

$$E[f_{P,V}(C)] \leq 2f_P(D) + E[f_{P,V}(C_{opt})] \leq 4E[f_{P,V}(C_{opt})] + E[f_{P,V}(C_{opt})] = 5E[f_{P,V}(C_{opt})]. \quad ◀$$

[2] Further using the fact that there are exactly k entries which are 1 in this vector of length $2k-1$, gives the more precise bound on the number of such vectors, $\binom{2k-1}{k} = O(4^k/\sqrt{k})$.

2.2 Euclidean PTAS

In this section we provide a fixed parameter tractable $(1+\varepsilon)$-approximation to the optimal radius for instances of Faulty k-Center Clustering where $P \subseteq \mathbb{R}^d$. To achieve this we consider the axis aligned regular grid of cell side length Δ, which is a value to be determined shortly. Specifically, for any point $p \in P$, where $p = (p^1, \ldots, p^d)$, its cell is given by $cell_\Delta(p) = (\lfloor p^1/\Delta \rfloor, \ldots, \lfloor p^d/\Delta \rfloor)$. Assuming this limited use of the floor function takes $O(1)$ time, in $O(dn)$ time we can compute the cell of every point in P. Moreover, as these cells are given by integer vectors, using hashing in the same time we can also compute the set of non-empty grid cells and the corresponding points in each cell. Let $Grid_\Delta(P)$ denote this partition of P into the non-empty grid cells. We have the following standard observation (see for example [14]).

▶ **Observation 11.** *Let $B(c,r)$ denote the ball of radius r and center c, for any point c and radius $r > 0$. Consider the regular grid of cell side length Δ. Then the number of grid cells intersecting $B(c,r)$ is at most $(2 + \lceil 2r/\Delta \rceil)^d$.*

The following theorem uses similar observations about grids and k-center as [1]. In particular, as a starting point, we similarly make use of the algorithm of Feder and Greene [8], which achieves a 2-approximation for k-center clustering in $O(dn \log k)$ time.

▶ **Theorem 12.** *Let $P \subset \mathbb{R}^d$ be an instance of Faulty k-Center Clustering in d-dimensional Euclidean space, and let C_{opt} denote an optimal solution to this instance. Then for any $\varepsilon > 0$, in $O(dn \log(k)) + (1/\varepsilon)^{O(kd \log d)}$ time[3] one can compute a set $C \subseteq P$ of k centers such that $E[f_{P,V}(C)] \leq (1+\varepsilon)E[f_{P,V}(C_{opt})]$.*

Proof. First, use the algorithm of [8] to compute a set of k centers C' which covers all of P within radius $r = f_P(C') \leq 2 \min_{Z \subseteq P, |Z|=k} f_P(Z) \leq 2E[f_{P,V}(C_{opt})]$. Now set $\Delta = \varepsilon r/(4\sqrt{d})$, and compute $Grid_\Delta(P)$. Let x denote the number of entries in $Grid_\Delta(P)$, where by Observation 11,

$$x \leq k(2 + \lceil 2r/\Delta \rceil)^d = k(2 + \lceil 8\sqrt{d}/\varepsilon \rceil)^d = k(1/\varepsilon)^{O(d \log d)}.$$

Observe that $Grid_\Delta(P)$ is a partition of P, with diameter $diam(Grid_\Delta(P)) \leq \Delta\sqrt{d} = \varepsilon r/4$. Let $S = Canon(char(C_{opt}, Grid_\Delta(P)))$ By Lemma 6 we have,

$$E[f_{P,V}(S)] \leq \frac{\varepsilon r}{4} + E[f_{P,V}(C_{opt})] \leq \frac{\varepsilon}{2}E[f_{P,V}(C_{opt})] + E[f_{P,V}(C_{opt})] = (1+\varepsilon/2)E[f_{P,V}(C_{opt})].$$

Therefore in order to compute a $(1+\varepsilon/2)$-approximation, it suffices to compute all possible characteristic vectors for $char(C_{opt}, Grid_\Delta(P))$, evaluate the expected cost of their corresponding canonical subsets, and take the minimum. To speed up the time to evaluate the expected cost of a canonical subset, we instead compute additive $\varepsilon r/4 \leq (\varepsilon/2)E[f_{P,V}(C_{opt})]$ approximations to these values, thus in total achieving a relative $(1+\varepsilon)$-approximation.

As for the running time, similar to Observation 8 one can argue that the number of possible characteristic vectors is at most

$$\binom{x+k-1}{k} \leq \left(\frac{(x+k)e}{k}\right)^k = (1/\varepsilon)^{O(kd \log d)}.$$

[3] Technically the $(1/\varepsilon)^{O(kd \log d)}$ bound term assumes $\varepsilon < 1$. If $\varepsilon \geq 1$ this terms becomes $2^{O(kd \log d)}$. That said, it is standard practice to write the bound in this simplified manner.

For each characteristic vector we need to compute the corresponding canonical subset. Observe that within a given cell of $Grid_\Delta(P)$, the canonical subset consists of the m highest probability points for some value $m \leq k$. Therefore, as a preprocessing step, we can throw out all but the k highest probability points in each cell, and then sort the remaining points in each cell by their probability. Throwing out all but the k highest probability points takes $O(n)$ time in total by using linear time median selection in each cell. All the sorting can be done in $O(x \cdot k \log k)$ time total. After preprocessing, for a specific characteristic vector, it takes $O(x+k)$ time to compute the canonical subset. Suppose that for this canonical subset, we can compute an additive $\varepsilon r/4$-approximation to its expected cost in $O(2^k kx)$ time. Then the total time is

$$O(dn \log k + n + xk \log k) + (x + k + 2^k kx)(1/\varepsilon)^{O(kd \log d)} = O(dn \log k) + (1/\varepsilon)^{O(kd \log d)}.$$

So let S be any given canonical subset. What remains is to show an additive $\varepsilon r/4$-approximation to its expected cost can be computed in $O(2^k kx)$ time. Let R be any realization of S. Consider the set of points X in some cell of the grid partition $Grid_\Delta(P)$. Consider any two points $p, q \in X$, and let r_p, r_q be the nearest center in R to p and q respectively. By the triangle inequality we have

$$||p - r_p|| \leq ||p - r_q|| \leq ||p - q|| + ||q - r_q|| \leq \sqrt{d}\Delta + ||q - r_q|| = \varepsilon r/4 + ||q - r_q||.$$

Therefore, $||q - r_q|| \leq \max_{p \in X} ||p - R|| \leq \varepsilon r/4 + ||q - r_q||$, that is the distance from q to its nearest center in R is an additive $\varepsilon r/4$ approximation to the maximum distance of a point in the cell to its nearest center in R. Thus to get an $O(kx)$ time additive $\varepsilon r/4$-approximation to $f_P(R)$, it suffices to pick an arbitrary representative q in each cell, compute its nearest center in R, and take the maximum. As there are $O(2^k)$ possible realizations R of S, the claim now follows in a similar fashion to Observation 9. ◀

3 k-Median and k-Means

In this section, we develop a fixed parameter tractable approximation scheme for Faulty k-Median Clustering of a collection of points $P \subset \mathbb{R}^d$ in d-dimensional Euclidean space. As in our approximation scheme for Faulty k-Center, we begin by finding a constant-factor approximate solution to k-Median without faulty centers and then partition the points of P into disjoint subsets based on their location in appropriately sized regions of space. Unlike the algorithm for Faulty k-Center, however, these subsets may have different diameters depending on how far away each subset is from the nearest member of the non-faulty k-Median solution. At the end of this section, we describe the minor changes necessary for our algorithm to apply to Faulty k-Means instead of k-Median. For simplicity, throughout this section we assume $0 < \varepsilon \leq 1$.

We now turn to our algorithm description. Let C'_{opt} denote an optimal solution for non-faulty k-Median. We compute a set of centers $D = \{d_1, \ldots, d_k\}$ such that $g_P(C'_{opt}) \leq g_P(D) \leq 2g_P(C'_{opt})$. The set D can be computed in $O(n) + k^{O(1)} \log^{O(1)} n$ time [13].[4]

As in Section 2.1, we partition P into a collection of k subsets defined by the distance from each point to its nearest member of D. For any subset $Z \subseteq P$ and any $i \in \{1, \ldots, k\}$, let $Vor_i(Z, D)$ denote the subset of points of Z whose nearest center in D is d_i.

[4] The algorithms in [13] are randomized, though they achieve this time with high probability. There are deterministic constant factor approximations, though with higher polynomial dependence on n.

We proceed to refine the above partition as follows. Fix any $i \in \{1, \ldots, k\}$. Let $t = \left\lceil \log_{(1+\varepsilon)} 2n \right\rceil = O((1/\varepsilon) \log n)$. For each $j \in \{0, \ldots, t\}$, let $r_j = (g_P(D)/(2n)) \cdot (1+\varepsilon)^j$, and let $B_{i,j} = B(d_i, r_j)$, the ball of radius r_j centered at d_i. We partition the points of $Vor_i(P, D) \cap B_{i,0}$ into $O(1/\varepsilon^d)$ *batches* each of diameter $\varepsilon r_0/4$. For each $j \in \{1, \ldots, t\}$, partition the points of $Vor_i(P, D) \cap (B_{i,j} \setminus B_{i,j-1})$ into $O(1/\varepsilon^{d-1})$ batches each of diameter $\varepsilon r_j/8$. These batches can be computed in time linear in their number and the size of $Vor_i(P, D)$ by partitioning points according to their location in a sufficiently fine grid. (See the discussion before Observation 11 and Observation 11 itself.)

We perform the above assignment to batches for each $i \in \{1, \ldots, k\}$. Observe that no point $p \in P$ can lie further than $g_P(D)$ from its nearest center in D, implying all points are assigned to exactly one batch. Let $\rho = \{\rho_1, \ldots, \rho_m\}$ denote the partition of P into batches.

▶ **Observation 13.** *For the partition $\rho = \{\rho_1, \ldots, \rho_m\}$, we have $m = O((k/\varepsilon^d) \log n)$.*

Recall the definitions of canonical subsets given in Section 2. As in our algorithm for Faulty k-Center, we will enumerate characteristic vectors for subsets of size k with respect to ρ, taking the best canonical subset for these vectors as our solution. We adapt Lemma 6 for the current setting.

▶ **Lemma 14.** *Let $\rho = \{\rho_1, \ldots, \rho_m\}$ denote the partition of P as described above. Let $Q \subseteq P$ be any subset such that $|Q| \leq k$, and let $S = Canon(char(Q, \rho))$. Then we have,*

$$E[g_{P,V}(S)] \leq (1+\varepsilon) E[g_{P,V}(Q)].$$

Proof. Our proof follows the one for Lemma 6 except when comparing costs between two realizations $R_S \in Real(S)$ and $R_Q \in Real(Q)$ with the same characteristic vector. Consider any point $p \in P$. Let s be the closest point in R_S to p, and let q be the closest point in R_Q to p. Let i_q be such that $q \in Vor_{i_q}(P, D)$, and let ℓ be the index such that $q \in \rho_\ell$. Given R_S and R_Q have the same characteristic vector, there must be some point $s' \in R_S$ such that $s' \in \rho_\ell$. We have

$$||p - s|| \leq ||p - s'|| \leq ||p - q|| + ||q - s'||.$$

Suppose $q \in B_{i_q,0}$. Recall, $B_{i_q,0}$ is the ball of radius $r_0 = g_P(D)/(2n)$ centered at d_{i_q}. Then,

$$||p - s|| \leq ||p - q|| + ||q - s'|| \leq ||p - q|| + \frac{\varepsilon r_0}{4} = ||p - q|| + \frac{\varepsilon g_P(D)}{8n}.$$

Now, suppose $q \in (B_{i_q,j} \setminus B_{i_q,j-1})$ for some $j > 0$. Centers d_{i_p} and d_{i_q} are the closest members of D for p and q respectively, and $||q - D|| \geq r_{j-1}$. By triangle inequality,

$$r_{j-1} \leq ||q - D|| \leq ||q - p|| + ||p - D||.$$

Therefore,

$$\begin{aligned} ||p - s|| &\leq ||p - q|| + ||q - s'|| \leq ||p - q|| + \frac{\varepsilon(1+\varepsilon) r_{j-1}}{8} \leq ||p - q|| + \frac{\varepsilon r_{j-1}}{4} \\ &\leq ||p - q|| + \frac{\varepsilon ||p - q||}{4} + \frac{\varepsilon ||p - D||}{4}. \end{aligned}$$

Finally, summing over all p and observing $g_P(D) \leq 2 g_P(R_Q)$.

$$\begin{aligned} g_P(R_S) = \sum_{p \in P} ||p - R_S|| &\leq \sum_{p \in P} \left(||p - R_Q|| + \frac{\varepsilon g_P(D)}{8n} + \frac{\varepsilon ||p - R_Q||}{4} + \frac{\varepsilon ||p - D||}{4} \right) \\ &\leq g_P(R_Q) + \frac{\varepsilon g_P(R_Q)}{4} + \frac{\varepsilon g_P(D)}{8} + \frac{\varepsilon g_P(D)}{4} \\ &\leq (1+\varepsilon) g_P(R_Q). \end{aligned}$$

The rest of the proof proceeds as in the one for Lemma 6, with "$(1+\varepsilon)E[g_{P,V}(Q) \mid e_Q(u)]$" appearing in place of "$E[f_{P,V}(Q) \mid e_Q(u)] + diam(\rho)$" alongside analogous substitutions. ◀

We conclude with our main theorem for Faulty k-Median.

▶ **Theorem 15.** *Let $P \subset \mathbb{R}^d$ be an instance of Faulty k-Median in d-dimensional Euclidean space, and let C_{opt} denote an optimal solution to this instance. Then for any $\varepsilon > 0$, in $(2^{O(k \log k)}/\varepsilon^{dk})n^{1+o(1)}$ time, one can compute a set $C \subseteq P$ of k centers such that $E[g_{P,V}(C)] \leq (1+\varepsilon)E[g_{P,V}(C_{opt})]$.*

Proof. As stated above, our algorithm begins by computing the set $D \subseteq P$ of k centers in $O(n) + k^{O(1)} \log^{O(1)} n$ time [13]. We then find the closest center in D for each point in P, and then refine this partition into batches as described above. This produces a partition ρ into $O((k/\varepsilon^d) \log n)$ batches by Observation 13, and thus ρ is computed in $O(kn + (k/\varepsilon^d) \log n)$ time. We enumerate all $O((k/\varepsilon^d) \log n)^k)$ possible characteristic vectors W for $char(C_{opt}, \rho)$, finding and computing the expected cost of the canonical subset $Canon(W)$ in $O(2^k kn)$ time for each, and return the solution C with the minimum expected cost. Lemma 14 implies our solution has the correct expected cost.

We observe that $\log^k n = 2^{O(k \log k)} n^{o(1)}$ [6]: If $n \leq 2^{k^2}$, then $\log^k n \leq k^{2k} = 2^{O(k \log k)}$, and if $n > 2^{k^2}$, then $\log^k n \leq 2^{\sqrt{\log n} \log \log n} = n^{o(1)}$. Therefore, for the $(2^{O(k \log k)}/\varepsilon^{dk})n^{o(1)}$ characteristic vectors, in $(2^{O(k \log k)}/\varepsilon^{dk})n^{1+o(1)}$ time total we compute the expected costs of their canonical subsets, and this dominates the running time of all other operations. ◀

3.1 k-Means

We now discuss how to modify our algorithm for Faulty k-Median to instead find a $(1+\varepsilon)$-approximate solution to Faulty k-Means. As before, we start by computing a 2-approximate solution D to non-faulty k-Means in $O(n + k^{k+2}\varepsilon^{-(2d+1)k} \log^{k+1} n \log^k(1/\varepsilon))$ time [13]. Then, we partition the points of P into the k subsets $Vor_i(P, D)$ and refine the partition.

We need to modify the refined partition somewhat to work with k-Means. We set the parameter t that determines the number of concentric balls to $\lceil \log_{1+\varepsilon}(\sqrt{2n}) \rceil$, and let $r_j = \sqrt{h_P(D)/(2n)} \cdot (1+\varepsilon)^j$. We now partition the points of $Vor_i(P, D) \cap B_{i,0}$ into $O(1/\varepsilon^d)$ batches each of diameter $\varepsilon r_0/8$, and for each $j \in \{1, \ldots, t\}$, partition the points of $Vor_i(P, D) \cap (B_{i,j} \setminus B_{i,j-1})$ into $O(1/\varepsilon^{d-1})$ batches each of diameter $\varepsilon r_j/16$.

The rest of the algorithm itself remains the same, but for the analysis we need to revise Lemma 14 to account for the different objective in k-Means.

▶ **Lemma 16.** *Let $\rho = \{\rho_1, \ldots, \rho_m\}$ denote the partition of P for Faulty k-Means. Let $Q \subseteq P$ be any subset such that $|Q| \leq k$, and let $S = Canon(char(Q, \rho))$. Then we have,*

$$E[h_{P,V}(S)] \leq (1+\varepsilon)E[h_{P,V}(Q)].$$

Proof. We recall the notation from Lemma 6 and Lemma 14 that is used in the novel part of the proof. Let $R_S \in Real(S)$ and $R_Q \in Real(Q)$ have the same characteristic vector. Consider any point $p \in P$. Let s be the closest point in R_S to p, and let q be the closest point in R_Q to p. Let i_q be such that $q \in Vor_{i_q}(P, D)$, and let ℓ be the index such that $q \in \rho_\ell$. Given R_S and R_Q have the same characteristic vector, there must be some point $s' \in R_S$ such that $s' \in \rho_\ell$. We have

$$||p - s|| \leq ||p - s'|| \leq ||p - q|| + ||q - s'||.$$

The remainder of the proof will rely on the fact that for any $x, y \geq 0$, $xy \leq x^2/2 + y^2/2$. This fact is a special case of both Young's [27] inequality for products and the AM-GM inequality.

Suppose $q \in B_{i_q,0}$. Recall, $B_{i_q,0}$ is the ball of radius $r_0 = \sqrt{h_P(D)/(2n)}$ centered at d_{i_q}. Then,

$$\begin{aligned}
||p-s||^2 \leq (||p-q|| + ||q-s'||)^2 &\leq \left(||p-q|| + \frac{\varepsilon r_0}{8}\right)^2 = \left(||p-q|| + \frac{\varepsilon\sqrt{h_P(D)}}{8\sqrt{2n}}\right)^2 \\
&= ||p-q||^2 + \frac{\varepsilon}{4}||p-q|| \cdot \frac{\sqrt{h_P(D)}}{\sqrt{2n}} + \frac{\varepsilon^2 h_P(D)}{128n} \\
&= ||p-q||^2 + \sqrt{\frac{\varepsilon}{4}}||p-q|| \cdot \sqrt{\frac{\varepsilon h_P(D)}{8n}} + \frac{\varepsilon^2 h_P(D)}{128n} \\
&\leq ||p-q||^2 + \frac{\varepsilon||p-q||^2}{8} + \frac{9\varepsilon h_P(D)}{128n}
\end{aligned}$$

Now, suppose $q \in (B_{i_q,j} \setminus B_{i_q,j-1})$ for some $j > 0$. Again, $r_{j-1} \leq ||p-q|| + ||p-D||$. Therefore,

$$\begin{aligned}
||p-s||^2 \leq (||p-q|| + ||q-s'||)^2 &\leq \left(||p-q|| + \frac{\varepsilon(1+\varepsilon)r_{j-1}}{16}\right)^2 \\
&\leq \left(||p-q|| + \frac{\varepsilon r_{j-1}}{8}\right)^2 \leq \left(\left(1+\frac{\varepsilon}{8}\right)||p-q|| + \frac{\varepsilon||p-D||}{8}\right)^2 \\
&= \left(1 + \frac{\varepsilon}{4} + \frac{\varepsilon^2}{64}\right)||p-q||^2 + 2\left(\frac{\varepsilon}{8} + \frac{\varepsilon^2}{64}\right)||p-q|| \cdot ||p-D|| + \frac{\varepsilon^2||p-D||^2}{64} \\
&\leq \left(1 + \frac{17\varepsilon}{64}\right)||p-q||^2 + 2\sqrt{\frac{9\varepsilon}{64}}||p-q|| \cdot \sqrt{\frac{9\varepsilon}{64}}||p-D|| + \frac{\varepsilon||p-D||^2}{64} \\
&\leq ||p-q||^2 + \frac{13\varepsilon||p-q||^2}{32} + \frac{5\varepsilon||p-D||^2}{32}
\end{aligned}$$

Finally, summing over all p and observing $h_P(D) \leq 2h_P(R_Q)$.

$$\begin{aligned}
h_P(R_S) = \sum_{p \in P} ||p-R_S||^2 &\leq \sum_{p \in P}\left(||p-R_Q||^2 + \frac{13\varepsilon||p-R_Q||^2}{32} + \frac{9\varepsilon h_P(D)}{128n} + \frac{5\varepsilon||p-D||^2}{32}\right) \\
&\leq h_P(R_Q) + \frac{13\varepsilon h_P(R_Q)}{32} + \frac{9\varepsilon h_P(D)}{128} + \frac{5\varepsilon h_P(D)}{32} \\
&\leq (1+\varepsilon)h_P(R_Q).
\end{aligned}$$

The rest of the proof proceeds as in the one for Lemma 6, with "$(1+\varepsilon)E[h_{P,V}(Q) \mid e_Q(u)]$" appearing in place of "$E[f_{P,V}(Q) \mid e_Q(u)] + diam(\rho)$" alongside analogous substitutions. ◀

The running time analysis for Faulty k-Means is virtually the same as that for Faulty k-Median, except for the larger time needed to find the initial non-faulty 2-approximate solution. We thus conclude with our final theorem.

▶ **Theorem 17.** *Let $P \subset \mathbb{R}^d$ be an instance of Faulty k-Means in d-dimensional Euclidean space, and let C_{opt} denote an optimal solution to this instance. Then for any $\varepsilon > 0$, in $(2^{O(k\log k)}/\varepsilon^{(2d+1)k})n^{1+o(1)}$ time, one can compute a set $C \subseteq P$ of k centers such that $E[h_{P,V}(C)] \leq (1+\varepsilon)E[h_{P,V}(C_{opt})]$.*

References

1. P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.
2. Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, 2009. doi:10.1007/s10994-009-5103-0.
3. Michael Chau, Reynold Cheng, Ben Kao, and Jackey Ng. Uncertain data mining: An example in clustering location data. In *Advances in Knowledge Discovery and Data Mining, 10th Pacific-Asia Conference (PAKDD)*, volume 3918, pages 199–204. Springer, 2006. doi:10.1007/11731139_24.
4. Shiva Chaudhuri, Naveen Garg, and R. Ravi. The p-neighbor k-center problem. *Inf. Process. Lett.*, 65(3):131–134, 1998. doi:10.1016/S0020-0190(97)00224-X.
5. Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009. doi:10.1137/070699007.
6. Vincent Cohen-Addad, Marcin Pilipczuk, and Michal Pilipczuk. Efficient approximation schemes for uniform-cost clustering problems in planar graphs. In *27th Annual European Symposium on Algorithms (ESA)*, pages 33:1–33:14, 2019. doi:10.4230/LIPIcs.ESA.2019.33.
7. Graham Cormode and Andrew McGregor. Approximation algorithms for clustering uncertain data. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 191–200. ACM, 2008. doi:10.1145/1376916.1376944.
8. T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444. ACM, 1988. doi:10.1145/62212.62255.
9. Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In Jeff Erickson, editor, *Proceedings of the 23rd ACM Symposium on Computational Geometry, Gyeongju, South Korea, June 6-8, 2007*, pages 11–18. ACM, 2007. doi:10.1145/1247069.1247072.
10. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
11. Sudipto Guha and Kamesh Munagala. Exceeding expectations and clustering uncertain data. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 269–278. ACM, 2009. doi:10.1145/1559795.1559836.
12. Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. AMS, Boston, MA, USA, 2011.
13. Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004. doi:10.1145/1007352.1007400.
14. Sariel Har-Peled and Benjamin Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *J. ACM*, 62(6):44:1–44:35, 2015. doi:10.1145/2831230.
15. D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.
16. W. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.
17. Lingxiao Huang and Jian Li. Stochastic k-center and j-flat-center problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 110–129. SIAM, 2017. doi:10.1137/1.9781611974782.8.
18. Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 731–740. ACM, 2002. doi:10.1145/509907.510012.

19 Pegah Kamousi, Timothy M. Chan, and Subhash Suri. Closest pair and the post office problem for stochastic points. In *Algorithms and Data Structures - 12th International Symposium (WADS)*, volume 6844 of *Lecture Notes in Computer Science*, pages 548–559. Springer, 2011. doi:10.1007/978-3-642-22300-6_46.

20 Pegah Kamousi, Timothy M. Chan, and Subhash Suri. Stochastic minimum spanning trees in euclidean spaces. In *Proceedings of the 27th ACM Symposium on Computational Geometry (SoCG)*, pages 65–74. ACM, 2011. doi:10.1145/1998196.1998206.

21 Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k-center problems. *Theor. Comput. Sci.*, 242(1-2):237–245, 2000. doi:10.1016/S0304-3975(98)00222-9.

22 Nirman Kumar and Benjamin Raichel. Fault tolerant clustering revisited. In *Proceedings of the 25th Canadian Conference on Computational Geometry (CCCG)*. Carleton University, Ottawa, Canada, 2013. URL: http://cccg.ca/proceedings/2013/papers/paper_36.pdf.

23 Christiane Lammersen, Melanie Schmidt, and Christian Sohler. Probabilistic k-median clustering in data streams. *Theory Comput. Syst.*, 56(1):251–290, 2015. doi:10.1007/s00224-014-9539-7.

24 Alexander Munteanu, Christian Sohler, and Dan Feldman. Smallest enclosing ball for probabilistic data. In *30th Annual Symposium on Computational Geometry (SoCG)*, page 214. ACM, 2014. doi:10.1145/2582112.2582114.

25 Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y. Yip. Efficient clustering of uncertain data. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM)*, pages 436–445. IEEE Computer Society, 2006. doi:10.1109/ICDM.2006.63.

26 Haitao Wang and Jingru Zhang. One-dimensional k-center on uncertain data. *Theor. Comput. Sci.*, 602:114–124, 2015. doi:10.1016/j.tcs.2015.08.017.

27 William Henry Young. On classes of summable functions and their fourier series. *Proc. R. Soc. Lond. A*, 87:225–229, 1912. doi:10.1098/rspa.1912.0076.

Combinatorial and Algorithmic Aspects of Monadic Stability

Jan Dreier
TU Wien, Austria

Nikolas Mählmann
Universität Bremen, Germany

Amer E. Mouawad
American University of Beirut, Lebanon
Universität Bremen, Germany

Sebastian Siebertz
Universität Bremen, Germany

Alexandre Vigny
Universität Bremen, Germany

Abstract

Nowhere dense classes of graphs are classes of sparse graphs with rich structural and algorithmic properties, however, they fail to capture even simple classes of dense graphs. Monadically stable classes, originating from model theory, generalize nowhere dense classes and close them under transductions, i.e. transformations defined by colorings and simple first-order interpretations. In this work we aim to extend some combinatorial and algorithmic properties of nowhere dense classes to monadically stable classes of finite graphs. We prove the following results.

- For every monadically stable class $\mathscr{C}$ and fixed integer $s \geq 3$, the Ramsey numbers $R_{\mathscr{C}}(s,t)$ are bounded from above by $\mathcal{O}(t^{s-1-\delta})$ for some $\delta > 0$, improving the bound $R(s,t) \in \mathcal{O}(t^{s-1}/(\log t)^{s-1})$ known for the class of all graphs and the bounds known for k-stable graphs when $s \leq k$.
- For every monadically stable class $\mathscr{C}$ and every integer r, there exists $\delta > 0$ such that every graph $G \in \mathscr{C}$ that contains an r-subdivision of the biclique $K_{t,t}$ as a subgraph also contains K_{t^δ,t^δ} as a subgraph. This generalizes earlier results for nowhere dense graph classes.
- We obtain a stronger regularity lemma for monadically stable classes of graphs.
- Finally, we show that we can compute polynomial kernels for the independent set and dominating set problems in powers of nowhere dense classes. Formerly, only fixed-parameter tractable algorithms were known for these problems on powers of nowhere dense classes.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Theory of computation → Logic

Keywords and phrases Monadic Stability, Structural Graph Theory, Ramsey Numbers, Regularity, Kernels

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.11

Funding This paper is a part of a project that has received funding from the German Research Foundation (DFG) with grant agreement No 444419611.

1 Introduction

The notion of nowhere density was introduced by Nešetřil and Ossona de Mendez [22] and provides a very robust notion of uniform sparsity of graphs. These classes can be characterized in various ways, leading to a rich combinatorial theory and enabling the design

of efficient algorithms for problems that are hard in general. For example, the (distance-r) dominating set problem [6] and more generally, the first-order model-checking problem [16], is fixed-parameter tractable on nowhere dense graph classes.

Unfortunately, the techniques developed for nowhere dense classes are not applicable on dense graphs. However, there has been recent success of defining notions of *structural sparsity* by considering first-order transductions of sparse classes. Intuitively, a first-order transduction transforms graphs by first non-deterministically adding colors and then reinterpreting the edge relation using a first-order formula $\varphi(x, y)$, see e.g. [9, 14, 15, 21, 25]. This has led to the notion of *structurally nowhere dense* classes, which are first-order transductions of nowhere dense classes. If $\mathscr{C}$ is a nowhere dense class, then for example the class of all complements of graphs from $\mathscr{C}$ or for an integer d the class of all dth powers of graphs from $\mathscr{C}$ forms a structurally nowhere dense class.

It has been conjectured that structurally nowhere dense classes coincide with *monadically stable* classes (see e.g. [24]), which have long been studied in model theory [4]. This conjecture is based on the results of Adler and Adler [1], showing that every nowhere dense class is monadically stable and in fact for monotone classes (i.e. classes closed under taking subgraphs) the notions of nowhere density and monadic stability coincide. However, monadically stable classes are vastly more general than nowhere dense classes. Further indications for the truth of the conjecture is given by a recent characterization of monadically stable classes in terms of *flip-wideness* [10], which generalizes the notion of uniform quasi-wideness, which characterizes nowhere dense classes.

So far, the main tool for the study of monadically stable classes has been *indiscernible sequences* [10]. In our restricted context we focus on indiscernible sequences of vertices, that is, sequences $(v_i)_{1\leq i\leq n}$ of vertices such that all (increasing tuples of) vertices from the sequence satisfy the same first-order formulas. For example, an indiscernible sequence of vertices in a graph will induce an independent set or a clique. Furthermore, indiscernible sequences impose strong structural properties on their neighborhoods, which we exploit in our combinatorial and algorithmic study of monadically stable classes.

In the remainder of this section we describe our contribution in detail.

Ramsey numbers in monadically stable classes of graphs. Ramsey's theorem is a foundational result in combinatorics, stating in one of its graph theoretic formulations that in every sufficiently large graph one either finds a large clique or a large independent set. For a graph class $\mathscr{C}$, the Ramsey number $R_{\mathscr{C}}(s,t)$ denotes the smallest number of vertices that a graph from $\mathscr{C}$ must have such that one can either find a clique of size s or an independent set of size t. If $\mathscr{C}$ is the class of all graphs, we omit the subscript. Focusing on the case that s is fixed and t is growing, folklore bounds give us $R(s,t) \leq \binom{s+t-2}{s-1} \in \mathcal{O}_s(t^{s-1})$ (the subscript s in the asymptotic notation indicates that constant factors depending on s are hidden). For $s = 1, 2$ this bound is trivial and tight. The classical result of Ajtai, Komlós and Szemerédi [2] improves this for $s \geq 3$ to $R(s,t) \in \mathcal{O}_s(t^{s-1})/(\log t)^{s-2}$. We can rephrase this by saying that every K_s-free n-vertex graph with $s \geq 3$ contains an independent set of order $\Omega_s\left(n^{1/(s-1)} \cdot (\log n)^{(s-2)/(s-1)}\right)$. For every stable class $\mathscr{C}$ there exists an integer k (the ladder index, see Section 2 for formal definitions) such that every K_s-free n-vertex graph $G \in \mathscr{C}$ with $s \geq k-1$ contains an independent set of order $\Omega((n/k)^{1/(k+1)})$, strongly improving the general bound by removing the dependence on s from the exponent. These bounds are implicit in [20] and made explicit in [27, Observation 6.12]. However, on stable graph classes for $s < k-1$ the classical bounds of Ajtai et al. are the currently best known. Things look better on structurally nowhere dense classes: For every such class $\mathscr{C}$, $\epsilon > 0$ and

integer s, it holds that every K_s-free n-vertex graph from $\mathscr{C}$ contains an independent set of size $\Omega_{s,\epsilon,\mathscr{C}}(n^{1-\epsilon})$. This follows from the fact that graphs from structurally nowhere dense classes can be partitioned into $\mathcal{O}_{\mathscr{C},\epsilon,s}(n^{\epsilon})$ parts, each of which induces a cograph [9], and we have $R_{\text{co-graphs}}(s,t) = st$. (To be more precise, it is proved in [9] that each of the parts has bounded shrubdepth and it is proved in [26] that every graph of bounded shrubdepth can be decomposed into a bounded number of cographs.)

In light of the conjecture that every monadically stable class is a transduction of a nowhere dense class, we conjecture that also for every monadically stable class $\mathscr{C}$, $\epsilon > 0$ and integer s, it holds that every K_s-free n-vertex graph from $\mathscr{C}$ contains an independent set of size $\Omega_{\mathscr{C},\epsilon,s}(n^{1-\epsilon})$. We were unable to prove this conjecture, however, we were able to improve the bounds for monadically stable classes with ladder index k and $s < k-1$ by proving the following. If $\mathscr{C}$ is a K_s-free monadically stable class for some fixed $s \geq 3$, then there exists a real $\delta > 0$ such that in every n-vertex graph from $\mathscr{C}$ we can find an independent set of size $\Omega_{\mathscr{C}}(n^{1/(s-1)+\delta})$. This implies that we have $R_{\mathscr{C}}(s,t) \in \mathcal{O}_{\mathscr{C}}(t^{s-1-\delta})$ (Remark 3.2 below). To put this into perspective, Ajtai et al.'s classical result improves the size of an independent set in K_s-free graphs by a factor $\Theta((\log n)^{(s-2)/(s-1)})$ compared to the folklore bounds of $\Omega(n^{1/(s-1)})$ and our result improves it by a factor $\Theta(n^{\delta})$ for monadically stable graph classes.

Large bicliques in monadically stable classes of graphs. An r-subdivision of a graph H is a graph obtained by replacing the edges of H by paths of length (exactly) $r+1$ (containing r inner vertices). A class $\mathscr{C}$ of graphs is nowhere dense if and only if there exists a function f such that for every $r \geq 0$ the r-subdivision of a biclique (complete bipartite graph) $K_{f(r),f(r)}$ with parts of size $f(r)$ is excluded as a subgraph [23]. Hence, every class that is not nowhere dense must contain for some value of r arbitrarily large r-subdivided bicliques. It follows from [11] that every monadically stable class that is not nowhere dense even contains arbitrarily large bicliques as subgraphs. We demonstrate that these two quantities (size of r-subdivided bicliques and size of bicliques) are polynomially bounded in monadically stable classes, that is, we prove that for every monadically stable class $\mathscr{C}$ and every $r > 0$, there exist an integer m and a real $\delta > 0$, such that for every $t \geq m$ it holds that every graph $G \in \mathscr{C}$ that contains an r-subdivision of $K_{t,t}$ as a subgraph, also contains K_{t^δ,t^δ} as a subgraph.

Regularity lemma for monadically stable classes of graphs. Szemerédi's regularity lemma [30] is a celebrated result in extremal graph theory. Let us give the required formal definitions to formulate the lemma. Let G be a graph and let $A, B \subseteq V(G)$ be two disjoint non-empty subsets of vertices. We write $E(A,B)$ for the set of edges with one end in A and the other end in B. We define the *density* of the pair (A,B) as $\operatorname{dens}(A,B) := |E(A,B)|/|A||B|$. For a real $\epsilon > 0$, we call the pair (A,B) *ϵ-regular* if, for all subsets $A' \subseteq A$ and $B' \subseteq B$ with $|A'| \geq \epsilon|A|$ and $|B'| \geq \epsilon|B|$, we have $|\operatorname{dens}(A',B') - \operatorname{dens}(A,B)| \leq \epsilon$. A partition $V_1, \ldots, V_k$ of a set into disjoint parts is called an *equipartition* if $||V_i| - |V_j|| \leq 1$ for $1 \leq i < j \leq k$.

Szemerédi's Regularity Lemma [30] states that for every real $\epsilon > 0$ and integer $m \geq 1$ there exist two integers M and n_0 with the following property. For every graph G with $n \geq n_0$ vertices there exists an equipartition of the vertex set into k parts $V_1, \ldots, V_k$, $m \leq k \leq M$, such that all but at most ϵk^2 of the pairs (V_i, V_j) are ϵ-regular.

Irregular pairs cannot be avoided as witnessed by the example of ladders, see [3]. A *ladder* of order k is the bipartite graph H_k on vertices $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ with edges $\{a_i, b_j\}$ for $i \leq j$. For disjoint subsets X, Y of vertices of a graph G, we write $G[X,Y]$ for the subgraph of G semi-induced by X and Y, that is, the subgraph with vertex set $X \cup Y$

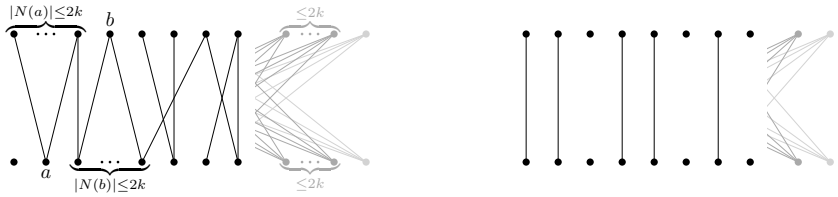


Figure 1 A possible connection between two parts for Malliaris and Shelah's regularity lemma (left) and our regularity lemma for monadically stable classes (right). In this example in both cases, two parts A and B form an independent set and are placed at the top and bottom, the local exceptional vertices ($2k$ or one) of each part are placed on the right in dark gray, and the additional single global exceptional vertex of each part is placed even further right in light gray. The defining statements are applied once from A to B and once from B to A.

and all the edges of G with one endpoint in X and one endpoint in Y. A bipartite graph H is a semi-induced subgraph of G if H is isomorphic to $G[X, Y]$ for some disjoint subsets X and Y of $V(G)$.

Malliaris and Shelah [20] proved that semi-induced ladders are the only reason for the exceptional pairs and proved a stronger regularity lemma (see [18] for an in-depth discussion of regularity lemmas for restricted graph classes). By giving up the requirement of partitioning into a constant number of parts, they proved that for every class $\mathscr{C}$ of graphs that exclude a ladder of order k as semi-induced subgraph there exist an integer n_0 and a real δ, such that for every graph $G \in \mathscr{C}$ of size $n \geq n_0$ there exists an equipartition into disjoint parts of size $\Omega(n^\delta)$ such that after possibly omitting one vertex from each part the following holds.

1. Every part forms a clique or an independent set.
2. For every two parts A and B, either for all vertices $a \in A$, except for at most $2k$, it holds that $|N(a) \cap B| \leq 2k$, or for all $a \in A$, except for at most $2k$, it holds that $|N(a) \cap B| \geq |B| - 2k$.

For every monadically stable class of graphs $\mathscr{C}$ we strengthen this result by replacing the at most $2k$ exceptional vertices by single exceptional vertices. More precisely, we prove that there exist an integer n_0 and a real δ, such that for every graph $G \in \mathscr{C}$ of size $n \geq n_0$ there exists an equipartition into disjoint parts of size $\Omega(n^\delta)$ such that after possibly omitting one vertex from each part the following holds (see Figure 1 for an illustration of both regularity lemmas).

1. Every part forms a clique or an independent set.
2. For every two parts A and B, either for all vertices $a \in A$, except for at most one, it holds that $|N(a) \cap B| \leq 1$, or for all $a \in A$, except for at most one, it holds that $|N(a) \cap B| \geq |B| - 1$. This means, with the exception of one vertex per part, the connection between A and B semi-induces either a subgraph of a matching or a supergraph of a co-matching.

Polynomial kernels for independent and dominating set in powers of nowhere dense graphs. Finally, we study algorithmic applications of structurally nowhere dense classes. A parameterized graph problem is called fixed-parameter tractable with respect to a parameter k if on input G and k it can be solved in time $f(k) \cdot |V(G)|^c$ for some function f and constant c. The independent set and dominating set problem are important algorithmic problems, which are unlikely to be fixed-parameter tractable in general graphs [8]. A kernel is a polynomial time algorithm that reduces an instance (G, k) to an equivalent instance (H, k') such that $|H| + |k'|$ is bounded by a function $g(k)$ only. In case g is a polynomial function, one

speaks of a polynomial kernel. For an integer d and a graph G, the dth power of G is the graph G^d on the same vertex set as G, where two (distinct) vertices are connected if their distance in G is at most d. A class $\mathscr{D}$ is a power of another class $\mathscr{C}$ if there exists d such that $\mathscr{D} = \{G^d \ : \ G \in \mathscr{C}\}$. It was recently shown that the independent set problem and dominating set problem are fixed-parameter tractable on powers of nowhere dense graphs [13], however no polynomial kernels were known. We complete the picture by proving that both problems admit polynomial kernels on powers of nowhere dense classes of graphs.

Note that the independent/dominating set problem on the dth power G^d of a graph G naturally corresponds to the following distance-d version of the problem on G. A *distance-d* independent set in a graph G is a set of vertices of pairwise distance greater than d, while a *distance-d dominating* set is a set of vertices such that every vertex of G has distance at most d to a vertex of the dominating set. The distance-d independent set problem and distance-d dominating set problem are fixed-parameter tractable on every nowhere dense class of graphs [6], and in fact admit polynomial kernels [12, 19]. However, note that our result is not implied by these results, as the difficulty in dealing with powers of nowhere dense classes stems from the fact that it is hard to compute for a given graph H a graph G such that $H = G^d$.

2 Preliminaries

We use standard notation from graph theory and model theory and refer to [7] and [17] for extensive background. We write $[m]$ for the set of integers $\{1, \dots, m\}$.

Graphs. Given a graph G, we denote its vertex set by $V(G)$ and its edge set by $E(G)$. For a vertex $u \in V(G)$ we define $N(u) := \{v \mid \{u, v\} \in E(G)\}$ to be the *open neighborhood* of u. A *partition* of $V(G)$ is a collection of pairwise disjoint, non-empty subsets whose union is $V(G)$. An *r-subdivision* of G is a graph obtained by subdividing each edge of G exactly r times. We call a bipartite graph H a *semi-induced subgraph* of G if H can be obtained from G by choosing two disjoint subsets A, B of $V(G)$, taking the subgraph of G induced by $A \cup B$ and removing all edges with both endpoints in A, respectively B. Let H be a bipartite graph with vertices $a_1, \dots, a_n$ and $b_1, \dots, b_n$. Then H is

- a *biclique* of order n if we have $\{a_i, b_j\} \in E(H)$ for all $i, j \in [n]$,
- a *matching* of order n if we have $\{a_i, b_j\} \in E(H) \iff i = j$ for all $i, j \in [n]$,
- a *co-matching* of order n if we have $\{a_i, b_j\} \in E(H) \iff i \neq j$ for all $i, j \in [n]$, and
- a *ladder* of order n if $\{a_i, b_j\} \in E(H) \iff i \leq j$ for all $i, j \in [n]$.

The *ladder index* of a graph G is the order of the longest ladder, which is contained as a semi-induced subgraph in G.

Monadic stability. A *transduction* (on graphs) is an operation mapping an input graph to a set of output graphs. A transduction is defined by an integer k and two first-order formulas $\varphi(x, y), \nu(x)$ over the language of vertex colored graphs, where φ is a symmetric and irreflexive formula (that is $\models \varphi(x, y) \leftrightarrow \varphi(y, x)$ and $\models \neg\varphi(x, x)$) and is the composition of the following operations. First, the input graph is mapped to k disjoint copies. Edges between the copies are created by pairwise connecting copies of the same vertex. The resulting graph is mapped to the set of all of its possible vertex colorings. At last, every colored graph G^+ is mapped to an uncolored graph H whose vertex set $\{v \in V(G^+) \mid G^+ \models \nu(v)\}$ is defined by ν and whose edge set $\{\{u, v\} \mid G^+ \models \varphi(u, v), u, v \in V(H)\}$ is defined by φ. Note that since φ is symmetric and irreflexive, it defines the edge set of an undirected graph.

This definition lifts to classes of graphs where we say a class $\mathscr{C}$ *transduces* a class $\mathscr{D}$ if there exists a fixed transduction T which produces from $\mathscr{C}$ all graphs from $\mathscr{D}$, i.e. $\mathscr{D} \subseteq \bigcup_{G \in \mathscr{C}} \mathsf{T}(G)$. We call a class *monadically stable* if it does not transduce the class of all ladders. Generalizing this notion, we call a class *monadically NIP* if it does not transduce the class of all graphs.

Transductions are transitive, i.e., for classes $\mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3$ we have that if $\mathscr{C}_1$ transduces $\mathscr{C}_2$ and $\mathscr{C}_2$ transduces $\mathscr{C}_3$, then also $\mathscr{C}_1$ transduces $\mathscr{C}_3$. Combining this observation with the fact that there exists a simple transduction yielding all induced subgraphs of an input graph, we will from now on assume without loss of generality that every monadically stable class is *hereditary*, i.e., closed under taking induced subgraphs.

Indiscernible sequences. Let G be a graph and $\varphi(x_1, \ldots, x_m)$ a formula. A sequence $(v_i)_{1 \leq i \leq n}$ of vertices from G is a *φ-indiscernible sequence* of length n, if for every two sequences of indices $i_1 < \cdots < i_m$ and $j_1 < \cdots < j_m$ from $[n]$ we have $G \models \varphi(v_{i_1}, \ldots, v_{i_m})$ if and only if $G \models \varphi(v_{j_1}, \ldots, v_{j_m})$. For a set of formulas Δ we call a sequence *Δ-indiscernible* if it is φ-indiscernible for all $\varphi \in \Delta$. In classical model theory, indiscernible sequences are usually defined over arbitrary structures, where they have infinite length and may consist of tuples of elements. However for our purpose we can restrict ourselves to finite sequences of vertices in graphs.

Indiscernible sequences have recently gained attention also in finite model theory and graph theory when Malliaris and Shelah showed that in *stable* structures we can find polynomially large indiscernible sequences [20, Theorem 3.5]. The extraction process was made algorithmic in [19, Theorem 2.8] and we now use the formulation of [10].

▶ **Theorem 2.1.** *Let $\mathscr{C}$ be a stable class of graphs and let Δ be a finite set of first-order formulas. Assume that we can evaluate, given $\varphi \in \Delta$, an n-vertex graph $G \in \mathscr{C}$ and a tuple $\bar{v}$ in G, whether $G \models \varphi(\bar{v})$ in time $f(\mathscr{C}, \Delta, n)$. Then there is an integer t and an algorithm that finds in any vertex-sequence of size at least m^t in an n-vertex graph $G \in \mathscr{C}$ a Δ-indiscernible sub-sequence of length m. The running time of the algorithm is $\mathcal{O}(|\Delta| \cdot k \cdot n^{1/k} \cdot f(\mathscr{C}, \Delta, n))$, where k is the maximal number of free variables of formulas in Δ.*

We refrain from formally defining the notion of stability and just note that every monadically stable class is also stable, hence, the theorem is applicable in our context.

Let $E(x, y)$ be the formula testing whether two vertices are adjacent in a graph. For $k \in \mathbb{N}$ we define $\Delta_k := \{E(x, y)\} \cup \{\gamma_k^S \mid S \subseteq [k]\}$ where for every $S \subseteq [k]$ we define

$$\gamma_k^S(y_1, \ldots, y_k) := \exists z \big(\bigwedge_{i \in S} E(z, y_i) \wedge \bigwedge_{i \notin S} \neg E(z, y_i) \big).$$

Since for every graph G, set $S \subseteq [k]$ and tuple $\bar{v} \in V(G)^k$, we can evaluate $G \models \gamma_k^S(\bar{v})$ in time $\mathcal{O}(k \cdot n)$, we derive the following corollary of Theorem 2.1.

▶ **Corollary 2.2.** *For every stable class of graphs $\mathscr{C}$ and integer k there exist an integer t and an algorithm that, given an n-vertex graph $G \in \mathscr{C}$ and a sequence $J \subseteq V(G)$ of length m^t, extracts a Δ_k-indiscernible sub-sequence of J of length m in time $\mathcal{O}(2^k \cdot k^2 \cdot n^2)$.*

Note that since the formula $E(x, y)$ is included in Δ_k, every Δ_k-indiscernible sequence is either a clique or an independent set. In [5], Blumensath studied the behaviour of indiscernible sequences in monadically NIP classes of infinite structures. Recently Dreier, Mählmann, Siebertz, and Toruńczyk finitized and extended those results with a special focus on monadically stable classes [10]. Their results are of very general nature and work for

arbitrary signatures and formulas. For our purposes we restrict their results to monadically stable classes of graphs using only the edge relation and distance formulas. Here we have that Δ_k-indiscernible sequences impose a strong structure on their neighborhood. The following follows from [10, Corollary 3.5].

▶ **Theorem 2.3.** *For every monadically stable class of graphs $\mathscr{C}$ there exist integers n, k such that for every Δ_k-indiscernible sequence I of length at least n in a graph $G \in \mathscr{C}$ and for every vertex $v \in V(G)$ we have that $|N(v) \cap I| \leq 1$ or $|N(v) \cap I| \geq |I| - 1$.*

They also prove strong properties for sequences of disjoint first-order definable sets. In our case we are only interested in disjoint balls of radius r around vertices. The following follows from [10, Theorem 4.2 and proof of Theorem 5.2].

▶ **Theorem 2.4.** *For every monadically stable class of graphs $\mathscr{C}$ and integer r, there exist integers k and t such that for every $m \in \mathbb{N}$, every graph $G \in \mathscr{C}$, and every set J of size m^t of disjoint r-balls in G the following holds. There exists a subset $I \subseteq J$ of size at least m with the following two properties.*

1. *There exists a set S of at most k sample vertices such that for every $v \in V(G)$ there exist a sample vertex $s(v) \in S$ and an exceptional ball $X(v) \in I$ such that for all $B \in I \setminus \{X(v)\}$ it holds that $N(v) \cap B = N(s(v)) \cap B$.*
2. *There exist a coloring $\mathrm{col} : V(G) \to [k]$ with k colors and a relation $R \subseteq [k]^2$ such that for every two vertices u, v from different r-balls in I we have that $G \models E(u, v)$ if and only if $\big(\mathrm{col}(u), \mathrm{col}(v)\big) \in R$.*

In the above theorem we can assume that for every sample vertex $s \in S$ we have that $N(s)$ intersect with every or none of the r-balls from I. This can be achieved by adding the vertices of S as constants to G and taking a Ψ-indiscernible subsequence I' of I where $\Psi(y) := \{\psi_s(y) = \exists z \in N(s) : \mathrm{dist}_{\leq r}(y, z) \mid s \in S\}$. By Theorem 2.1 the size of I' will still be polynomial.

3 Ramsey numbers in monadically stable graph classes

For a graph class $\mathscr{C}$, the Ramsey number $R_{\mathscr{C}}(s, t)$ denotes the smallest number of vertices that a graph from $\mathscr{C}$ must have such that one can either find a clique of size s or an independent set of size t. If $\mathscr{C}$ is the class of all graphs, we omit the subscript. As shown by Ajtai, Komlós and Szemerédi [2] we have $R(s, t) \in \mathcal{O}_s(t^{s-1}/(\log t)^{s-2})$. We improve these bounds in monadically stable classes by showing that for every fixed $s \geq 3$ and K_s-free monadically stable class $\mathscr{C}$ of graphs, there exists a constant $\delta > 0$ such that we have $R_{\mathscr{C}}(s, t) \in \mathcal{O}_{\mathscr{C}}(t^{s-1-\delta})$. The main result of this section is the following theorem.

▶ **Theorem 3.1.** *Let $s \geq 3$ and $\mathscr{C}$ be a K_s-free monadically stable class of graphs. There exist a real $\delta > 0$ and an integer n_0 such that every graph $G \in \mathscr{C}$ with $n \geq n_0$ vertices contains an independent set of size at least $n^{\frac{1}{s-1}+\delta}$.*

The aforementioned bound $R_{\mathscr{C}}(s, t) \in \mathcal{O}_{\mathscr{C}}(t^{s-1-\delta})$ then follows from the following remark.

▶ Remark 3.2. Assume we have a graph class $\mathscr{C}$, integers $s \geq 3$ and n_0 and a real $0 < \delta \leq 1$ such that every K_s-free graph from $\mathscr{C}$ with $n \geq n_0$ vertices contains an independent set of size at least $n^{\frac{1}{s-1}+\delta}$, that is, $R_{\mathscr{C}}(s, n^{\frac{1}{s-1}+\delta}) \leq n$ for all $n \geq n_0$. By setting $t = n^{\frac{1}{s-1}+\delta}$, we obtain $n = t^{\frac{s-1}{1+(s-1)\delta}}$. Basic arithmetic then yields $\frac{s-1}{1+(s-1)\delta} \leq s - 1 - \delta$ for $0 < \delta \leq 1$, $s \geq 3$ and therefore $n \leq t^{s-1-\delta}$. This means we have $R_{\mathscr{C}}(s, t) \leq t^{s-1-\delta}$.

The proof of Theorem 3.1 will be by induction on s. The heart of the proof is the base case $s = 3$, where $\mathscr{C}$ is triangle-free.

▶ **Lemma 3.3.** *For every triangle-free monadically stable class $\mathscr{C}$ of graphs there exist a real $\delta > 0$ and an integer n_0 such that every graph $G \in \mathscr{C}$ with $n \geq n_0$ vertices contains an independent set of size at least $n^{\frac{1}{2}+\delta}$.*

The core of the proof of Lemma 3.3 is to compute indiscernible sequences and reason about their neighborhoods using Theorem 2.3 and Theorem 2.4. For technical reasons we want the additional property that the indiscernible sequences have a "large" neighborhood. We first show that if we cannot find such indiscernible sequences, then the graph contains a large independent set.

Proof of Lemma 3.3. Let us fix a graph $G \in \mathscr{C}$. We greedily build an independent set $\mathcal{I}$ as follows. We look for an independent set I of size $\lceil (n/2)^\delta \rceil$ with $|N(I) \setminus I| \leq n^{\frac{1+\delta}{2}}$, where the value $0 < \delta \leq 1$ depends only on $\mathscr{C}$ and will be specified later. Then we place the vertices of I into the independent set $\mathcal{I}$ and remove I and its neighborhood from the graph, since these vertices are not allowed to be chosen into $\mathcal{I}$ anymore. We repeat this process as long as possible. Let us first show that the statement of the lemma follows if we can repeat this process as long as there are still at least $n/2$ vertices in the graph. In each iteration we remove at most $\lceil (n/2)^\delta \rceil + n^{\frac{1+\delta}{2}}$ vertices from the graph, add $\lceil (n/2)^\delta \rceil$ vertices to the independent set $\mathcal{I}$ and do not terminate unless $n/2$ vertices are removed. We choose n_0 to be large enough such that with $n \geq n_0$ it holds that $4\lceil (n/2)^\delta \rceil + 4n^{\frac{1+\delta}{2}} \leq n^{\frac{1+1.1\delta}{2}}$. The process therefore yields an independent set $\mathcal{I}$ of size at least

$$\lceil (n/2)^\delta \rceil \cdot \frac{n/2}{\lceil (n/2)^\delta \rceil + n^{\frac{1+\delta}{2}}} \geq \frac{n^{1+\delta}}{4\lceil (n/2)^\delta \rceil + 4n^{\frac{1+\delta}{2}}} \geq \frac{n^{1+\delta}}{n^{\frac{1+1.1\delta}{2}}} = n^{\frac{1+0.9\delta}{2}}$$

and the statement of the lemma follows by rescaling δ.

From now on, we can therefore assume that the process could not proceed, while there were still at least $n/2$ vertices in the graph. Thus, we have a subgraph G' of G with at least $n/2$ vertices and every independent set I in G' of size $\lceil (n/2)^\delta \rceil$ has $|N(I) \setminus I| \geq n^{\frac{1+\delta}{2}}$ (later on referred to as assumption $(*)$).

We search for a Δ_{k_1}-indiscernible sequence I_0 in G' with k_1 chosen as required by Theorem 2.3. By Corollary 2.2, I_0 can be chosen of length at least $(n/2)^{\delta_1}$ for some real δ_1 depending only on $\mathscr{C}$.

By triangle-freeness, I_0 is not a clique but an independent set. Let G'_1 be the induced subgraph of G' where all vertices adjacent to at least two elements of I_0 have been removed. The radius 1-balls around the elements of I_0 in G'_1 are disjoint and we can apply Theorem 2.4, yielding a real δ_2 and an integer k_2, both depending only on $\mathscr{C}$, with the following properties. There exists a subset $I \subseteq I_0$ of size at least $(n/2)^\delta$ for $\delta := \delta_1 \cdot \delta_2$ together with a k_2-coloring $\mathcal{K}$ of the 1-balls around I in G'_1 and a relation $R \subseteq [k_2]^2$, completely determining the edge relation between vertices from different balls.

We go back to the graph G' where we partition $N(I)$ into two sets $B_1 \cup B_2$, where B_1 contains all vertices that are adjacent to exactly one vertex in I and B_2 contains all vertices that are adjacent to at least two vertices in I. Note that in G' the coloring $\mathcal{K}$ still describes the adjacency between elements from B_1 with different neighbors in I. Since I is a sub-sequence of the Δ_{k_1}-indiscernible sequence I_0, Theorem 2.3 still applies and all vertices in B_2 are adjacent to at least $|I| - 1$ vertices of I. This means any two vertices $u, v \in B_2$ have a common neighbor in I and since G' is triangle-free, there can be no edge between u and v, and B_2 forms an independent set. We can assume $|B_2| \leq \frac{1}{2} n^{\frac{1+\delta}{2}}$, since otherwise

the statement of the lemma follows. As argued before I is also an independent set. Thus, by assumption $(*)$, $|N(I) \setminus I| \geq n^{\frac{1+\delta}{2}}$. In summary, we so far have $N(I) \setminus I = B_1 \cup B_2$, $|N(I) \setminus I| \geq n^{\frac{1+\delta}{2}}$ and $|B_2| \leq \frac{1}{2} n^{\frac{1+\delta}{2}}$. This means $|B_1| \geq \frac{1}{2} n^{\frac{1+\delta}{2}}$.

We pick the color class $K \in \mathcal{K}$ containing the most vertices of B_1. It has size at least $|K| \geq |B_1|/k_2 \geq \frac{1}{2k_2} n^{\frac{1+\delta}{2}}$. For $v \in I$ let $K_v = K \cap N(v)$. Since every vertex in B_1 has exactly one neighbor in I, this describes a partition of K. For every $v \in I$, the vertices in K_v have a common neighbor and thus form (by triangle-freeness of G) an independent set. If the relation R does not contain (K, K) then vertices from different sets K_v and K_w are not connected for all $v \neq w \in I$ and therefore K forms an independent set. On the other hand, if the relation R contains (K, K) then vertices from different sets K_v and K_w are connected for all $v \neq w \in I$. Then there can be at most two vertices $v \in I$ with $K_v \neq \varnothing$, since otherwise there would be a triangle. Hence, there exists $v \in I$ with $|K_v| \geq |K|/2$. In either case, we find an independent set of size at least $|K|/2 \geq \frac{1}{4k_2} n^{\frac{1+\delta}{2}}$. The statement of the lemma now follows by rescaling δ. ◀

We now turn to the proof of Theorem 3.1. The proof is by induction with Lemma 3.3 as the base case.

Proof of Theorem 3.1. If $s = 3$, the claim follows from Lemma 3.3. For larger values of s, we prove the claim by induction. The class $\mathscr{C}'$ of K_{s-1}-free induced subgraphs of $\mathscr{C}$ is monadically stable. By induction, there exist $0.1 \geq \delta' > 0$ and n_0' such that every graph in $\mathscr{C}'$ with $n \geq n_0'$ vertices contains an independent set of size at least $n^{\frac{1}{s-2}+\delta'} \geq n^{\frac{1+\delta'}{s-2}}$. Note that the upper bound of 0.1 for δ' is no restriction: if the statement holds for $\delta' > 0.1$, then it also holds for $\delta' = 0.1$. Let $G \in \mathscr{C}$ be a graph of size $n \geq n_0 \geq n_0'$, where we specify n_0 later. Let $\beta = (1 - \delta'/2)\frac{s-2}{s-1}$. We distinguish two cases.

First, assume there is a vertex v with degree at least n^β. Then the induced subgraph $G[N(v) \setminus \{v\}]$ has size at least n^β and is K_{s-1}-free. Since it is contained in $\mathscr{C}'$, there is an independent set in $G[N(v) \setminus \{v\}]$ (and therefore also in G) of size at least $n^{\beta \frac{1+\delta'}{s-2}}$.
Since $\delta' \leq 0.1$, we have $(1 - \delta'/2)(1 + \delta') = 1 + \delta'/2 - \delta'^2/2 \geq 1 + \delta'/3$ and therefore:

$$\beta \frac{1+\delta'}{s-2} = (1 - \delta'/2) \cdot \frac{s-2}{s-1} \cdot \frac{1+\delta'}{s-2} = (1 - \delta'/2)(1 + \delta') \cdot \frac{1}{s-1} \geq \frac{1}{s-1} + \frac{\delta'/3}{s-1}.$$

We set $\delta = (\delta'/3)/(s-1)$. Then G contains an independent set of size at least $n^{\frac{1}{s-1}+\delta}$.

If on the other hand there is no vertex with degree at least n^β then all vertices have degree less than n^β. In this case, we greedily construct an independent set by repeatedly picking a vertex and removing it together with its neighbors from the graph. This yields an independent set of size at least $n/(n^\beta + 1)$. We choose n_0 to be large enough such that with $n \geq n_0$ it holds that $n^\beta + 1 \leq n^{(1-\delta'/3)\frac{s-2}{s-1}}$. Then G contains an independent set of size at least $n/(n^\beta + 1) \geq n^{1-(1-\delta'/3)\frac{s-2}{s-1}} = n^{\frac{1}{s-1}+(\delta'/3)\frac{s-2}{s-1}}$. The claim follows with $\delta = (\delta'/3)\frac{s-2}{s-1}$. ◀

4 Subdivided bicliques

For every $r, s, t \in \mathbb{N}$ denote by $\mathrm{sd}_r(K_{s,t})$ the r-subdivision of the complete bipartite graph with partitions of size s and t. When dealing with subdivisions, we will call the vertices of the original graph *principal vertices* and the newly inserted vertices *subdivision vertices*. In this section we prove the following theorem.

▶ **Theorem 4.1.** *For every monadically stable class $\mathscr{C}$ and every integer r, there exist an integer n_0 and a real $\delta > 0$, such that for every $n \geq n_0$ it holds that every graph $G \in \mathscr{C}$ that contains $\mathrm{sd}_r(K_{n,n})$ as a subgraph, also contains $K_{\lceil n^\delta \rceil, \lceil n^\delta \rceil}$ as a subgraph.*

Proof. For the class $\mathscr{C}$ and radius r, Theorem 2.4 yields a sample set size bound $k \in \mathbb{N}$ and an exponent $\frac{1}{\lambda} \in \mathbb{N}$ (referred to as t in Theorem 2.4) bounding the size of the set required to apply Theorem 2.4. Define $t_0 := n$ and $t_{i+1} := \frac{1}{2}t_i^{\lambda}$. Choose n_0 and δ such that it holds for all $n \geq n_0$ and $0 \leq i \leq r+1$ that $k \cdot n^{\delta} \leq \frac{1}{2}t_i$.

We work in the induced subgraph H of G that only contains the vertices of the $\mathrm{sd}_r(K_{n,n})$. Let A and B be the sets of principal vertices in the $\mathrm{sd}_r(K_{n,n})$. Assume towards a contradiction, that H contains no $K_{\lceil n^{\delta} \rceil, \lceil n^{\delta} \rceil}$ as a subgraph.

▷ Claim 4.2. For every $0 \leq i \leq r+1$, H contains an induced subgraph H_i that contains as a subgraph $\mathrm{sd}_r(K_{\lceil t_i \rceil, \lceil t_i \rceil})$, with sets of principal vertices $A_i \subseteq A$ and $B_i \subseteq B$ such that the vertices from A_i have disjoint distance-i neighborhoods.

Proving the above claim will immediately lead to a contradiction, as the distance-$(r+1)$ neighborhoods of any two principal vertices in A_{r+1} overlap in every vertex of B_{r+1} as they are the principal vertex of an r-subdivided biclique. We prove the claim by induction on i.

Proof. For the base case set $H_0 := H$, $A_0 = A$, and $B_0 = B$. For the inductive step we are given H_i such that the vertices of A_i have disjoint distance-i neighborhoods. They therefore form the centers of a set J of disjoint i-balls and we can apply Theorem 2.4. We obtain a subset $I \subseteq J$ of size at least $|A_i|^{\lambda}$ together with a set S of at most k sample vertices such that for every $v \in H_i$, there exist a sample $s(v) \in S$ and an exceptional ball such that $N(v) \cap B = N(s(v))$ for every ball $B \in I$ which is not the exceptional ball. Let $s^{-1}(s) := \{v \in V(H_i) \mid s(v) = s\}$ be the set of all vertices described by $s \in S$. Let S_I be subset of sample vertices from S whose neighborhood intersects all of the balls from I. Conversely let $S_{\varnothing}$ be subset of sample vertices from S whose neighborhood intersects none of the balls from I. As argued in Section 2 following Theorem 2.4 we have that $S = S_I \cup S_{\varnothing}$. We note that one can also assume, without loss of generality, that $|S_{\varnothing}| \leq 1$.

We now argue that $s^{-1}(s)$ has less than n^{δ} elements for every $s \in S_I$. Towards a contradiction, assume otherwise and let Q be a subset of $s^{-1}(s)$ of size $\lceil n^{\delta} \rceil$. Since every vertex from Q can behave exceptionally to at most one ball from I and we have that $|I| \geq 2\lceil n^{\delta} \rceil$, we find a subset of balls centered at $I' \subseteq I$ of size $\lceil n^{\delta} \rceil$ in which every vertex from Q has the same neighborhood as s. Since s has at least one neighbor in every ball of I' this yields a biclique of size $\lceil n^{\delta} \rceil$; a contradiction.

It follows that the set $S_I^{-1} := \bigcup_{s \in S_I} s^{-1}(s)$ of vertices described by sample vertices from S_I has size less than $k \cdot n^{\delta}$. Next we build the induced subgraph H_{i+1} of H_i by removing the vertices of S_I^{-1}. Let $A_i' \subseteq A_i$ be the center vertices of the balls in I. After the deletion of S_I^{-1} we have that every vertex in $v \in V(H_{i+1})$ is described by a sample vertex from $S_{\varnothing}$. Since the neighborhoods of the vertices in $S_{\varnothing}$ intersect none of the i-balls in I, the neighborhood of v can intersect at most one of the i-balls in I (i.e. the exceptional ball of v). It follows that the vertices from A_i' have disjoint $(i+1)$-balls.

It remains to show that there exist sets $A_{i+1} \subseteq A_i'$ and $B_{i+1} \subseteq B_i$ forming the principal vertices of a $\mathrm{sd}_r(K_{\lceil t_{i+1} \rceil, \lceil t_{i+1} \rceil})$ subgraph in H_{i+1}. We started with the graph H_i containing as a subgraph $\mathrm{sd}_r(K_{\lceil t_i \rceil, \lceil t_i \rceil})$ with principal vertices A_i and B_i. It follows that H_i contains as a subgraph $\mathrm{sd}_r(K_{|A_i'|, \lceil t_i \rceil})$ with principal vertices A_i' and B_i. It holds that for every $t \geq 2$, every graph obtained by removing a single vertex from $\mathrm{sd}_r(K_{t,t})$ still contains $\mathrm{sd}_r(K_{t-1,t-1})$ as a subgraph. Therefore after we removed S_I^{-1} from H_i to build H_{i+1} we still find in it a set $A_{i+1} \subseteq A_i'$ of size $|A_{i+1}| \geq |A_i'| - |S_I^{-1}| \geq |A_i|^{\lambda} - |S_I^{-1}| \geq t_i^{\lambda} - k \cdot n^{\delta} \geq \frac{1}{2}t_i^{\lambda} = t_{i+1}$ forming one side of a $\mathrm{sd}_r(K_{\lceil t_{i+1} \rceil, \lceil t_{i+1} \rceil})$ subgraph in H_{i+1}. The other side is formed by a subset of B_i. ◁

This completes the proof of the theorem. ◀

5 Regularity in monadically stable classes of graphs

In this section we prove the following theorem.

▶ **Theorem 5.1.** *For every monadically stable class of graphs $\mathscr{C}$, there exist an integer n_0 and a real δ, such that for every graph $G \in \mathscr{C}$ of size $n \geq n_0$ there exists an equipartition into parts of size n^δ, such that after possibly omitting one element from each part, they satisfy the following properties:*

1. *Every part forms a clique or an independent set.*
2. *For every two parts A and B, except for at most one vertex $a_{\text{ex}} \in A$, for all other vertices $a \in A$, either $|N(a) \cap B| \leq 1$ or $|N(a) \cap B| \geq |B| - 1$. This means, with the exception of one vertex per part, the connections between A and B semi-induce either a subgraph of a matching or a supergraph of a co-matching.*

Note that the additional vertex that needs to be omitted from a part A when comparing it to a part B may differ from the vertex that needs to be omitted from A when comparing it to a different part C.

We follow the approach by Malliaris and Shelah, by partitioning the graph into indiscernible sequences. We start by analyzing the interaction between two indiscernible sequences.

▶ **Lemma 5.2.** *For every monadically stable class of graphs $\mathscr{C}$, there exist integers n_0 and k such that for every graph $G \in \mathscr{C}$ the following holds. Let A and B be disjoint Δ_k-indiscernible sequences of length at least n_0 in G. Except for at most one vertex $a_{\text{ex}} \in A$, for all other vertices $a \in A$, either $|N(a) \cap B| \leq 1$ or $|N(a) \cap B| \geq |B| - 1$. In particular, after removing at most one vertex from each sequence, the connections between A and B semi-induce either a subgraph of a matching or a supergraph of a co-matching.*

Proof. Choose $n_0 \geq 5$ and k as required by Theorem 2.3.

▷ Claim 5.3. The set of vertices of A adjacent to at least two vertices from B has size 0,1, $|A| - 1$, or $|A|$.

Proof. Assume towards a contradiction that there exist two elements $a_1, a_2 \in A$ adjacent to at least two vertices in B and two vertices $a_3, a_4 \in A$ adjacent to at most one vertex in B. Since a_1, a_2 have at least two neighbors in B, by Theorem 2.3 they both must have at least $|B| - 1$ neighbors in B. Therefore there must be a set $B' \subseteq B$ of size at least $|B| - 2 \geq 3$ containing only vertices adjacent to both a_1 and a_2. Again by Theorem 2.3 every vertex in B' must have at least $|A| - 1$ neighbors in A. As a result either a_3 or a_4 will have at least two neighbors in B' and therefore in B and we reach a contradiction. ◁

Let $A' \subseteq A$ be the set of vertices adjacent to at least two vertices from B. It has size 0,1, $|A| - 1$, or $|A|$. Assume $|A'| = 0$. All vertices of A have at most one neighbor in B and we have at most $|A|$ edges between the two sets. If no two vertices have the same neighbor, then we have a subgraph of a matching between A and B and we are done. Assume two vertices of A share a neighbor b in B. By Theorem 2.3, b must have at least $|A| - 1$ neighbors in A. We therefore have at most one edge between A and $B \setminus \{b\}$, a subgraph of a matching as required.

If $|A'| = 1$, only one vertex $a \in A$ has at least two neighbors in B we omit a from A and argue as in the previous case. The two cases where $|A'| \geq |A| - 1$ follow by symmetry. ◀

Proof of Theorem 5.1. Let n_0' and k be from Lemma 5.2 and let $\lambda \leq \frac{1}{2}$ be the real given by Corollary 2.2 such that any graph from $\mathscr{C}$ of size m contains a Δ_k-indiscernible sequence of size at least m^λ. Therefore, we can iteratively build a partition $\mathcal{G}$ of Δ_k-indiscernible sequences of size $n^{\frac{\lambda}{2}}$ in G as long as $|V(G) \setminus \bigcup \mathcal{G}| = n - |\bigcup \mathcal{G}| \geq n^{\frac{1}{2}}$. Once we have that $|\bigcup \mathcal{G}| > n - n^{\frac{1}{2}}$, we know that $\mathcal{G}$ contains at least

$$\left\lfloor \frac{n - n^{\frac{1}{2}}}{n^{\frac{\lambda}{2}}} \right\rfloor = \left\lfloor n^{1-\frac{\lambda}{2}} - n^{\frac{1}{2}-\frac{\lambda}{2}} \right\rfloor \geq n^{\frac{1}{2}}$$

many parts. Therefore we can distribute the leftover elements from $V(G) \setminus \bigcup \mathcal{G}$ among the parts of $\mathcal{G}$, such that every part of $\mathcal{G}$ gets at most one additional vertex. Now $\mathcal{G}$ is the desired equipartition with part size at least n^δ for $\delta := \frac{\lambda}{2}$. By omitting the leftover element property (1) follows from the fact that any Δ_k-indiscernible sequence is either a clique or an independent set and property (2) follows from Lemma 5.2 and by setting $n_0 \geq {n_0'}^{\frac{1}{\delta}}$. ◀

6 Polynomial kernels in powers of nowhere dense classes

In this section we turn to algorithmic applications of indiscernible sequences in monadically stable classes of graphs. We focus on powers of classes of nowhere dense graphs. Recall that the dth power G^d of a graph G is obtained by pairwise connecting in G vertices that have distance at most d. This notion is lifted to classes of classes by defining $\mathscr{C}^d := \{G^d \mid G \in \mathscr{C}\}$ to be the dth power of a class $\mathscr{C}$.

Powers of nowhere dense graphs were studied by Fabiański et al. who showed that they admit fixed-parameter tractable algorithms for Independent Set and Dominating Set parameterized by solution size k [13]. We will strengthen their result by giving polynomial kernels for both problems and prove the following theorem.

▶ **Theorem 6.1.** *For every integer d and nowhere dense class $\mathscr{C}$, Independent Set and Dominating Set parameterized by solution size k admit a polynomial kernel on $\mathscr{C}^d$.*

To obtain this result we will make use of the following two properties of powers of nowhere dense graphs. For every integer d and nowhere dense class $\mathscr{C}$, the class $\mathscr{C}^d$ (1) is monadically stable, and (2) excludes a fixed co-matching as a semi-induced subgraph. For every integer d the operation of taking the dth power of a graph is expressible as a transduction. Therefore the first property follows trivially from the monadic stability of nowhere dense classes. The second property was proved by Fabiański et al. in [13, Theorem 11]. Assuming those two properties, we will construct our kernels. In fact, our results apply to all graph classes satisfying these two properties, with powers of nowhere dense classes merely being the most prominent example.

▶ **Theorem 6.2.** *Independent Set parameterized by solution size k admits a polynomial kernel on every monadically stable class of graphs which excludes a co-matching of size ℓ as a semi-induced subgraph.*

Proof. We search for an independent set of size k in a graph G, which we assume to be from a monadically stable class $\mathscr{C}$ that excludes a co-matching of size ℓ. By Theorem 2.3, for every monadically stable class of graphs $\mathscr{C}$ there exist integers n_0 and p such that for every Δ_p-indiscernible sequence I of length at least n_0 in a graph $G \in \mathscr{C}$ and for every vertex $v \in V(G)$ we have that $|N(v) \cap I| \leq 1$ or $|N(v) \cap I| \geq |I| - 1$. We fix such a pair of integers n_0 and p, as given by Theorem 2.3. Now by Corollary 2.2, there exist an integer t and an algorithm that, given an n-vertex graph $G \in \mathscr{C}$, where $n \geq n_0^t$, extracts a Δ_p-indiscernible

sequence $I \subseteq V(G)$ of length n_0 in time $\mathcal{O}(2^p \cdot p^2 \cdot n^2)$. We now have that in any graph $G \in \mathscr{C}$ of size at least n_0^t, we can find in time $\mathcal{O}(2^p \cdot p^2 \cdot n^2)$ a Δ_p-indiscernible sequence I of size at least $n^{1/t}$ such that, by Theorem 2.3, every vertex of $V(G) \setminus I$ is adjacent to at most one vertex from I or all but at most one vertex from I. Recall that since we include the edge relation in Δ_p, every I is either a clique or an independent set.

If $n \leq (k+1)^t$, then the graph itself is a polynomial kernel, so we can assume without loss of generality that $n > (k+1)^t$. This means that we can find in any graph $G \in \mathscr{C}$ of size at least $(k+1)^t$ in time $\mathcal{O}(2^p \cdot p^2 \cdot n^2)$ a Δ_p-indiscernible sequence I of size at least $k+1$. We will (for later) similarly assume, without loss of generality, that $k+1 \geq \ell$; since otherwise we can solve the problem in polynomial time, i.e., $\mathcal{O}(n^\ell)$-time, and return a trivial yes- or no-instance. Our kernelization algorithm proceeds by applying the following reduction rule exhaustively.

> If G has at least $(k+1)^t$ vertices, find a Δ_p-indiscernible sequence I of size at least $k+1$. If I is an independent set return a trivial yes-instance. Otherwise remove from G all vertices of I except for k many.

Since $\mathscr{C}$ is closed under taking induced subgraphs, we can apply this rule repeatedly without leaving $\mathscr{C}$. The procedure runs in polynomial time and obviously, upon termination, the resulting graph has size at most $(k+1)^t$. Since the procedure only removes vertices, it cannot turn no-instances into a yes-instances. It remains to argue that the procedure does not turn yes-instances into no-instances.

If I forms an independent set, then we return a trivial yes-instance, so the only remaining case is I forming a clique. In this case, I can contain at most one vertex from a solution. Assume there exists a vertex outside I that is non-adjacent to only one element of I. By indiscernibility of I, there exists such a "private vertex" for every element in I which implies a co-matching of size at least $k+1 \geq \ell$. This is a contradiction to our assumption of excluding a co-matching of size ℓ. Therefore, by Theorem 2.3, every vertex outside I is adjacent to either all or at most one vertex of I.

Assume we have a yes-instance, witnessed by an independent set S of size k. Since I is a clique, I contains at most one vertex from S. If S contains no vertex from I, then after applying the reduction rule we trivially still have a yes-instance. On the other hand, if S contains exactly one vertex from I then the remaining $k-1$ vertices outside I are not adjacent to all vertices in I and therefore have at most one neighbor each within I. This means in any subset of k vertices of I there will be at least one vertex that is non-adjacent to the remaining $k-1$ vertices of S, completing it to an independent set of size k and preserving the yes-instance. ◀

▶ **Theorem 6.3.** *Dominating Set parameterized by solution size k admits a polynomial kernel on every monadically stable class of graphs which excludes a co-matching of size ℓ as a semi-induced subgraph.*

Proof. We search for a dominating set of size k in a graph from a monadically stable graph class $\mathscr{C}$ which excludes a co-matching of size ℓ as a semi-induced subgraph. We outline our approach: Given a Dominating Set instance (G_0, k), we will first construct an equivalent instance (G, k) of the Red Blue Dominating Set problem. The input to this problem is a bipartite graph G whose two parts are colored red and blue, respectively, and a number k, and the task is to find k blue vertices that dominate all red vertices. We will ensure that G is still part of another monadically stable class $\mathscr{D}$ that excludes a co-matching. We then

kernelize (G, k) to obtain a kernel (K, k) to Red Blue Dominating Set whose size is polynomial in k. Finally we will transform (K, k) back into a Dominating Set instance $(K_0, k+1)$ by only introducing $\mathcal{O}(k)$ many new vertices.

From Dominating Set to Red Blue Dominating Set. We create the instance (G, k) from (G_0, k) as follows. The vertices of G consist of a red copy A and a blue copy B of $V(G_0)$. For every vertex $v \in V(G_0)$ we connect the red copy of v with each blue copy in $N[v]$. Note that no vertices of the same color are connected. There exists a fixed transduction which produces G from G_0, so we know that G must be from a monadically stable class $\mathscr{D}$ depending only on $\mathscr{C}$. It is easy to see that G_0 contains a size k dominating set if and only if there exist k vertices from A which together dominate all of B. Now assume towards a contradiction that G contains a co-matching C of size 3ℓ as a semi-induced subgraph.

We build a semi-induced co-matching C_0 in G_0 by iteratively taking a red vertex a and a blue vertex b from C that are not connected and adding them to C_0. Since a and b are non-neighbors in different colors, we know that they are not connected and correspond to different vertices in G_0. If the blue copy of a or the red copy of b is contained in C we remove them together with their co-matching partners from C. In G_0, a will be connected to the original of every blue vertex in C except for b, and analogously b will be connected to the original of every red vertex in C except a. In every step, we add one pair to C_0 and remove at most three pairs from C. Therefore by fully exhausting C we find a semi-induced co-matching of size at least ℓ in C, a contradiction. It follows that G contains no co-matching of size 3ℓ.

Kernelization. By Theorem 2.3, for every monadically stable class of graphs $\mathscr{D}$ there exist integers n_0 and p such that for every Δ_p-indiscernible sequence I of length at least n_0 in a graph $G \in \mathscr{D}$ and for every vertex $v \in V(G)$ we have that $|N(v) \cap I| \leq 1$ or $|N(v) \cap I| \geq |I| - 1$. We fix such a pair of integers n_0 and p, as given by Theorem 2.3. Now by Corollary 2.2, there exist an integer t and an algorithm which given an n-vertex graph $G \in \mathscr{D}$, where $n \geq n_0^t$, extracts a Δ_p-indiscernible sequence $I \subseteq V(G)$ of length n_0 in time $\mathcal{O}(2^p \cdot p^2 \cdot n^2)$. We now have that in any graph $G \in \mathscr{D}$ of size at least n_0^t, we can find in time $\mathcal{O}(2^p \cdot p^2 \cdot n^2)$ a Δ_p-indiscernible sequence I of size at least $n^{1/t}$ such that, by Theorem 2.3, every vertex of $V(G) \setminus I$ is adjacent to at most one vertex from I or all but at most one vertex from I.

We assume (similarly to Theorem 6.2), without loss of generality, that $k + 2 \geq 3\ell$, $k > 1$, $|B| > (k+2)^t$. Thus, if the set of to-be-dominated vertices B contains at least $(k+2)^t$ vertices, there exists a Δ_p-indiscernible sequence I of size at least $k+2$ among B such that by Theorem 2.3, all vertices in A are connected to at most one or all but at most one vertex in I. We will kernelize it using the following reduction rules.

1. If A contains two vertices with the same neighborhood in B, remove one of them.
2. If the set of to-be-dominated vertices B contains at least $(k+2)^t$ vertices, find a Δ_p-indiscernible sequence I of size at least $k+2$ among B. Remove from G all vertices of I except for $k+1$ many.

The first reduction rule is obviously efficient and safe. As discussed above, the second rule can be implemented in polynomial time. Let (G', k) be the instance obtained from (G, k) by the second rule, where a Δ_p-indiscernible sequence I is reduced to a sub-sequence I' of size $k+1$. If (G, k) is a yes-instance, then removing to-be-dominated vertices from B will still yield a yes-instance. To show safety of the second rule, we assume that the reduced

instance (G', k) is a yes-instance and show that (G, k) also is a yes-instance. Since (G', k) is a yes-instance, there exists a set $S \subseteq A$ of size at most k that dominates all of I'. As discussed above, by Theorem 2.3 every vertex in S is connected to at most one, all, or all but one vertex of I. Assume for contradiction there exists a vertex connected to all but one vertex of I. By indiscernibility, one vertex of I having a private non-neighbor would mean that all vertices of I have a private non-neighbor, implying the existence of a co-matching of size $|I| \geq k + 2 \geq 3\ell$, a contradiction. Therefore every vertex in S is either connected to at most one or all vertices of I. Since I' has $k + 1$ vertices and S only k, S must include a vertex connected to at least two elements of I and therefore connected to all of I. Hence, S is also a solution to (G, k), meaning that (G, k) is a yes-instance

It remains to prove that the kernel (K, k) obtained by exhaustively applying the two rules has size polynomial in k. Both rules reduce the size of the graph and when applied exhaustively, B has size at most $(k+2)^t$ and A contains no more twins. We use the following theorem which applies to *NIP* classes which contain monadically NIP and therefore also monadically stable classes.

▶ **Theorem 6.4** (Sauer-Shelah Lemma [28, 29, 31]). *Let $G \in \mathscr{D}$ and let $B \subseteq V(G)$. Let $\mathcal{P} = \{P_1, P_2, \ldots\}$ be a partition of $V(G) \setminus B$ such that two vertices u and v belong to $P \in \mathcal{P}$ whenever $N(u) \cap B = N(v) \cap B$. If $\mathscr{D}$ is a NIP class of graphs then there exists a constant d such that $|\mathcal{P}| \leq |B|^d + 1$.*

Since we have no twins in A, every vertex in $a \in A$ is uniquely identified by its neighborhood $N(a) \cap B$ and therefore $|A| \leq |B|^d + 1 \leq (k + 2)^{t \cdot d} + 1$. Thus, (K, k) has polynomial size.

From Red Blue Dominating Set to Dominating Set. Finally we are going to convert (K, k) into a Dominating Set instance $(K_0, k + 1)$ by creating a star $K_{1,k+2}$ of size $k + 2$ and connecting its center c to every red (i.e. dominating) vertex from K.

If (K, k) admits a red blue dominating set S of size k, then $S \cup \{c\}$ is a $k + 1$ dominating set for K_0. Vice versa, the only way to dominate the $k + 2$ leaves of $K_{1,k+2}$ with $k + 1$ vertices is by picking the center c. Hence, if there exists a solution S for $(K_0, k + 1)$, it must contain c. Note that all the red vertices are dominated by c and every blue vertex has no blue neighbor. Therefore, we can replace every blue vertex in S with any of its red neighbors. This yields a set S' containing c and at most k red vertices dominating all blue vertices. These red vertices witness that (K, k) is a yes-instance as well. ◀

Having proved Theorem 6.2 and Theorem 6.3, this concludes the proof of Theorem 6.1.

References

1 Hans Adler and Isolde Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics*, 36:322–330, 2014.

2 Miklós Ajtai, János Komlós, and Endre Szemerédi. A note on ramsey numbers. *Journal of Combinatorial Theory, Series A*, 29(3):354–360, 1980.

3 Noga Alon, Richard A Duke, Hanno Lefmann, Vojtěch Rödl, and Raphael Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16(1):80–109, 1994.

4 John T Baldwin and Saharon Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.

5 Achim Blumensath. Simple monadic theories and indiscernibles. *Mathematical Logic Quarterly*, 57(1):65–86, 2011. doi:10.1002/malq.200910121.

6 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *FSTTCS*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness ii: On completeness for w [1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

9 Jan Dreier, Jakub Gajarskỳ, Sandra Kiefer, Michał Pilipczuk, and Szymon Toruńczyk. Treelike decompositions for transductions of sparse graphs. *arXiv preprint*, 2022. `arXiv:2201.11082`.

10 Jan Dreier, Nikolas Mählmann, Sebastian Siebertz, and Szymon Toruńczyk. Indiscernibles and wideness in monadically stable and monadically NIP classes. *arXiv preprint*, 2022. `arXiv:2206.13765`.

11 Zdeněk Dvořák. Induced subdivisions and bounded expansion. *European Journal of Combinatorics*, 69:143–148, 2018.

12 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In *ICALP*, volume 80 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

13 Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. Progressive Algorithms for Domination and Independence. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.STACS.2019.27`.

14 Jakub Gajarskỳ, Petr Hliněnỳ, Jan Obdržálek, Daniel Lokshtanov, and MS Ramanujan. A new perspective on fo model checking of dense graph classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–23, 2020.

15 Jakub Gajarskỳ, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona De Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–41, 2020.

16 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM (JACM)*, 64(3):1–32, 2017.

17 Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

18 Yiting Jiang, Jaroslav Nešetřil, Patrice Ossona de Mendez, and Sebastian Siebertz. Regular partitions of gentle graphs. *Acta Mathematica Hungarica*, 161(2):719–755, 2020.

19 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Transactions on Algorithms (TALG)*, 15(2):1–19, 2018.

20 M. Malliaris and S. Shelah. Regularity lemmas for stable graphs. *Transactions of the American Mathematical Society*, 366(3):1551–1585, 2014. URL: `http://www.jstor.org/stable/23813167`.

21 Jaroslav Nešetřil and P Ossona de Mendez. Structural sparsity. *Russian Mathematical Surveys*, 71(1):79, 2016.

22 Jaroslav Nešetřil and Patrice Ossona De Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

23 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

24 Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Rankwidth meets stability. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2014–2033. SIAM, 2021.

25 Jaroslav Nešetřil, Patrice Ossona de Mendez, and Sebastian Siebertz. Structural properties of the first-order transduction quasiorder. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

26 Jaroslav Nešetřil, Roman Rabinovich, Patrice Ossona de Mendez, and Sebastian Siebertz. Linear rankwidth meets stability. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1180–1199. SIAM, 2020.

27 Bejamin Oye. Stable graphs.

28 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

29 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

30 Endre Szemerédi. Regular partitions of graphs. *Colloq. Int. CNRS*, 260:399–401, 1978.

31 Vladimir Naumovitch Vapnik and Alexeï Iakovlevitch Červonenkis. On the uniform convergence of relative sequences of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

Complexity and Algorithms for Isometric Path Cover on Chordal Graphs and Beyond

Dibyayan Chakraborty
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Antoine Dailly
G-SCOP, Univ. Grenoble-Alpes, Grenoble, France

Sandip Das
Indian Statistical Institute, Kolkata, India

Florent Foucaud
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans Cedex 2, France

Harmender Gahlawat
Ben-Gurion University of the Negev, Beer-Sheva, Israel
Indian Statistical Institute, Kolkata, India

Subir Kumar Ghosh
Ramakrishna Mission Vivekananda Educational and Research Institute, Kolkata, India

Abstract

A path is *isometric* if it is a shortest path between its endpoints. In this article, we consider the graph covering problem Isometric Path Cover, where we want to cover all the vertices of the graph using a minimum-size set of isometric paths. Although this problem has been considered from a structural point of view (in particular, regarding applications to pursuit-evasion games), it is little studied from the algorithmic perspective. We consider Isometric Path Cover on chordal graphs, and show that the problem is NP-hard for this class. On the positive side, for chordal graphs, we design a 4-approximation algorithm and an FPT algorithm for the parameter solution size. The approximation algorithm is based on a reduction to the classic path covering problem on a suitable *directed acyclic graph* obtained from a *breadth first search traversal* of the graph. The approximation ratio of our algorithm is 3 for interval graphs and 2 for proper interval graphs. Moreover, we extend the analysis of our approximation algorithm to k-chordal graphs (graphs whose induced cycles have length at most k) by showing that it has an approximation ratio of $k + 7$ for such graphs, and to graphs of treelength at most ℓ, where the approximation ratio is at most $6\ell + 2$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Shortest paths, Isometric path cover, Chordal graph, Interval graph, AT-free graph, Approximation algorithm, FPT algorithm, Treewidth, Chordality, Treelength

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.12

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03710812`

Funding This research was financed by the IFCAM project "Applications of graph homomorphisms" (MA/IFCAM/18/39).
Florent Foucaud: This author was financed by the ANR project GRALMECO (ANR-21-CE48-0004-01) and the French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).
Harmender Gahlawat: This author was financed by the ISF grant no. 1176/18.

Acknowledgements We thank Vincent Limouzy, Joydeep Mukherjee, Lucas Pastor and Jean-Florent Raymond for initial discussions on the topic of this paper.

1 Introduction

Problems involving paths in graphs are fundamental in theoretical computer science. A prominent example is PATH COVER, which asks whether the vertex set of an input graph can be covered by at most k paths. This problem is NP-hard even for $k = 1$ as, in this case, it is HAMILTON PATH. PATH COVER is extensively studied and has many applcations, see [3, 5, 26]. A packing counterpart of PATH COVER is the well-known problem DISJOINT PATHS which asks, given pairs of terminal vertices of a graph G, for disjoint paths joining the terminal pairs. DISJOINT PATHS has found many applications, due to its connections to the Graph Minor project [29].

Certain types of paths are of special interest, in particular, shortest paths between vertex pairs are important in many applications. A path is called *isometric* if it is a shortest path between two vertices. The corresponding variant of DISJOINT PATHS, called DISJOINT SHORTEST PATHS, has recently gained some attention [22]. The goal of this paper is to study the "shortest path" variant of PATH COVER, which was introduced in [13] with inspiration from earlier work on pursuit-evasion games [2].

An *isometric path cover* of a graph G is a set of isometric paths such that each vertex of G belongs to at least one of the paths. The *isometric path number* of G is the smallest size of an isometric path cover of G. The algorithmic problem studied in this paper is as follows.

ISOMETRIC PATH COVER
Input: A graph G, and a positive integer k.
Question: Does G have an isometric path cover of size at most k?

ISOMETRIC PATH COVER was introduced in the context of the well-known *Cops and Robber* game (where multiple cops try to catch a robber, each protagonist being able to move to an adjacent vertex in each round of the game). Indeed, given an isometric path cover, one can assign a cop to "guard" each isometric path: each cop patrols along its path, always staying as close as possible to the robber. This strategy shows that the isometric path number of the graph is an upper bound to the cop number of the graph (the smallest number of cops needed to catch one robber) [2, 13]. More sophisticated techniques, still based on isometric paths, have been developed in this context, for example *cop-decompositions*, which are tree decomposition where the subgraph induced by the vertices of each bag that are not present in the parent's bag has a small isometric path cover [1]. ISOMETRIC PATH COVER plays a crucial role in the proof of the *Product Structure Theorem* [11] of planar graphs. ISOMETRIC PATH COVER is also studied in the context of machine learning [30].

Surprisingly, the algorithmic complexity of ISOMETRIC PATH COVER has not garnered much attention. Its NP-hardness was recently posed as an open problem in both [23, 24]. The problem is easy to solve on trees [3]. More generally, ISOMETRIC PATH COVER is known to be polynomial-time solvable on block graphs [27]. It can be approximated in polynomial time within a factor of $\log(d)$ for graphs of diameter d by a greedy algorithm [30] and solved in polynomial time for every fixed value of k by an XP algorithm [12]. ISOMETRIC PATH COVER has also been studied from a structural point of view: the optimal solution sizes have been determined for square grids [13], hypercubes [14], complete r-partite graphs [28] and Cartesian products of complete graphs [28], and it was recently proved that the pathwidth of a graph is always upper-bounded by the size of its smallest isometric path cover [12]. The version where the cover is actually a *partition* was also studied [23]. The variants where the set of endpoints of the paths is prescribed in the input is studied in [8, 12, 21], and when the set of allowed paths is prescribed it is studied on trees in [18].

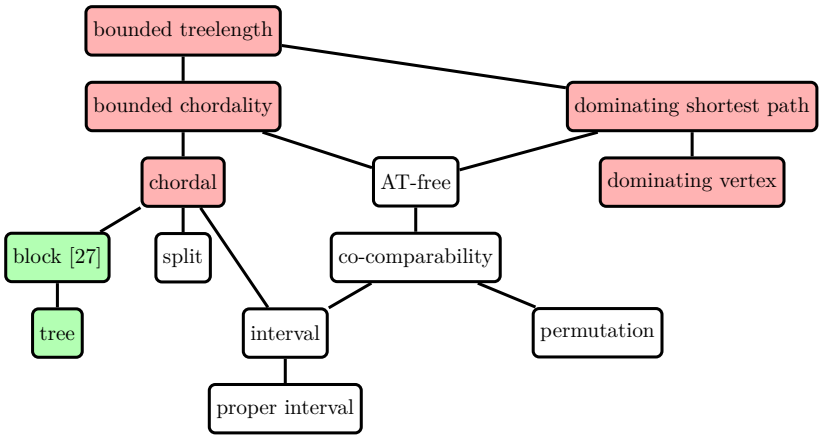


Figure 1 Inclusion diagram for graph classes discussed here (and related ones). If a class A has an upward path to class B, then A is included in B. For graphs in the green classes, Isometric Path Cover is polynomial-time solvable; for graphs in the red classes, it is NP-complete. For all shown graph classes, Isometric Path Cover is constant-factor approximable in polynomial time.

Graph classes studied in this paper. Let us introduce the various graph classes studied in this paper. See Figure 1 for a visualization of the inclusion diagram of the graph classes discussed in this paper (and related ones). A *chordal graph* is a graph without any induced cycle of order at least 4. An *interval representation* of a graph G is a set $\mathcal{I} = \{[x_u^-, x_u^+] : u \in V(G)\}$ of intervals where each interval in $\mathcal{I}$ corresponds to a vertex, and two intervals intersect if and only if the corresponding vertices share an edge. A graph is an *interval graph* if it has an interval representation. A graph is a *proper interval* graph if it has an interval representation where no interval contains another interval as a subset. Interval graphs are also chordal graphs. In fact, Fulkerson & Gross [15] proved that interval graphs are exactly the chordal graphs without an *asteroidal triple* (three vertices a, b, c of a graph G form an asteroidal triple if for any $\{w_1, w_2, w_3\} = \{a, b, c\}$, there is a path between w_1, w_2 that does not contain any vertex from the neighbourhood of w_3.)

Even though the classes of AT-free graphs (*i.e.*, graphs without an asteroidal triple) and chordal graphs are incomparable, both of them have bounded *chordality*. A graph is *k-chordal* if it does not contain an induced cycle of order greater than k. Chordal graphs are exactly the class of 3-chordal graphs, and all AT-free graphs are 5-chordal. A superclass of AT-graphs are graphs that contain a *dominating shortest path* [7]. A graph G has a *dominating shortest path* if there exists a shortest path P in G such that the closed neighbourhood of any vertex of the graph intersects with the vertex set of P. Even though graph classes with bounded chordality are incomparable with the class of graphs having a dominating shortest path, both of these classes have bounded *treelength*, a parameter introduced by Dourisboure & Gavoille [10]. A *tree-decomposition* of a graph G is a tree T where each vertex v of T is associated to a subset X_v of $V(G)$ called *bag*, such that: (i) $\bigcup_{v \in V(T)} X_v = V(G)$, (ii) for each edge $xy \in E(G)$, there exists a bag X_v such that $\{x, y\} \subseteq X_v$, and (iii) for every vertex $x \in V(G)$, the vertices of T associated to the bags containing x induce a connected subtree of T. Define $length(T) = \max_{\substack{v \in V(T) \\ u,v \in X_v}} \mathsf{d}(u, v)$, where the distance $\mathsf{d}(u, v)$ denotes the number of edges in a shortest path between u and v in G. The *treelength* of G, denoted as $tl(G)$, is defined as $\min_T length(T)$, where the minimum is taken over all tree-decompositions of G.

Our results. We first settle the question of the complexity of ISOMETRIC PATH COVER, showing it is NP-hard even for chordal graphs.

▶ **Theorem 1.** *ISOMETRIC PATH COVER is NP-hard, even for chordal graphs with a dominating vertex.*

To complement the above result, we design a constant-factor approximation algorithm for ISOMETRIC PATH COVER for graph classes that strictly contain chordal graphs, and other related ones. We summarize these results in Theorem 2.

▶ **Theorem 2.** *There is a polynomial-time approximation algorithm that computes a valid solution for ISOMETRIC PATH COVER for every input graph, and has performance ratio of:*
(a) 2 *on proper interval graphs,*
(b) 3 *on interval graphs,*
(c) 4 *on chordal graphs,*
(d) 5 *on graphs with a dominating shortest path,*
(e) $(k+7)$ *on k-chordal graphs, for $k \geq 4$, and*
(f) $(6\ell+2)$ *on graphs with treelength at most ℓ.*

Theorem 2 is proved by analyzing one algorithm, which is based on constructing a suitable directed acyclic graph by breadth-first-search, and a reduction to the directed path covering problem for this digraph. We also prove that our analysis is tight for items (a), (b) and (c).

We then show that, on chordal graphs, one can solve ISOMETRIC PATH COVER in linear time when the treewidth (*i.e.*, the clique number) is bounded, which implies that ISOMETRIC PATH COVER is fixed-parameter-tractable (FPT) on this class for parameter solution size.

▶ **Theorem 3.** *On chordal graphs of order n and treewidth w, ISOMETRIC PATH COVER can be solved in time $2^{k2^{\mathcal{O}(w)}}n$ and in time $2^{2^{\mathcal{O}(k)}}n$, where k is the solution size.*

Organisation of the paper. We first prove our hardness result in Section 2. We then describe and analyze our approximation algorithm for ISOMETRIC PATH COVER in Section 3. The FPT algorithm for ISOMETRIC PATH COVER on chordal graphs is described in Section 4. We conclude in Section 5.

General notations. A sequence of vertices forms a *path* P if any two consecutive vertices are adjacent. Whenever we fix a path P of G, we shall refer to the subgraph formed by the edges between the consecutive vertices of P. For a path P of a graph G between two vertices u and v, the vertices $V(P) \setminus \{u, v\}$ are *internal vertices* of P. A path between two vertices u and v is called a (u, v)-path. Similarly, we have the notions of *isometric* (u, v)*-path* and *induced* (u, v)*-path*.

2 NP-hardness of Isometric Path Cover on chordal graphs

In this section, we prove that ISOMETRIC PATH COVER is NP-hard, answering a question raised in both [24, 23]. In fact, we prove that ISOMETRIC PATH COVER is NP-hard for chordal graphs. To prove this, we reduce INDUCED P_3-PARTITION on chordal graphs to ISOMETRIC PATH COVER on chordal graphs (in fact the reduction is the same as the one in [23]). Given a graph G, the objective of INDUCED P_3-PARTITION is to decide if there exists a partition $\mathcal{P}$ of $V(G)$ such that each set in $\mathcal{P}$ induces a path on three vertices in G. We use the following result, which is implied from a result of van Bevern et al. [31, Theorem 9] (their result does not concern *induced* paths, but one can easily check that their reduction holds with this restriction too).

▶ **Proposition 4** ([31]). *Induced P_3-Partition is NP-hard even if the input is a chordal graph with $3k$ vertices for some integer k.*

Proof of Theorem 1. To prove this, we give a reduction from Induced P_3-Partition on chordal graphs to Isometric Path Cover on chordal graphs. Let G be a chordal graph such that $|V(G)| = 3k$ for some integer $k \geq 1$. Let G' be the graph whose vertex set is $V(G') = V(G) \cup \{u, v, w\}$, where u, v, w are three new vertices. The edge set of G' is $E(G') = E(G) \cup \{(u, v), (v, w)\} \cup \{(v, x) \mid x \in V(G)\}$. It is easy to see that G' is a chordal graph, and that v is a dominating vertex.

We shall show that G is a yes-instance of Induced P_3-Partition if and only if G' has an isometric path cover of cardinality $k + 1$. We have the following observation, due to the fact that G' has diameter 2.

▶ **Observation 5.** *Any isometric path of G' contains at most three vertices.*

First, let $\mathcal{P}$ be a partition of $V(G)$ such that each set $P \in \mathcal{P}$ induces a path on three vertices. Observe that for any two vertices $u, v \in V(G')$, the isometric (u, v)-path in G' contains at most three vertices. Therefore, each path $P \in \mathcal{P}$ is in fact an isometric path in G'. Hence, $\mathcal{P} \cup \{(u, v, w)\}$ is a set of isometric paths with cardinality $k + 1$ that covers all vertices of G'.

To prove the reverse direction, assume that G' has an isometric path cover $\mathcal{C}$ of size at most $k + 1$. We now have the following observation.

▶ **Observation 6.** *There is an isometric path in $\mathcal{C}$ that covers both u and w in G'.*

Proof. Otherwise, let $P, Q \in \mathcal{C}$ be two distinct isometric paths that cover u and v, respectively and $S = V(G') \setminus (P \cup Q)$. Observe that $|S| \geq 3k-2$ and $\mathcal{C}$ contains a subset with $\mathcal{C}'$ containing $k - 1$ isometric paths and covering all vertices of S. Therefore, $\mathcal{C}'$ contains an isometric path that covers at least four vertices of S. But this contradicts Observation 5. ◀

Now consider the set $\mathcal{C}' = \mathcal{C} \setminus \{P\}$ where $P \in \mathcal{C}$ that covers both u and w. Observe that $\mathcal{C}'$ contains k paths that must cover all vertices of $V(G') \setminus \{u, v, w\}$, *i.e.*, of $V(G)$. Due to the facts that $|V(G)| = 3k$ and any isometric path in G' contains at most three vertices of G' (Observation 5), we have that $\mathcal{C}'$ is a partition of $V(G)$, and therefore G is a yes-instance of Induced P_3-Partition. ◀

3 An approximation algorithm for Isometric Path Cover

In this section, we will describe our approximation algorithm and prove Theorem 2. We will need the following definitions. For a vertex r of G and a set S of vertices of G, the *distance of S from r*, denoted as $\mathsf{d}(r, S)$, is the minimum of the distance between any vertex of S and r. For a subgraph H of G, the *distance of H w.r.t. r* is $\mathsf{d}(r, V(H))$. Formally, we have $\mathsf{d}(r, S) = \min\{\mathsf{d}(r, v) : v \in S\}$ and $\mathsf{d}(r, H) = \mathsf{d}(r, V(H))$.

For a graph G and a vertex $r \in V(G)$, consider the following operations on G. First, remove all edges xy from G such that $\mathsf{d}(r, x) = \mathsf{d}(r, y)$. Let G'_r be the resulting graph. Then, for each edge $e = xy \in E(G'_r)$ with $\mathsf{d}(r, x) = \mathsf{d}(r, y) - 1$, orient e from y to x. Let $\overrightarrow{G_r}$ be the directed acyclic graph formed after applying the above operation on G'. Note that this digraph can easily be computed in linear time using a Breadth-First Search (BFS) traversal with starting vertex r.

In a digraph, a *directed path* is a path in the underlying undirected graph, such that all arcs are oriented in the same direction. A *directed path cover* of $\overrightarrow{G_r}$ is a set of directed paths such that each vertex of $\overrightarrow{G_r}$ belongs to at least one of the paths. We have the following observation, which holds because any directed path of $\overrightarrow{G_r}$ is an isometric path in G.

Algorithm 1 An algorithm for ISOMETRIC PATH COVER.

Input : A graph G and a vertex $v \in V(G)$.
Output : An isometric path cover of G.

1 Construct the graph $\overrightarrow{G_v}$ using a BFS starting at v;
2 $\mathcal{P}_v \leftarrow$ directed path cover of $\overrightarrow{G_v}$ with minimum cardinality, computed using the reduction to bipartite matching;
3 **return** the set of paths obtained from $\mathcal{P}_v$ by removing all orientations.

▶ **Observation 7.** *For any vertex r of a graph G, a directed path cover of $\overrightarrow{G_r}$ is an isometric path cover of G.*

The directed path cover problem in directed acyclic digraphs is the subject of Dilworth's theorem [9] (phrased in the equivalent language of partially ordered sets), which states that the size of an optimal solution is equal to the maximum size of an *antichain*, that is, a set of vertices in which no two vertices have a directed path connecting them. In a constructive proof of this theorem, Fulkerson [16] showed that the problem can be reduced to the maximum matching problem in a suitable bipartite graph, and thus, can be solved optimally in polynomial time.

The pseudocode of our algorithm for ISOMETRIC PATH COVER is given in Algorithm 1. Even though our algorithm will remain the same for all the considered graph classes, the analysis will differ. We will show that, depending on the graph class of the input graph G, there exists a "favourable choice" of a vertex v such that a directed path cover of $\overrightarrow{G_v}$ is an isometric path cover of G, whose cardinality is not too far away from the isometric path number of G. To analyse the performance of our algorithm, we need the following definitions.

▶ **Definition 8.** *For a graph G and a vertex $r \in V(G)$, two vertices $x, y \in V(G)$ are* antichain vertices *if there are no directed paths from x to y or from y to x in $\overrightarrow{G_r}$. A set X of vertices of G is an* antichain set *if any two vertices in X are antichain vertices. The cardinality of the largest antichain set in $\overrightarrow{G_r}$ will be denoted by $\beta\left(\overrightarrow{G_r}\right)$. The cardinality of the largest antichain set of G, is defined as*

$$\beta(G) = \min\left\{\beta\left(\overrightarrow{G_r}\right) : r \in V(G)\right\}$$

▶ **Definition 9.** *Let r be a vertex of a graph G. For a path P, $A_r(P)$ shall denote the maximum antichain set of P in $\overrightarrow{G_r}$. The* isometric path antichain cover number *of $\overrightarrow{G_r}$, denoted by $ipacc\left(\overrightarrow{G_r}\right)$, is defined as follows:*

$$ipacc\left(\overrightarrow{G_r}\right) = \max\{|A_r(P)| : \ P \text{ is an isometric path}\}$$

The isometric path antichain cover number *of graph G, denoted as $ipacc(G)$, is defined as the minimum over all possible antichain covers of its associated directed acyclic graphs:*

$$ipacc(G) = \min\left\{ipacc\left(\overrightarrow{G_r}\right) : r \in V(G)\right\}$$

We will use the next lemma that follows directly from Dilworth's Theorem [9], Observation 7, and Definitions 8 and 9.

▶ **Lemma 10.** *Let G be a graph and $\mathcal{P}$ be any isometric path cover of G with minimum cardinality. We have:*

$$\frac{\beta(G)}{ipacc(G)} \leq |\mathcal{P}| \leq \beta(G).$$

Proof. Let r be a vertex of G such that $\beta(G) = \beta\left(\overrightarrow{G_r}\right)$. Then, by Observation 7, we have that $|\mathcal{P}| \leq \beta\left(\overrightarrow{G_r}\right) = \beta(G)$. Now, let r' be a vertex of G such that $ipacc(G) = ipacc\left(\overrightarrow{G_{r'}}\right)$. Since any isometric path in $\mathcal{P}$ contains at most $ipacc\left(\overrightarrow{G_{r'}}\right)$ many elements of $\beta\left(\overrightarrow{G_{r'}}\right)$, we have $\frac{\beta\left(\overrightarrow{G_{r'}}\right)}{ipacc\left(\overrightarrow{G_{r'}}\right)} \leq |\mathcal{P}|$. Finally, since $\beta(G) \leq \beta\left(\overrightarrow{G_{r'}}\right)$, we have $\frac{\beta(G)}{ipacc(G)} \leq |\mathcal{P}|$. ◀

In the next section, we will prove upper bounds on the isometric path antichain cover number of various graph classes, implying the approximation ratios fulfilled by Algorithm 1.

3.1 Lemmas on the isometric path antichain cover number

We now prove some lemmas relating the isometric path antichain cover number with other parameters. We begin by establishing a relationship between the length of an isometric path P and the size of $A_r(P)$, which will be crucial for our analysis of Algorithm 1.

▶ **Lemma 11.** *Let G be a graph and r, an arbitrary vertex of G. Consider the directed acyclic graph $\overrightarrow{G_r}$, and let P be an isometric path between two vertices x and y in G with $\mathsf{d}(r,x) \leq \mathsf{d}(r,y)$. Then $|P| \geq \mathsf{d}(r,y) - \mathsf{d}(r,x) + |A_r(P)| - 1$.*

Proof. Orient the edges of P from y to x in G. First, observe that P must contain a set E_1 of oriented edges such that $|E_1| = \mathsf{d}(r,y) - \mathsf{d}(r,x)$ and for any $\overrightarrow{ab} \in E_1$, $\mathsf{d}(r,a) = \mathsf{d}(r,b) + 1$. Let the vertices of the largest antichain set of P in $\overrightarrow{G_r}$, *i.e.*, $A_r(P)$, be ordered as $a_1, a_2, \ldots, a_t$ according to their occurrence while traversing P from y to x. For $i \in [2,t]$, let P_i be the subpath of P between a_{i-1} and a_i. Observe that for any $i \in [2,t]$, since a_i and a_{i-1} are antichain vertices, there must exist an oriented edge $\overrightarrow{b_i c_i} \in E(P_i)$ such that either $\mathsf{d}(r,b_i) = \mathsf{d}(r,c_i)$ or $\mathsf{d}(r,b_i) = \mathsf{d}(r,c_i) - 1$. Let $E_2 = \{b_i c_i\}_{i \in [2,t]}$. Observe that $E_1 \cap E_2 = \emptyset$ and therefore $|P| \geq |E_1| + |E_2| = \mathsf{d}(r,y) - \mathsf{d}(r,x) + |A_r(P)| - 1$. ◀

Next, we shall relate isometric path antichain cover number with a parameter called *cluster diameter*, introduced in [10]. Let G be a graph and r be an arbitrary vertex of G. For a non-negative integer i, let $G^i(r)$ denote the graph induced by the vertices whose distance from r is at least i. Formally, $G^i(r) = G[\{u : \mathsf{d}(r,u) \geq i\}]$. A *cluster* is a set S of vertices such that all vertices of S are at the same distance from r and any two vertices of S lie in the same connected component of $G^i(r)$, where $i = \mathsf{d}(r,S)$. The *cluster diameter of G with respect to r*, denoted as $\Delta_r(G)$, was defined in [10] as follows:

$$\Delta_r(G) = \max\{\mathsf{d}(u,v) : u, v \text{ lie in the same cluster with respect to } r\}$$

We shall use the following technical lemma to prove bounds on the isometric path antichain cover number of graphs with bounded treelength and on graphs with bounded chordality in Lemma 16 and Lemma 18, respectively.

▶ **Lemma 12.** *Let G be a graph, r be an arbitrary vertex of G, and let P be an isometric path such that $|A_r(P)| \geq \alpha$ in $\overrightarrow{G_r}$. Then $\Delta_r(G) \geq \left\lceil \frac{\alpha}{2} \right\rceil - 1$.*

Proof. Let the two endpoints of P be u and v and, without loss of generality, assume $\mathsf{d}(r,u) \leq \mathsf{d}(r,v)$. Let $A = A_r(P)$, and let a be a vertex of P such that $\mathsf{d}(r,a) = \mathsf{d}(r,P)$. Let P_u (resp. P_v) denote the subpath of P between u and a (resp. between v and a). Observe that there exists a path $Q \in \{P_u, P_v\}$ such that Q contains an antichain set of cardinality $\lceil \frac{\alpha}{2} \rceil$. Notice that a is one of the endpoints of Q and $\mathsf{d}(r,Q) = \mathsf{d}(r,a)$. Let c be the other endpoint of Q. Let a_1 be the first vertex of A which is encountered while traversing Q starting from c and ending at a. If $\mathsf{d}(r,a_1) = \mathsf{d}(r,a)$, then let $b = a_1$. Otherwise, consider a vertex b such that $\mathsf{d}(r,b) = \mathsf{d}(r,a)$ and there is an oriented path from a_1 to b in $\overrightarrow{G_r}$. Clearly, a and b lie in the same cluster of G with respect to r. If $\mathsf{d}(a,b) \leq \left(\lceil \frac{\alpha}{2} \rceil - 2\right)$, then $\mathsf{d}(a,a_1) \leq \mathsf{d}(a,b) + \mathsf{d}(b,a_1) \leq \left(\lceil \frac{\alpha}{2} \rceil - 2\right) + \mathsf{d}(r,a_1) - \mathsf{d}(r,b) < |A_r(Q)| - 1 + \mathsf{d}(r,a_1) - \mathsf{d}(r,a)$. But this contradicts Lemma 11. Hence, $\mathsf{d}(a,b) \geq \lceil \frac{\alpha}{2} \rceil - 1$. ◀

Next, we state the following definition.

▶ **Definition 13.** *For an integer $t \geq 1$, a graph G is* t-slender *if there exists a vertex $r \in V(G)$ such that, for all vertices $u, v \in V(G)$ with $\mathsf{d}(r,u) = \mathsf{d}(r,v)$, we have $\mathsf{d}(u,v) \leq t$.*

Observe that if a graph G is t-slender, then, there exists a vertex $r \in V(G)$ such that $\Delta_r(G) \leq t$. Lemma 12 then implies $ipacc(G) \leq 2t + 2$. However, the following lemma will help us prove better upper bounds for graphs that are t-slender.

▶ **Lemma 14.** *Let G be a t-slender graph for some integer $t \geq 1$. Then, $ipacc(G) \leq t + 1$.*

Proof. By definition, there exists a vertex $r \in V(G)$ such that for any $u, v \in V(G)$ with $\mathsf{d}(r,u) = \mathsf{d}(r,v)$, we have $\mathsf{d}(u,v) \leq t$. Let P be an isometric path between two vertices x and y with, without loss of generality, $\mathsf{d}(r,x) \leq \mathsf{d}(r,y)$. If $\mathsf{d}(r,x) < \mathsf{d}(r,y)$, then let y' be a vertex such that $\mathsf{d}(r,y') = \mathsf{d}(r,x)$ and there is a path between y to y' in $\overrightarrow{G_r}$. Otherwise, let $y' = y$. Since $\mathsf{d}(x,y') \leq t$, $\mathsf{d}(x,y) \leq t + \mathsf{d}(r,y) - \mathsf{d}(r,x)$. On the other hand, due to Lemma 11, we have $|P| = \mathsf{d}(x,y) \geq \mathsf{d}(r,y) - \mathsf{d}(r,x) + |A_r(P)| - 1$. Hence, $|A_r(P)| \leq t + 1$. ◀

In particular, we shall use the above lemma to prove better upper bounds for the isometric anti chain path cover number of graphs containing a dominating shortest path, interval graphs and proper interval graphs in Lemma 20, 21, and 22, respectively.

3.2 Upper bounds on the isometric path antichain cover number

In this section, we will first show that $ipacc(G)$ can be bounded by a linear function of $tl(G)$. We will use the following result of Dourisboure & Gavoille [10], which was restated in the following form by Abdulhakeem & Dragan [25].

▶ **Proposition 15** ([10, 25])**.** *Let r be an arbitrary vertex of a graph G with treelength at most ℓ. Then $\Delta_r(G) \leq 3\ell$.*

▶ **Lemma 16.** *If G is a graph with treelength at most ℓ, then $ipacc(G) \leq 6\ell + 2$.*

Proof. Assume that there exists a vertex r of G and an isometric path P of G such that $|A_r(P)| \geq 6\ell + 3$ in $\overrightarrow{G_r}$. Then, by Lemma 12, there are two vertices a and b such that $\mathsf{d}(r,a) = \mathsf{d}(r,b)$, $\mathsf{d}(a,b) \geq \lceil \frac{6\ell+3}{2} \rceil - 1 \geq 3\ell + 1$, and a, b lie in the same cluster with respect to r. Hence, $\Delta_r(G) \geq 3\ell + 1$. This contradicts Proposition 15. ◀

Now, we will prove an upper bound for the isometric path antichain cover number of k-chordal graphs. Note that the treelength of k-chordal graphs is at most $\frac{k}{2}$ [17]. Therefore, Lemma 16 implies that the isometric path antichain cover number of a k-chordal graph is at most $3k + 1$. To prove a better upper bound, we will also use the following result of Dourisboure & Gavoille [10].

▶ **Proposition 17** ([10]). *Let r be any vertex of a k-chordal graph G. Then, $\Delta_r(G) \leq \frac{k}{2} + 2$.*

▶ **Lemma 18.** *If G is a k-chordal graph with $k \geq 4$, then $ipacc(G) \leq k + 7$.*

Proof. Assume that there exists a vertex r of G and an isometric path P of G such that $|A_r(P)| \geq k + 8$ in $\overrightarrow{G_r}$. Then, by Lemma 12, there are two vertices a and b such that $\mathsf{d}(r,a) = \mathsf{d}(r,b)$, $\mathsf{d}(a,b) \geq \lceil \frac{k+8}{2} \rceil - 1 \geq \frac{k}{2} + 3$, and a, b lie in the same cluster with respect to r. This contradicts Proposition 17. ◀

Now, we will prove upper bounds on the isometric path antichain cover number of graphs with a dominating shortest path. Recall that a shortest path P of a graph G is *dominating* if any vertex of the graph is either in P or adjacent to at least one of the vertices of P. Note that the class of graphs with a dominating shortest path is incomparable with the class of k-chordal graphs for any fixed integer k. We can now prove the following lemma.

▶ **Lemma 19.** *If a graph G has a dominating shortest path, then G is 4-slender.*

Proof. Let r, s be the endpoints of a dominating shortest path P. Let $x_0 = r, x_1, x_2, \ldots, x_i = s$ be the vertices of P ordered as they are encountered while traversing P from r to s. Let a, b be two vertices of G such that $\mathsf{d}(r,a) = \mathsf{d}(r,b)$. If $\mathsf{d}(r,a) = i + 1$, then $\mathsf{d}(a,b)$ is at most 2 as both a, b will be adjacent to x_i. If $\mathsf{d}(r,a) = i$, then $\mathsf{d}(a,b)$ is at most 3 since $\{a, b\} \subset N[x_i] \cup N[x_{i-1}]$. Otherwise, $0 < \mathsf{d}(r,a) \leq i - 1$. In this case, $\mathsf{d}(a,b)$ is at most 4 since $\{a, b\} \subset N[x_{i-1}] \cup N[x_i] \cup N[x_{i+1}]$. ◀

Taken together, Lemmas 14 and 19 imply the following lemma which we will use again in Section 3.3.

▶ **Lemma 20.** *If a graph G has a dominating shortest path, then $ipacc(G) \leq 5$.*

We will prove an improved version of Lemma 20 for interval graphs.

▶ **Lemma 21.** *For any interval graph G, we have $ipacc(G) \leq 3$.*

Proof. Due to Lemma 14, we will be done by showing that if G is an interval graph, then, G is 2-slender. Let $\mathcal{I} = \{[x_u^-, x_u^+] : u \in V(G)\}$ be an interval representation of G. Let v be the vertex such that $x_v^+ = \min\{x_a^+ : a \in V(G)\}$. In other words, v corresponds to the interval with the leftmost right endpoint. For a vertex w, define $r_w = z$ such that $x_z^+ = \max\{x_{z'}^+ : z' \in N[z]\}$. In other words, z is the neighbour of w that has the rightmost right endpoint. Observe that G has a dominating shortest path $x_0 = v, x_1, x_2, \ldots, x_i$ such that for each $1 \leq j \leq i$, $x_i = r_{x_{i-1}}$. Now, consider two vertices a, b with $\mathsf{d}(v,a) = \mathsf{d}(v,b) = j$. Observe that $\{a, b\} \subset N[x_{j-1}]$. Hence, $\mathsf{d}(a,b) \leq 2$ and therefore, G is 2-slender. ◀

Any proper interval graph G has a vertex v such that the spanning tree $\mathcal{T}_v$ obtained from a BFS starting at v is 1-slender [19], and thus, G is 1-slender as well. An immediate consequence of this result, due to Lemma 14, is the following.

▶ **Lemma 22.** *If G is a proper interval graph, then $ipacc(G) \leq 2$.*

Interval graphs are a subclass of chordal graphs (*i.e.*, graphs with chordality 3). Since chordal graphs are exactly the graphs with treelength 1, Lemma 16 implies that the isometric path antichain cover number of chordal graphs is at most 7. Below, we prove a better upper bound using the two following properties of chordal graphs.

▶ **Observation 23.** *Let r be an arbitrary vertex of a chordal graph G. Let $u, v \in V(G)$ be two vertices such that $\mathsf{d}(r,u) = \mathsf{d}(r,v) = i$ and there exists an (u,v)-path P such that $V(P - \{u,v\}) \subseteq V(G_r^{i+1})$. Then $uv \in E(G)$.*

Proof. If P has length one, the result is obvious, thus, we can assume that P has length at least 2. Assume for contradiction that $uv \notin E(G)$. Let P_1 be an isometric (u,v)-path in the graph induced by $V(G - G^i(r)) \cup \{u,v\}$. Consider another isometric (u,v)-path Q such that $\mathsf{d}(r, Q - \{u,v\}) = \mathsf{d}(r,u) + 1$. Note that the existence of P guarantees that there is at least one such path. For any two nonconsecutive vertices $u_i, u_j \in Q$, $u_iu_j \notin E(G)$. Since $uv \notin E(G)$, note that P_1 and Q are induced paths. Moreover, since, for any vertex $p \in V(P_1 - \{u,v\})$ and $q \in V(Q - \{u,v\})$, $\mathsf{d}(r,q) - \mathsf{d}(r,p) \geq 2$, the paths P_1 and Q are internally disjoint. Therefore, $P_1 \cup Q$ induces a cycle of length at least 4, which contradicts the fact that G is a chordal graph. ◀

▶ **Observation 24.** *Let r be an arbitrary vertex of a chordal graph G, and let P be an isometric path of G. Let $u, v \in V(P)$ be two distinct vertices of P such that $\mathsf{d}(r,u) = \mathsf{d}(r,v)$. Then, there cannot be any vertex w in the (u,v)-subpath of P such that $\mathsf{d}(r,w) > \mathsf{d}(r,v)$.*

Proof. Assume for contradiction that such a path P and vertices u, v, and w exist, and let $\mathsf{d}(r,u) = \mathsf{d}(r,v) = i$. Consider the (u,v)-subpath Q of P, and let $u = v_1, \ldots, v_\ell = v$ be the ordering of vertices of Q, along the path Q. Moreover, let the alias of the vertex w in this ordering be v_b. Then, observe that there exist two vertices $v_a, v_c \in V(Q)$, where $1 \leq a < b < c \leq \ell$, such that $\mathsf{d}(r,v_a) = \mathsf{d}(r,v_c) = i$, $v_av_c \notin E(G)$, and there is a (v_a,v_c)-subpath Q' of Q such that $V(Q' - \{v_a,v_c\}) \subseteq V(G_r^{i+1})$. However, this contradicts Observation 23. ◀

▶ **Lemma 25.** *If G is a chordal graph, then, $ipacc(G) \leq 4$.*

Proof. Let r be an arbitrary vertex of G. Now, assume by contradiction that there is an isometric path P in G with endpoints u and v, such that $|A_r(P)| \geq 5$ in $\overrightarrow{G_r}$. Let $a_1, \ldots, a_5 \in A_r(P)$ be five antichain elements, that appear in this order while traversing P from u to v. We will eventually show that the existence of P implies that $\Delta_r(G) \geq 4$, contradicting Proposition 17. Let $\mathsf{d}(r,P) = i$ and x_u (resp. x_v) be a vertex such that $\mathsf{d}(r,x_u) = i$ (resp. $\mathsf{d}(r,x_v) = i$) and there is an oriented path from u (resp. v) to x_u (resp. x_v) in $\overrightarrow{G_r}$ (possibly, $u = x_u$ or $v = x_v$). Observe that x_u and x_v lie in the same cluster with respect to r. First, we prove the following claim.

▷ Claim 26. $\mathsf{d}(r,P) = \mathsf{d}(r,a_3)$.

Proof. Assume by contradiction that $\mathsf{d}(r,P) < \mathsf{d}(r,a_3)$. Refer to Figure 2 for an illustration of the different notations for this proof. Let $top \in V(P)$ be a vertex such that $\mathsf{d}(r,top) = \mathsf{d}(r,P)$. Recall that $\mathsf{d}(r,top) = i$. Now, let P_u be the (u,top)-subpath of P, and let P_v be the (v,top)-subpath of P. Observe that either $a_3 \in V(P_u)$ or $a_3 \in V(P_v)$. Without loss of generality, assume that $a_3 \in V(P_u)$.

Let P' be the (top,a_3)-subpapth of P_u, and let $top = u_1, \ldots, u_\ell = a_3$ be the ordering of vertices of $V(P')$, along the path P'. Let u_j, for $j \geq 1$, be the vertex with minimum index j such that $\mathsf{d}(r,u_j) = i$ and $\mathsf{d}(r,u_{j+1}) = i+1$. Note that u_j is distinct from a_3, and u_j can be the same as top.

(+) The (u_j,u)-subpath of P_u, say P_j, satisfies $V(P_j - \{u_j\}) \subseteq V(G_r^{i+1})$.

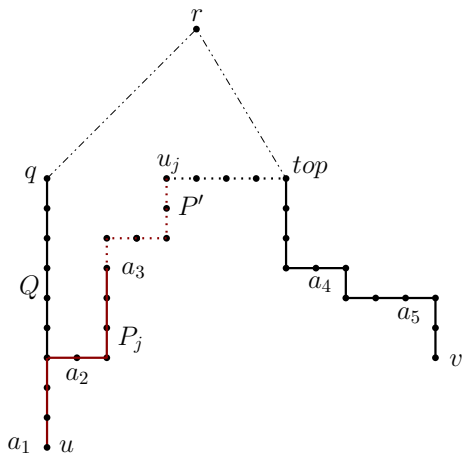


Figure 2 Proof of Claim 26. Path P' is dotted, and path P_j is red.

To prove $(+)$, assume by contradiction that there is a vertex $w \in V(P_j - \{u_j\})$ with $\mathsf{d}(r,w) = i$. Then, due to the definition of u_j, u_{j+1} is in the (u_j, w)-subpath, with $\mathsf{d}(r,w) = \mathsf{d}(r,u_j) = i$ and $\mathsf{d}(r,u_{j+1}) = i+1$. But this contradicts Observation 24 and completes the proof of $(+)$.

Let q be a vertex such that $\mathsf{d}(r,q) = i$ and there is an oriented path $\overrightarrow{Q}$ in $\overrightarrow{G_r}$ from u to q. Due to $(+)$, q is distinct from u. Let Q be the path obtained after removing the orientation of $\overrightarrow{Q}$. Note that Q is an isometric (q,u)-path in G. Also, note the following:

$(++)$ $V(Q - \{q\}) \subseteq V(G_r^{i+1})$.

Now, let us consider the (u_j, u)-subpath P_j of P_u defined above. Combining $(+)$ and $(++)$, we have that $V(Q) \cup V(P_j)$ forms a (u_j, q)-path, say T, such that $V(T - \{u_j, q\}) \subseteq V(G_r^{i+1})$. Due to Observation 23, $u_j q \in E(G)$. This implies that $\mathsf{d}(u,u_j) \leq \mathsf{d}(r,u) - \mathsf{d}(r,q) + 1$. But, since P_j is an isometric (u_j, u)-path and $|A_r(P_j)| \geq 3$ (indeed a_3, a_2, a_1 lie in P_j), due to Lemma 11, we have $\mathsf{d}(u,u_j) \geq \mathsf{d}(r,u) - \mathsf{d}(r,u_j) + 2 = \mathsf{d}(r,u) - \mathsf{d}(r,q) + 2$, which is a contradiction. ◁

Using Claim 26, we can now prove that $\mathsf{d}(x_u, x_v) \geq 4$. Since P is an isometric (u,v)-path and $a_3 \in V(P)$, we have:

$$\mathsf{d}(u,v) = \mathsf{d}(u,a_3) + \mathsf{d}(a_3,v) \tag{1}$$

Observe that the (u,a_3)-subpath of P contains at least three antichain elements of $\overrightarrow{G_r}$. Lemma 11 then implies $\mathsf{d}(u,a_3) \geq \mathsf{d}(r,u) - \mathsf{d}(r,a_3) + 2 = \mathsf{d}(r,u) - \mathsf{d}(r,x_u) + 2$. Since x_u belongs to an isometric (r,u)-path, we have $\mathsf{d}(x_u,u) = \mathsf{d}(r,u) - \mathsf{d}(r,x_u)$, and therefore:

$$\mathsf{d}(u,a_3) \geq \mathsf{d}(x_u,u) + 2 \tag{2}$$

By symmetry, we have:

$$\mathsf{d}(v,a_3) \geq \mathsf{d}(x_v,v) + 2 \tag{3}$$

Combining that $\mathsf{d}(u,v) \leq \mathsf{d}(u,x_u) + \mathsf{d}(x_u,x_v) + \mathsf{d}(x_v,v)$ with Equations 1-3, we get $\mathsf{d}(x_u,x_v) \geq 4$. Thus, $\Delta_r(G) \geq 4$, contradicting Proposition 17. This completes the proof. ◀

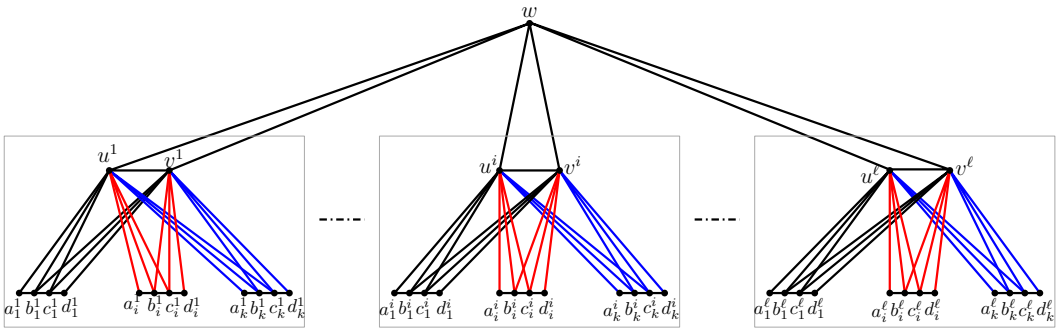


Figure 3 A tight-approximation example for chordal graphs.

3.3 Proof of Theorem 2 and tightness of our analysis

We now complete the proof of Theorem 2. Recall that Algorithm 1 takes as input a graph G and a vertex v and constructs the directed acyclic graph $\overrightarrow{G}_v$ and a directed path cover of $\overrightarrow{G_v}$.

First, we will prove Theorem 2a. Let G be a proper interval graph. Then, due to Lemma 22, G has a vertex v such that for any isometric path P of G, the cardinality of $A_v(P)$ in $\overrightarrow{G_v}$ is at most two. Let $\mathcal{P}_v$ be the isometric path cover returned by Algorithm 1 with G and v as input. Let $\mathcal{A}$ be the largest antichain set of $\overrightarrow{G_v}$ and OPT be a minimum cardinality isometric path cover of G. Due to Lemma 10, we have $|\mathcal{P}_v| \leq |\mathcal{A}| \leq 2|OPT|$. This completes the proof.

Proofs of Theorem 2b-f follow from similar arguments. In particular, by combining Lemmas 10 and 21 we have the proof of Theorem 2b. Combining Lemmas 10 and 25 we have the proof of Theorem 2c. Combining Lemmas 10 and 20 we have the proof of Theorem 2d. Combining Lemmas 10 and 18 we have the proof of Theorem 2e. Combining Lemmas 10 and 16 we have the proof of Theorem 2f.

In the following observations, we show that our analysis of Algorithm 1 is essentially tight for proper interval graphs, interval graphs and chordal graphs.

▶ **Observation 27.** *There exist chordal graphs whose isometric path antichain cover number is 4. Moreover, for any $c < 4$, Algorithm 1 cannot guarantee an approximation ratio of c for chordal graphs.*

Proof. For integers ℓ and k, consider the following construction for graph G_k^ℓ. Let $V(G_k^\ell) = \{w\} \cup \{u^i, v^i : i \in [\ell]\} \cup \{a_j^i, b_j^i, c_j^i, d_j^i : i \in [\ell], j \in [k]\}$ and $E(G_k^\ell) = \{wu_i, wv_i : i \in [\ell]\} \cup \{u^i a_j^i, u^i b_j^i, u^i c_j^i, v^i b_j^i, v^i c_j^i, v^i d_j^i : i \in [\ell], j \in [k]\} \cup \{a_j^i b_j^i, b_j^i c_j^i, c_j^i d_j^i : i \in [\ell], j \in [k]\}$. See Figure 3 for reference. Note that G_k^ℓ is a chordal graph and $ipacc(G_k^\ell) = 4$. Observe that the isometric path cover number of G is at most $\ell k + \ell + 1$. Indeed one such isometric path cover can be constructed as follows. For $i \in [\ell]$ and $j \in [k]$, consider the isometric paths $P_j^i = a_j^i\ b_j^i\ c_j^i\ d_j^i$. Also consider, for $i \in [\ell]$, the isometric paths $Q_i = u^i\ v^i$. Observe that

$$\mathcal{P} = \{w\} \bigcup_{i \in [\ell]} Q_i \bigcup_{\substack{i \in [\ell] \\ j \in [k]}} P_j^i$$

is an isometric path cover of G_k^ℓ of size $\ell k + \ell + 1$. Moreover, Algorithm 1 will return a solution of size at least $4k(\ell - 1)$ for G_k^ℓ. Indeed if Algorithm 1 has G_k^ℓ and w as the input, then it will return a solution of size $4\ell k$. Otherwise, there exists a $i \in [\ell], j \in [k]$ such that Algorithm 1 has G_k^ℓ and z as the input where $z \in \{u^i, v^i\} \cup \{a_j^i, b_j^i, c_j^i, d_j^i\}$. In this case, Algorithm 1 will

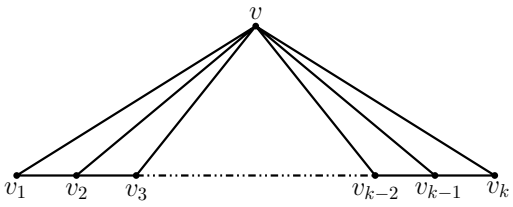


Figure 4 A tight-approximation example for interval graphs.

return a solution of cardinality at least $4k(\ell-1)$. This gives us an approximation ratio of $\frac{4k(\ell-1)}{\ell(k+1)+1}$. Now, for any $c<4$, we can set k and ℓ such that the approximation ratio is greater than c. ◀

▶ **Observation 28.** *There exist interval graphs whose isometric path antichain cover number is 3. Moreover, for any $c<3$, Algorithm 1 cannot guarantee an approximation ratio of c for interval graphs.*

Proof. To see this, consider the following graph. Let P_k be a path on k vertices $v_1,\ldots,v_k$ (where k is a multiple of 3). Let G be the graph obtained by adding a universal vertex v to P_k (*i.e.*, $V(G)=V(P_k)\cup\{v\}$ and $E(G)=E(P_k)\cup\{vv_i\colon i\in[k]\}$). See Figure 4 for reference. Note that G is an interval graph and $ipacc(()\,G)=3$. Moreover, isometric path cover number of G is $\frac{k}{3}+1$, and Algorithm 1 returns an isometric path cover of size at least $k-3$. Now, for any $c<3$, we can set k such that the approximation ratio $\left(\frac{3(k-3)}{k+1}\right)$ is greater than c. ◀

▶ **Observation 29.** *There exist proper interval graphs whose isometric path antichain cover number is 2. Moreover, for any $c<2$, Algorithm 1 cannot guarantee an approximation ratio of c for proper interval graphs.*

Proof. Let G be the complete graph on k vertices (where k is even). Note that G is a proper interval graph. Moreover, isometric path cover number of G is $\frac{k}{2}$, and Algorithm 1 returns an isometric path cover of size $k-1$. Now, for any $c<2$, we can set k such that the approximation ratio $\left(\frac{2(k-1)}{k}\right)$ is greater than c. ◀

4 An FPT algorithm for solution size on chordal graphs

In this section, we prove Theorem 3 using dynamic programming on tree decompositions. As the problem deals with shortest paths, it seems difficult to generally solve it on graphs of bounded treewidth, as it is not straightforwardly expressible in monadic second-order logic. Certain related problems like Geodetic Set are in fact W-hard for treewidth [20]. For chordal graphs however, we can exploit the structural properties of shortest paths to design such an algorithm. As a corollary, we show that this yields an FPT algorithm for the parameter solution size alone.

Indeed, we will prove the first part of Theorem 3 that Isometric Path Cover can be solved in time $2^{\mathcal{O}(2^k w^2)}\cdot n$ on chordal graphs of order n and treewidth w, where k is the solution size. To obtain the running time $2^{2^{\mathcal{O}(k)}}\cdot n$ as a corollary, note first that for chordal graphs, the treewidth w is equal to the clique number minus one (and the latter can be determined in polynomial time on this class). However, an isometric path can cover at most two vertices of any clique. Thus, if $k<(w+1)/2$, then we can return NO. Otherwise, the running time follows from the first running time.

We will use *nice tree decompositions*, a well-known tool for designing dynamic programming algorithms for graphs of bounded treewidth, that can be constructed optimally in linear time on chordal graphs, see [4, Section 4].

▶ **Definition 30.** *A* nice tree decomposition *of a chordal graph G is a rooted tree T where each node v is associated to a subset X_v of $V(G)$ called a* bag*, and each internal node has one or two children, with the following properties.*

1. *The nodes of T containing a given vertex of G form a nonempty connected subtree of T.*
2. *Any two adjacent vertices of G appear in the bag of a common node of T.*
3. *For each node v of T, X_v is a clique.*
4. *Each node of T belongs to one of the following types:* introduce*,* forget*,* join *or* leaf*.*
5. *A join node v has two children v_1 and v_2 such that $X_v = X_{v_1} = X_{v_2}$.*
6. *An introduce node v has one child v_1 such that $X_v \setminus \{x\} = X_{v_1}$, for some vertex $x \in X_v$.*
7. *A forget node v has one child v_1 such that $X_v = X_{v_1} \setminus \{x\}$, for some vertex $x \in X_{v_1}$.*
8. *A leaf node v is a leaf of T with $X_v = \{x\}$ for some vertex x of G.*
9. *The tree T is rooted at a leaf node r.*

For a nice tree decomposition and a node v, we define $G_{\leq v}$ as the subgraph of G induced by the vertices of the subtree of the decomposition rooted at v. We can similarly define $G_{<v} = G_{\leq v} - X_v$, $G_{\geq v} = G - G_{<v}$, and $G_{>v} = G - G_{\leq v}$.

Note that for a clique X and a vertex y, X can be partitioned into two (not necessarily both nonempty) sets of vertices according to their distances to y, as y has at most two distinct distance values to the vertices of X, with a difference of at most 1 between these values. Based on this, we give the following definition, inspired from [6].

▶ **Definition 31.** *For a clique X and a vertex y of G, we denote by* $\mathrm{close}(X, y)$ *the set of vertices of X that have minimal distance to y among the vertices of X, that is, for every vertex z of X, $\mathsf{d}(y, z) = d$ if z is in* $\mathrm{close}(X, y)$*, and $\mathsf{d}(y, z) = d + 1$ otherwise. We say that y is* close *to the set* $\mathrm{close}(X, y)$*.*

In a chordal graph, every maximal clique forms a clique cutset, and that clique will be associated to some node of the tree decomposition. As in most treewidth-based dynamic programming schemes, we will compute the potential solutions by bottom-up traversal of the tree decomposition. For this, we will define some *types* of solutions of Isometric Path Cover, depending on how they interact with a given bag. The number of types will be bounded by a function of k and w. We must then show how local solutions of a given type (if they exist) can be computed using the already computed information from the children.

Let us first give the key ideas needed for the dynamic programming scheme. We name the k paths $P_1, \ldots, P_k$. A *partial solution* for Isometric Path Cover with respect to a bag X_v of a node v of T, consists of k (possibly empty) subsets $P_1^v, \ldots, P_k^v$ of X_v of size at most 2, each representing the intersection of a path P_i with X_v, whose union equals X_v.

Making sure that an existing partial solution is extended so as to give an *induced* path cover is not too difficult, indeed, since the graph is chordal, if a path has a chord, that would give a cycle and thus there would be a triangle consisting of three vertices of the path. Necessarily, this triangle would be included in some bag, a contradiction. However, to make sure that the computed path is *isometric* is less trivial, but can be done due to the above definition of closeness. Indeed, we have the following lemma (due to space constraints, the proof is omitted: see the full version of the paper).

▶ **Lemma 32.** *Let G be a chordal graph and P be a path in G. The path P is isometric if and only if, for every clique X of G intersecting P and for every vertex y of P, there is exactly one vertex of $V(P) \cap X$ in* $\mathrm{close}(X, y)$*.*

Thus, following Lemma 32, for every bag X_v and for every subset X of X_v, we will keep track of whether each path P_i contains a vertex y with $\text{close}(X_v, y) = X$ in the previously computed partial solutions that can lead to the current partial solution. We will also keep track of whether the future partial solutions contain such a vertex. This information can be propagated along the bottom-up dynamic programming, together with the fact that the computed solutions must form a path cover. By Lemma 32, it will then be enough to check whether two partial solutions are compatible with respect to this information, to make sure they form a *valid* partial solution to Isometric Path Cover.

For a partial solution of node v, we define its *type* by the following information.

- The partial solution on X_v (*i.e.* the intersection of the k paths with X_v).
- For each path P_i and each vertex y of P_i in the partial solution of X_v, whether y is an endpoint of P_i, has a neighbour in P_i in $G_{<v}$, or in $G_{>v}$ (one can check that there are six distinct possibilities).
- For each path P_i, if P_i is not represented in the partial solution, a bit indicating whether P_i has been present in $G_{<v}$ or not (if yes, it can never be used in a future partial solution).
- For each path P_i, for each subset X of X_v, whether P_i has a vertex y in $G_{<v}$ with $\text{close}(X_v, y) = X$.
- For each path P_i, for each subset X of X_v, whether P_i has a vertex y in $G_{>v}$ with $\text{close}(X_v, y) = X$.

Note that the number of possible types for a node v is at most $k^{\mathcal{O}(w^2)} \times 6^{kw} \times 2^k \times k^{2^{\mathcal{O}(w)}} \times k^{2^{\mathcal{O}(w)}}$, which is dominated by $2^{k2^{\mathcal{O}(w)}}$.

The algorithm will consist of computing all tables in a bottom-up manner, and return YES if and only if the root node has an admissible partial solution type. To compute the table of a node, for each possible type, we need only to consider all (pairs of) types of the children nodes, and check their compatibility. We give all details in the full version.

5 Conclusion

We have studied the problem Isometric Path Cover in many subclasses of graphs of bounded treelength. Our main contribution is a polynomial-time algorithm to solve the problem that provides a constant-factor approximation on a very large class of graphs. It remains an interesting open question whether, for general graphs, there exists a polynomial-time constant-factor approximation algorithm for Isometric Path Cover, and whether Isometric Path Cover is FPT for solution size or treewidth. We also leave the complexity of Isometric Path Cover on interval graphs open. Other interesting classes on which to study Isometric Path Cover, and which seem challenging, can be found in Figure 1, for example split graphs or proper interval graphs.

Finally, we remark that some of our results also hold for the *partition* version of Isometric Path Cover, called Isometric Path Partition [23], where the isometric paths must be pairwise vertex-disjoint. This is the case for our NP-hardness proof for chordal graphs (indeed, all considered isometric path covers are in fact isometric path partitions), and the FPT algorithm for treewidth (indeed, it is not difficult to include in the constraints that the paths must form a partition). However, our approximation algorithm does not return a feasible solution for Isometric Path Partition, since it can produce overlapping paths.

References

1 I. Abraham, C. Gavoille, A. Gupta, O. Neiman, and K. Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM Journal on Computing*, 48(3):1120–1145, 2019.

2 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.

3 G. Andreatta and F. Mason. Path covering problems and testing of printed circuits. *Discrete Applied Mathematics*, 62(1-3):5–13, 1995.

4 R. Belmonte, P. A. Golovach, P. Heggernes, P. van't Hof, M. Kamiński, and D. Paulusma. Detecting fixed patterns in chordal graphs in polynomial time. *Algorithmica*, 69(3):501–521, 2014.

5 M. Cáceres, M. Cairo, B. Mumey, R. Rizzi, and A. I. Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 359–376. SIAM, 2022.

6 D. Chakraborty, S. Das, F. Foucaud, H. Gahlawat, D. Lajou, and B. Roy. Algorithms and complexity for geodetic sets on planar and chordal graphs. In *Proceedings of the 31st International Symposium on Algorithms and Computation, ISAAC 2020*, volume 181 of *LIPIcs*, pages 7:1–7:15, 2020.

7 D. G. Corneil, S. Olariu, and L. Stewart. Computing a dominating pair in an asteroidal triple-free graph in linear time. In *Workshop on Algorithms and Data Structures*, pages 358–368. Springer, 1995.

8 T. Davot, L. Isenmann, and J. Thiebaut. On the approximation hardness of geodetic set and its variants. In *Proceedings of the 27th International Computing and Combinatorics Conference, COCOON 2021*, volume 13025 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 2021.

9 R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

10 Y. Dourisboure and C. Gavoille. Tree-decompositions with bags of small diameter. *Discrete Mathematics*, 307(16):2008–2029, 2007.

11 V. Dujmović, G. Joret, P. Micek, P. Morin, T. Ueckerdt, and D. R. Wood. Planar graphs have bounded queue-number. *Journal of the ACM (JACM)*, 67(4):1–38, 2020.

12 Maël Dumas, Florent Foucaud, Anthony Perez, and Ioan Todinca. On graphs coverable by k shortest paths. In *Proceedings of the 33rd International Symposium on Algorithms and Computation, ISAAC 2022*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

13 D. C. Fisher and S. L. Fitzpatrick. The isometric number of a graph. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 38(1):97–110, 2001.

14 S. L. Fitzpatrick, R. J. Nowakowski, D. A. Holton, and I. Caines. Covering hypercubes by isometric paths. *Discrete Mathematics*, 240(1-3):253–260, 2001.

15 D. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

16 D. R. Fulkerson. Note on dilworth's decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7:701–702, 1956.

17 C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *ESA*, pages 476–487. Springer, 2001.

18 J. Guo, R. Niedermeier, and J. Uhlmann. Two fixed-parameter algorithms for vertex covering by paths on trees. *Information Processing Letters*, 106(2):81–86, 2008.

19 P. Heggernes, D. Meister, and C. Papadopoulos. A new representation of proper interval graphs with an application to clique-width. *Electronic Notes in Discrete Mathematics*, 32:27–34, 2009.

20 L. Kellerhals and T. Koana. Parameterized complexity of geodetic set. In *IPEC*, volume 180, pages 20:1–20:14, 2020.

21 C. V. G. C. Lima, V. F. dos Santos, J. H. G. Sousa, and S. A. Urrutia. On the computational complexity of the strong geodetic recognition problem, 2022. doi:10.48550/ARXIV.2208.01796.

22 W. Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *SODA*, pages 169–178, 2021. doi:10.1137/1.9781611976465.12.

23 P. D. Manuel. On the isometric path partition problem. *Discussiones Mathematicae: Graph Theory*, 41(4):1077–1089, 2021.

24 P. D. Manuel, S. Klavžar, A. Xavier, A. Arokiaraj, and E. Thomas. Strong geodetic problem in networks. *Discussiones Mathematicae Graph Theory*, 40(1):307–321, 2018.

25 A. Mohammed and F. F. Dragan. Slimness of graphs. *Discrete Mathematics & Theoretical Computer Science*, 21, 2019.

26 Simeon C. Ntafos and S. Louis Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, SE-5(5):520–529, 1979.

27 J. Pan and G. J. Chang. Isometric-path numbers of block graphs. *Information processing letters*, 93(2):99–102, 2005.

28 J. Pan and G. J. Chang. Isometric path numbers of graphs. *Discrete mathematics*, 306(17):2091–2096, 2006.

29 N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

30 M. Thiessen and T. Gaertner. Active learning of convex halfspaces on graphs. In *Proceedings of the 35th Conference on Neural Information Processing Systems, NeurIPS 2021*, volume 34, pages 23413–23425. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper/2021/file/c4bf1e24f3e6f92ca9dfd9a7a1a1049c-Paper.pdf.

31 R. van Bevern, R. Bredereck, L. Bulteau, J. Chen, V. Froese, R. Niedermeier, and G. J. Woeginger. Partitioning perfect graphs into stars. *Journal of Graph Theory*, 85(2):297–335, 2017.

Computation of Cycle Bases in Surface Embedded Graphs

Kyle Fox
University of Texas at Dallas, Richardson, TX, USA

Thomas Stanley
Unaffiliated, Dallas, TX, USA

Abstract

We present an $O(n^3g^2 \log g + m) + \tilde{O}(n^{\omega+1})$ time deterministic algorithm to find the minimum cycle basis of a directed graph embedded on an orientable surface of genus g. This result improves upon the previous fastest known running time of $O(m^3n + m^2n^2 \log n)$ applicable to general directed graphs.

While an $O(n^\omega + 2^{2g}n^2 + m)$ time deterministic algorithm was known for *undirected graphs*, the use of the underlying field $\mathbb{Q}$ in the directed case (as opposed to $\mathbb{Z}_2$ for the undirected case) presents extra challenges. It turns out that some of our new observations are useful for both variants of the problem, so we present an $O(n^\omega + n^2g^2 \log g + m)$ time deterministic algorithm for undirected graphs as well.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Computational geometry; Mathematics of computing → Graphs and surfaces

Keywords and phrases cycle basis, surface embedded graphs, homology

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.13

Funding *Kyle Fox*: Research partially supported by NSF grant CCF-1942597.

Acknowledgements Research by T. Stanley was partially performed while this author was a student at the University of Texas at Dallas.

1 Introduction

For a given connected undirected graph $G = (V, E)$, let $m = |E|$ and $n = |V|$ be the number of edges and vertices. We define a **cycle** to be a subset of the edges such that each vertex is incident to an even number of edges in the subset. It is known that cycles constitute a vector space with addition defined as symmetric difference of the edges and that this vector space is isomorphic to $\mathbb{Z}_2^{m-n+1}$.

We can form a vector space over cycles in directed graphs as well but we require some more complicated definitions. To this end, we represent a directed graph using its underlying *undirected* graph $G = (V, E)$ along with a mapping between each edge $e = uv$ and its two underlying **darts** $u{\to}v$ and $rev(u{\to}v) = v{\to}u$. One of these two darts is designated as the original/**canonical orientation** $\vec{e}$ of e. A **cycle** is a function $C : e \to \mathbb{Q}$ subject to certain restrictions. Informally, we could say the "amount" of cycle (read flow) entering each vertex is equal to the amount leaving. Formally, for each vertex v, we require $\sum_{e:\vec{e}=u\to v} C(e) = \sum_{e:\vec{e}=v\to w} C(e)$. Note that in general, cycles will have negative assignments to some edges. In other words, a cycle is allowed to travel "backwards" relative to the canonical orientation of an edge. Addition of cycles is defined to be an element-wise sum over the edges, and multiplication by a scalar $q \in \mathbb{Q}$ is element-wise multiplication by q over the edges. The cycles again form a vector space known to be isomorphic to $\mathbb{Q}^{m-n+1}$.

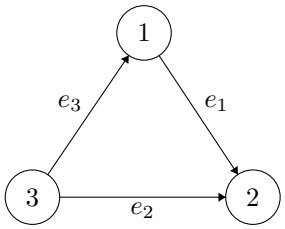


Figure 1 A small directed graph. Cycle sequence $\langle \vec{e_1}, rev(\vec{e_2}), \vec{e_3} \rangle$ corresponds to a cycle assigning 1 to edges e_1 and e_3 and -1 to edge e_2. The reversal of the sequence corresponds to a cycle assigning -1 to edges e_1 and e_3 and 1 to edge e_2.

We define a **cycle sequence** of G to be a sequence of darts $S = \langle v_0 \to v_1, v_1 \to v_2, \ldots, v_{k-1} \to v_k \rangle$ such that $v_0 = v_k$. Cycle sequence S corresponds to a cycle C_S where $C_S(e)$ is equal to the number of times $\vec{e}$ appears in S minus the number of times $rev(\vec{e})$ appears in S. We define the reversal of S as $rev(S) = \langle v_k \to v_{k-1}, v_{k-1} \to v_{k-2}, \ldots, v_1 \to v_0 \rangle$. Observe that $C_{rev(S)} = -C_S$. See Figure 1.

In both cases of an undirected or directed graph G, a **cycle basis** is a set of $d := m - n + 1$ independent cycles. It is well known that a cycle basis of G can be obtained from the fundamental cycles of any spanning tree of G. Given an assignment of non-negative weights $w : E \to \mathbb{Q}_{\geq 0}$ to the edges, we define the **weight** of a cycle C as $\sum_{e \in C} w(e)$ if G is undirected or $\sum_{e \in E} |C(e)| w(e)$ if G is directed. Note that the canonical orientations of the edges is irrelevant when computing the weight of a cycle. The **weight** of a cycle basis is defined as the sum of its constituent cycles' weights.

In undirected graphs, a **minimum cycle basis** is a cycle basis of minimum weight. Because we can always reduce the weight of a directed graph cycle by dividing it by a sufficiently large scalar, we define the **minimum cycle basis** of a directed graph as a minimum weight cycle basis in which every member corresponds to a cycle sequence (equivalently, every member has only integral assignments to the edges). We emphasize that the set of integral cycle bases in a directed graph and its underlying undirected graph may not be same. A counter example is given in Figure 2 by Hariharan, Kavitha, and Mehlhorn [12]. In this figure we look at three cycle sequences $\langle e_1, e_2, e_3, e_4 \rangle$, $\langle e_1, rev(e_5), rev(e_3), e_6 \rangle$, and $\langle e_2, e_6, rev(e_4), e_5 \rangle$. In the directed graph, the corresponding cycles are linearly independent. However, in the underlying undirected graph, the sum of two of these cycles equals the third, implying they are linearly dependent.

We do note the the problem of minimum directed cycle basis does have an alternative definition where cycles in the basis are required to follow edges in the correct direction. Using this variation the graph does not necessarily contain a cycle basis. This variation is not addressed in this paper. The minimum cycle basis has applications in many fields for both the directed [6, 10] and undirected [5, 16, 19] cases.

From the definition of independence in vector spaces, sets of independent cycles form a matroid. Therefore, one can use the standard greedy algorithm of sorting and eliminating to find the minimum cycle basis. However, the number of cycles in G is exponential in the undirected case and infinite in the directed case. Horton reduced the search space for the greedy algorithm to $O(mn)$ cycles by showing that every cycle in the minimum cycle basis must be a fundamental cycle of a shortest path tree, giving the first polynomial time algorithm [14]. Several other deterministic polynomial time algorithms have been given [1,7,13,17], the fastest being an $O(nm^2/\log n + n^2 m)$ time algorithm for the undirected case by Mehlhorn and Michail [17] and an $O(m^3 n + m^2 n^2 \log n)$ time algorithm for the

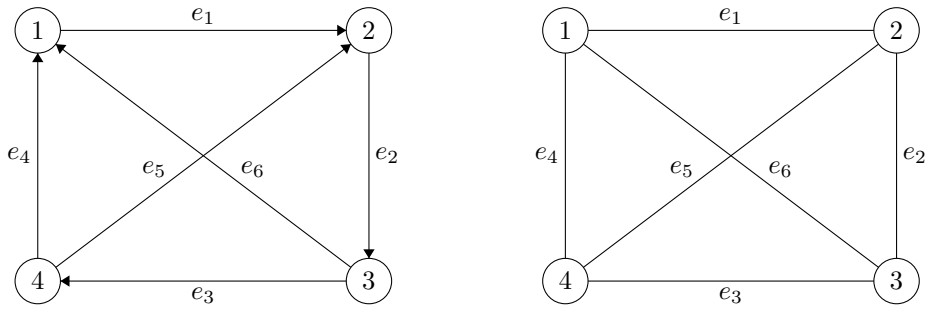


Figure 2 A directed graph with its underlying undirected graph [12].

directed case by Hariharan, Kavitha, and Mehlhorn [12]. There also exist faster $O(m^\omega)$ time randomized algorithms for both undirected and directed graphs where $O(m^\omega)$ is the time needed to multiply two $m \times m$ matrices [1, 18].

A surface or 2-manifold with boundary Σ is defined as a compact Hausdorff space such that every point lies in an open neighborhood homeomorphic to the Euclidean plane or the closed half plane. The boundary of the surface is the set of all points whose open neighborhood is homeomorphic to the closed half plane, and every boundary component is homeomorphic to the circle. A cycle *in the surface* is a continuous mapping from the unit circle to the surface, and the cycle is called simple if the mapping is injective. A surface is said to be orientable if it does not contain a subset homeomorphic to the Möbius band, and non-orientable otherwise. The genus of a surface, denoted g, is the maximum number of disjoint cycles in the surface such that their removal leaves a surface that is still connected. Two surfaces are homeomorphic if their genus, number of boundary components, and whether or not they are orientable all agree.

We can improve upon the deterministic running times by restricting ourselves to graphs that can be embedded on a surface of a certain genus. It is possible to bypass the matrix multiplication bound in planar graphs, ultimately resulting in a *near-linear* time algorithm that builds a data structure for quickly retrieving individual cycles [1, 3, 13]. The minimum cycle bases are identical in planar undirected and directed graphs, so these results work in both settings. Borradaile, Chambers, Fox, and Nayyeri [2] presented an $O(n^\omega + 2^{2g}n^2 + m)$ time algorithm for computing the minimum cycle basis in undirected graphs embedded on orientable surfaces of genus g.

1.1 Our results

We give an $O(n^3g^2 \log g + m) + \tilde{O}(n^{\omega+1})$ time deterministic algorithm to find the minimum cycle basis for a directed graph embedded in an orientable surface of genus g.[1] At a high level, we follow the same strategy used by others for computing minimum cycle bases [2, 7, 12]. We describe a way to represent each cycle as an integer vector of dimension d while also maintaining an ordered collection of d-dimensional *support vectors*. We maintain the property that after finding i members of the minimum cycle basis, the latter $d - i$ members of the collection are all orthogonal to the basis cycles' representations. The $i + 1$st cycle is simply the lightest cycle whose representation is *not* orthogonal to the $i + 1$st support vector.

[1] We use $\tilde{O}(\cdot)$ to hide terms polylogarithmic in n.

In general sparse directed graphs, a deterministic search for a single lightest non-orthogonal cycle takes $O(n^3 \log n)$ time [12]. Even in undirected graphs, such a search would take roughly quadratic time. However, Borradaile et al. [2] show how to perform the search in only $O(2^{2g}n)$ time given an undirected graph embedded on an orientable surface of g. Their idea is to partition a collection of candidate cycles into 2^{2g} subsets so that each subset of cycles nest. One can then use the nesting structure to search for the lightest non-orthogonal candidate cycle of any group in only $O(n)$ time.

We offer two main technical contributions on top of Borradaile et al.'s [2] algorithm to get our algorithm for directed graphs. First, we use a recent result of Greene [11] to argue that the number of subsets needed in the partition is actually $O(g^2 \log g)$ (we return to this point when we describe our second result). Second, we show how to extend the search within each group of candidates to work in the directed graphs. Doing so requires us to carefully acknowledge the orientation of edges and cycles as we do our searches while also dealing with the very large support vector elements that may arise over the course of our algorithm from no longer working over a finite field.

It turns out that our observation concerning the size of the candidate cycle partition applies to undirected graphs as well. Consequently, we get an $O(n^\omega + n^2 g^2 \log g + m)$ time deterministic algorithm to find the minimum cycle basis of a surface embedded undirected graph. The better running time requires no change to the algorithm of Borradaile et al. [2] beyond the observation which we will describe in the context of directed graphs, so we need not elaborate further beyond stating the relevant theorem.

▶ **Theorem 1.** *Let G be an undirected graph with n vertices and m non-negatively weighted edges, cellularly embedded in an orientable surface of genus g. We can deterministically compute a minimum-weight cycle basis of G in $O(n^\omega + n^2 g^2 \log g + m)$ time.*

2 Preliminaries

For a given graph $G = (V, E)$, an embedding of G on Σ is a mapping taking vertices to distinct points on Σ and edges to internally disjoint paths on Σ with endpoints that lie on their incident vertices' points. A face of the embedding is defined to be a maximally connected subset of Σ such that the subset does not intersect the embedded graph. If every face on an embedding is homeomorphic to an open disk we say the embedding is cellular. Only orientable cellular embeddings of graphs will be considered from now on. Without loss of generality, we also assume the surface has exactly one boundary component. Let ℓ denote the number of faces in an embedding of G. Based on our assumptions, Euler's formula guarantees $n - m + \ell = 1 - 2g$.

Every embedded graph G has a dual graph G^* which is constructed by creating a vertex for every boundary component in Σ as well as every face of G. Edges are then created between two vertices in G^* if the corresponding faces and boundary components are separated by an edge. We define the canonical orientation of edge e's dual to cross $\vec{e}$ from left to right. Finally, faces in G^* now correspond to vertices in G. The original graph is then known as the primal graph, where primal vertices are dual to dual faces, and dual vertices are dual to primal faces. We make no notational distinction between corresponding primal and dual objects in this paper.

2.1 Cycle signatures and homology

Let G be directed. A spanning tree of G is a subset of edges of G that form a tree containing every vertex of G. A tree-cotree decomposition is a partition of the edges of G into three sets, T a spanning tree of G, D a spanning tree of G^*, and L the leftover edges. Let $\beta = |L|$. Euler's formula implies $\beta = 2g$ [8, 9].

Let (T, L, D) be an arbitrary tree-cotree decomposition of G. Define c_i for $i \in \{1, \ldots, \beta\}$ to be either orientation of the unique simple cycle in G^* created by adding the ith edge in L to D. Let $f_{\beta+1}, \ldots f_{m-n+1}$ denote the faces of G. Define c_i for $i \in \{\beta+1, \ldots, m-n+1\}$ to be the simple path in D from f_i to the unique dual vertex for Σ's boundary. We define the **signature** $[e] \in \{-1, 0, 1\}^{m-n+1}$ of an edge e as a vector with the ith component defined as follows.

$$[e]_i = \begin{cases} 1 & \text{if } \vec{e} \text{ is in and oriented along } c_i \\ -1 & \text{if } rev(\vec{e}) \text{ is in and oriented along } c_i \\ 0 & \text{otherwise} \end{cases}$$

The **signature** of a cycle C is $[C] = \sum_{e \in E} C(e) \cdot [e]$. Borradaile, et al. [2] show that a similar cycle signature definition produces an isomorphism to the cycle space in an undirected graph. We use a nearly identical argument to prove the following lemma.

▶ **Lemma 2.** *Cycle signatures are an isomorphism between the cycle space of a directed graph and $\mathbb{Q}^{m-n+1}$. In particular, two cycles C and C' are equal if and only if $[C] = [C']$.*

Proof. The definition of cycle signatures immediately implies $[C + C'] = [C] + [C']$ and $[q \cdot C] = q \cdot [C]$ for any two cycles C and C'. Therefore, cycle signatures form a linear map.

Now, consider an arbitrary $w \in \mathbb{Q}^{m-n+1}$. From w, we will construct a cycle C such that $[C] = w$, implying cycle signatures are a surjection. Combined with them being a linear map between equal dimensional vector spaces, we conclude they must be an isomorphism. For each $i \in \{1, \ldots, \beta\}$, let C^i correspond to the unique simple cycle sequence in G created from adding the ith edge of L as defined above to T, oriented so that $[C^i]_i = 1$. Next, for each $i \in \{\beta+1, \ldots, m-n+1\}$, let C^i correspond to the boundary of face f_i as defined above, again oriented so that $[C^i]_i = 1$. Finally, let $C = \sum_{i=1}^{m-n+1} w_i \cdot C^i$.

Fix any $i, j \in \{1, \ldots, m-n+1\}$ such that $i \neq j$. If $i \in \{1, \ldots, \beta\}$, then C^i completely avoids the dual cycle or path c_j, implying $[C^i]_j = 0$. If $i \in \{\beta+1, \ldots, m-n+1\}$, then either C^i avoids c_j or c_j has exactly one dart entering f_i and one dart leaving f_i, again implying $[C^i]_j = 0$. We conclude $[C]_i = w_i$ for all i. ◀

The homology of G is an algebraic description of the topology of the surface as well as G's embedding. We are only concerned with the one-dimensional cellular homology over the finite field $\mathbb{Z}_2$ and use the underlying undirected graph when referencing the homology of G. We say a cycle sequence or its corresponding cycle is null-homologous if it is the boundary of a subset of faces. Two cycle sequences are homologous if their symmetric difference is null-homologous. These definitions allows us to partition the cycle sequences of G into 2^{2g} homology classes. We define the operation $G \nmid S$ as cutting both G and Σ along some cycle sequence S, creating two copies of S. Cutting G using any cycle sequence from the null-homology class cuts Σ into two separate surfaces, and cutting G along any two non-crossing sequences in the same homology class cuts Σ into two separate surfaces. We also define the **homology signature** $[e]^h \in \{0, 1\}^\beta$ of an edge e as a vector with the ith component defined as follows.

$$[e]_i^h = \begin{cases} 1 & \text{if } \vec{e} \text{ is in } c_i \\ 0 & \text{otherwise} \end{cases}$$

The **homology signature** of a cycle C is $[C]^h$ the bitwise exclusive-or of the homology signatures of its edges. We only use the homology signature to seperate the cycles by homology class and therefore do not use the direction of the edges to define the signature. Therefore this definition matches the undirected case of Borradaile, et al. [2].

2.2 Simplifying assumptions

We assume $g = O(n^{1-\varepsilon})$ for some constant $\varepsilon > 0$; otherwise our algorithms make no improvements upon what is known for general graphs. If G has no faces of degree 1 or 2, then Euler's formula implies the number of edges and faces to be $O(n)$. We guarantee this property of G as follows. If an edge e bounds a face of degree 1 on one side, then $S = \langle \vec{e} \rangle$ corresponds to the lightest cycle for which $C(e) = 1$. Any other cycle with $C(e) \neq 0$ can be made cheaper by setting $C(e)$ to 0, so we can safely assume C_S is in a minimum basis and remove e from G.

Suppose a face f has degree 2. If f is bounded twice by the same edge, then G must be embedded in the sphere and consist of only that single edge. There are no non-zero cycles, so we terminate the algorithm. Otherwise, f is bound by two distinct edges e_1 and e_2. Assume without loss of generality that $w(e_1) \leq w(e_2)$, and let $u \to v = \vec{e}_2$. Let σ denote the shortest path in G from u to v, and let $S = \vec{e}_2 \circ rev(\sigma)$, the concatenation of $\vec{e}_2$ with $rev(\sigma)$. Cycle C_S is the lightest cycle for which $C(e_2) = 1$, so we may assume it belongs to a minimum cycle basis. Observe $w(\sigma) \leq w(e_1) \leq w(e_2)$. Any other cycle C with $C(e_2) \neq 0$ can be made strictly lighter by adding or subtracting an appropriate multiple of C_S, so we may then remove e_2 from G.

We can guarantee all faces have degree at least 3 by computing all-pairs shortest paths in the subgraph of G that includes only the lightest member of each set of parallel edges in $O(n^2 \log n + m)$ time. Removing edges as described above takes $O(m)$ additional time total. From here on, we assume there are $O(n)$ edges and faces.

3 Algorithm

Our algorithm computes cycles of the minimum basis one by one. We do this by maintaining a set of support vectors that form the basis for the subspace orthogonal to the set of cycles already computed for the basis. To compute a new cycle in the basis, we choose a support vector that we have not used so far and find the cycle of minimum weight that is not orthogonal the chosen support vector. The unchosen support vectors are then updated so they remain orthogonal to the current incomplete basis we have computed. This is the method that many algorithm have used to compute minimum cycle bases [2, 7, 12].

Specifically, our algorithm uses the prime field modifications for directed graphs made by Hariharan, Kavitha, and Mehlhorn for dealing with the potentially large numbers generated by the coefficients from $\mathbb{Q}$ [12]. For choosing the non-orthogonal cycle our algorithm follows the basic idea of Borradaile, et al. [2] of constructing so-called *region trees* based on homology classes to improve the cycle selection time. However, modifications must be made to the cycle selection procedure to account for the coefficients from $\mathbb{Q}$.

3.1 Computing Support Vectors

The method of computing support vectors that are used to calculate the minimum cycle basis follows from the following theorem given and proved by Hariharan, Kavitha, and Mehlhorn [12]. We have adapted their theorem to use our cycle signatures. Its proof requires no changes thanks to the isomorphisms between various representations of the cycles.

▶ **Theorem 3.** *Integral cycles $C_1, \dots C_d$ form a minimum cycle basis if there are vectors $N_1 \dots N_d$ in $\mathbb{Q}^m$ such that for all i, $1 \leq j \leq i$:*

1. *Prefix orthogonality: $\langle N_i, [C_i] \rangle = 0$ for all j, $1 \leq j < i$.*
2. *Nonorthogonality: $\langle N_i, [C_i] \rangle \neq 0$.*
3. *Lightness: C_i is a lightest integral cycle with $\langle N_i, [C_i] \rangle \neq 0$*

Algorithm 1 is a simple deterministic algorithm that was given by Kavitha and Mehlhorn to compute the N_i's and C_i' [15].

Algorithm 1 An algorithm to compute N_i's and C_i's.

$N_1, \dots, N_d \leftarrow \hat{u_1}, \dots \hat{u_d}$ ▷ ($\hat{u_d}$ has a 1 in the ith position and 0's everywhere else)
for $i \leftarrow 1$ to d **do**
  $C_i \leftarrow$ lightest cycle with non-zero dot product with N_i
  for $j \leftarrow i+1$ to d **do**
    $N_j \leftarrow N_j - N_i \frac{\langle [C_i], N_j \rangle}{\langle [C_i], N_i \rangle}$
    $N_j \leftarrow N_j \frac{\langle [C_i], N_i \rangle}{\langle [C_{i-1}], N_{i-1} \rangle}$
  end for
end for

The correctness of this algorithm is based on a lemma given and proved by Kavitha and Mehlhorn [15].

▶ **Lemma 4.** *For any i, at the end of iteration $i-1$, the vectors $N_i, \dots, N_d$ are orthogonal to $[C_1], \dots, [C_{i-1}]$ and moreover for any j with $i \leq j \leq d$,*

$$N_j = \langle [C_{i-1}], N_{i-1} \rangle (x_{j,1}, \dots, x_{j,i-1}, 0, \dots, 0, 1, 0, \dots, 0)$$

where 1 occurs in the jth coordinate and the vector $\boldsymbol{x} = (x_{j,1}, \dots, x_{j,i-1})$ is the unique solution to the set of equations:

$$\begin{pmatrix} \tilde{C}_1^T \\ \vdots \\ \tilde{C}_{i-1}^T \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} -c_{1,j} \\ \vdots \\ -c_{i-1,j} \end{pmatrix}$$

Where $\tilde{C}_k$, $1 \leq k < i$, is the restriction of $[C_k]$ to its first $i-1$ coordinates and $c_{k,j}$ is the jth coordinate of $[C_k]$.

Furthermore, the running time of this algorithm was shown by Kavitha and Mehlhorn to be $\tilde{O}(m^4) + mO(\text{cycle})$, where $O(\text{cycle})$ is the time taken to find the lightest non-orthogonal integral cycle to N_i [15].

This simple algorithm was then improved upon by Hariharan, Kavitha, and Mehlhorn. By using a divide-and-conquer approach, the calculations spent updating the N_i vectors can be done in bulk [12]. Algorithm 2 gives the recursive step from index l to index h is as follows.

Algorithm 2 A faster algorithm to compute N_i's and C_i's.

$mid \leftarrow \lceil (l+h)/2 \rceil$
Find cycles $C_l, \ldots, C_{mid}$ using $N_l, \ldots, N_{mid}$ recursively
Update the vectors $N_{mid+1}, \ldots, N_h$
Find cycles $C_{mid+1}, \ldots, C_h$ using $N_{mid+1}, \ldots, N_h$ recursively

To compute the minimum cycle basis, we call this algorithm with $N_1, \ldots, N_d$ initialized to the first d unit vectors, $l = 1$, and $h = d$. Our goal when updating the vectors is to make the vectors $N_{mid+1}, \ldots, N_h$ orthogonal to the newly computed cycles $C_l, \ldots, C_{mid}$. To update the vectors, we make the following definitions.

- $A \in \mathbb{Q}^{k \times m}$, A's ith row is $[C_{l+i-1}]$
- $D \in \mathbb{Q}^{(h-k)\times(h-k)}$, D has the value $\langle N_{mid}, [C_{mid}] \rangle / \langle N_{l-1}, [C_{l-1}] \rangle$ in every diagonal
- $X \in \mathbb{Q}^{k \times (h-k)}$
- $N_d \in \mathbb{Q}^{m \times (h-k)}$, N_d's jth column is N_{mid+j}
- $N_u \in \mathbb{Q}^{m \times k}$, N_u's jth column is N_{l+j-1}

As shown by Hariharan, Kavitha, and Mehlhorn, we can update the vectors by solving the following system for X [12]:

$$AN_dD = -AN_uX$$

By construction (NN_u) is lower triangular with non-zero diagonal entries, and therefore is invertible. Hence we can write

$$X = -(AN_u)^{-1}AN_dD.$$

Finally our updated vectors $N_{mid+1}, \ldots, N_h$ can be found by computing $N_uX + N_dD$. This process of updating the vectors takes $O(nk^{\omega-1})$ arithmetic operations in total, where n^ω is the time it takes to multiply two $n \times n$ matrices using fast matrix multiplication. However, because we are in a directed graph, the elements of $(AN_u)^{-1}$ can be as large as $d^{\Theta(d^2)}$. Even assuming a model of computation that allows for constant time operations on words of up to $O(\log n)$ bits, we see each arithmetic operation can take up to $\tilde{\Theta}(d^2)$ time. Therefore, we get a runtime of $\tilde{\Theta}(n^{\omega+2})$ for the outermost step which is slower than the simpler algorithm for directed graphs [12].

In order to solve the problem of large intermediate elements we run the above algorithm over a ring $\mathbb{Z}_R$ where R is a specially chosen prime. Working over this ring allows us to do arithmetic operations in $O(1)$ time each. In order to be able to recover our N_j vectors, we must choose a R such that R is relativity prime to $\langle N_l, [C_l] \rangle, \langle N_{l+1}, [C_{l+1}] \rangle, \ldots, \langle N_{mid}, [C_{mid}] \rangle$ (to ensure AN_u is invertible in $\mathbb{Z}_R$), and relativity prime to $\langle N_{l-1}, [C_{l-1}] \rangle$ (to ensure that $\langle N_{mid}, [C_{mid}] \rangle / \langle N_{l-1}, [C_{l-1}] \rangle$ is well defined in $\mathbb{Z}_R$) [12]. We can select R for each iteration of the recursive step using Algorithm 3.

Pre-computing d^2 primes takes $\tilde{O}(d^2)$ time, and pre-computing the products $P_1, \ldots, P_d$ takes $\tilde{O}(d^3)$ time. The algorithm to select R runs in $\tilde{O}(d^2)$ time [12]. All together, the total time complexity for a single update step with modulo arithmetic is $\tilde{O}(n^2k^{\omega-1} + d^2k)$ or $\tilde{O}(n^2k^{\omega-1})$.

3.2 Finding the Minimum Cycle

What remains is to find the integral cycle of minimum weight that is non-orthogonal to a given support vector. To do this quickly, we modify an algorithm given by Borradaile, et al. [2] for undirected graphs. Their algorithm first computes a set of $O(2^{2g}n)$ candidate cycles that

Algorithm 3 An algorithm select a suitable R.

Require: $p_1, \ldots, p_{d^2}$, primes each of which is at least d, the products $P_1 = p_1 \ldots p_d, P_2 = p_1 \ldots p_{2d}, \ldots, P_d = p_1 \ldots p_{d^2}$ precomputed before running algorithm.
$L \leftarrow \langle N_{l-1}, [C_{l-1}]\rangle \langle N_l, [C_l]\rangle \ldots \langle N_{mid}, [C_{mid}]\rangle$
Binary search $P_1 \ldots P_d$ to find the smallest $s \geq 0$ such that $P_{s+1} \nmid L$
Determine a $p \in \{p_{sd+1}, \ldots, p_{sd+d}\}$ such that $p \nmid L$
return p^d

contain every member of some minimum cycle basis. The cycles are then partitioned into 2^{2g} sets and a tree representation of each set is built in $O(n^2)$ time. Each tree can then be searched in $O(n)$ time to find the minimum weight non-orthogonal cycle for that tree's subset of cycles [2].

We first show that it suffices to consider essentially the same small set of candidate cycles. A **Horton cycle** is defined to be a simple cycle sequence given by a shortest x, u-path, a shortest x, v-path, and an edge uv for some vertices x, u, and v. The set of all Horton cycles on a graph is given by the set of $m - n + 1$ fundamental cycles of the n shortest path trees [14]. A cycle sequence S is said to be **isometric** if for all vertices x and y appearing along S, the set of S's edges contain a shortest x, y-path.

▶ **Theorem 5.** *There exists a minimum cycle basis of directed graph G where every member corresponds to an isometric Horton cycle.*

Proof. Let $\mathcal{C}$ be a minimum cycle basis of G, let $C \in \mathcal{C}$, and let $\mathcal{C}' = \mathcal{C} \setminus \{C\}$. For any three paths α, β, and γ in G, we observe that if $C_{\alpha \circ rev(\beta)}$ and $C_{\beta \circ rev(\gamma)}$ are both dependent on cycles in $\mathcal{C}'$, then $C_{\alpha \circ rev(\gamma)}$ is dependent on cycles in $\mathcal{C}'$ by simple algebra on the cycle signatures. Therefore, the cycle sequences corresponding to dependent cycles follow the *three path condition* as defined by Cabello, Colin de Verdière, and Lazarus [4]. They show there exists a shortest cycle sequence *not* in the family corresponding to dependent cycles that is a fundamental cycle of a shortest paths tree, and that the root of this shortest paths tree can be any vertex of the cycle sequence. Such a cycle sequence is an isometric Horton cycle. ◀

From here on, we must assume all shortest paths are unique, and this assumption can be enforced without increasing our already-quadratic running time [13]. We compute all Horton cycles of the graph and extract the isometric cycles as shown by Amaldi et al [1]. As they are both dependent upon one-another, we keep only one of each pair of an isometric Horton cycle and its reversal.

Borradaile, et al. [2] show the isometric cycles can be partitioned into subsets of size $O(n)$, each corresponding to one of the 2^{2g} different homology classes. However, we observe that the number of non-empty classes is much smaller. Uniqueness of shortest paths implies that if two isometric cycles intersect, they do so along a single shortest path [3]. A recent result of Greene [11] implies that a set of cycle sequences that pairwise intersect (or cross) at most once must live in at most $O(g^2 \log g)$ *homotopy* classes. Homotopy is a finer relation between cycle sequences than homology, implying the number of non-empty homology classes for a collection of isometric cycles to be $O(g^2 \log g)$ as well. In particular, our minimum cycle basis algorithm need only consider $O(ng^2 \log g)$ candidate cycles total.

In order to follow the approach of Borradaile, et al. [2], we must first convert their algorithm to use cycle signatures in $\mathbb{Q}^d$ instead of $\mathbb{Z}^d$. As shown by Hariharan, Kavitha, and Mehlhorn using Hadamard's inequality the support vector can have elements of up to size $d^{d/2}$ [12] so naively modifying the algorithm will lead to problems with the speed of

arithmetic. We use a similar approach to that of Hariharan, Kavitha, and Mehlhorn [12] and first find the minimum weight cycles that are non-orthogonal to our chosen support vector modulo some prime p.

We first describe how to construct the region trees that will aid us in our search. We note that this step is done once before running the main algorithm and does not depend on the prime p. We compute the homology signatures for the candidate cycles and split them into their $O(g^2 \log g)$ homology classes. Then for each homology class, we compute its own region tree. Each vertex v of a region tree corresponds to a set of faces F^v, and each edge e corresponds to a candidate cycle C^e. A region tree $T_{\mathcal{S}}$ also has a designated cycle denoted $T^0_{\mathcal{S}}$. The **region trees** are defined as the result of Algorithm 4.

Algorithm 4 An algorithm to create a region tree for a homology class.

Require: Non-empty set of isometric Horton cycles $\mathcal{S}$ all belonging to the same homology class and the graph G
$T_{\mathcal{S}}$ starts out with one vertex v with F^v equal to all faces of G
if members of $\mathcal{S}$ have non-trivial homology **then**
  Choose an arbitrary $S_0 \in \mathcal{S}$
  $T_{\mathcal{S}}$ has a single edge e looping on v with $C^e = C_{S_0}$
  $G' \leftarrow G \nmid S_0$
  for $S \in \{\mathcal{C} \setminus S_0\}$ **do**
    $G' \leftarrow G' \nmid S$
    Cutting G' splits some component of G' into two new components
    Split vertex v corresponding to cut component into two new vertices with corresponding faces from the newly created components
    Assign C_S to the new edge
  end for
  $T^0_{\mathcal{S}} \leftarrow C_{S_0}$
  Remove edge corresponding to S_0
  Root $T_{\mathcal{S}}$ at the vertex whose region contains the boundary
else
  $G' \leftarrow G$
  for $S \in \mathcal{S}$ **do**
    Cutting G' splits some component of G' into two new components
    Split vertex corresponding to cut component into two new vertices with corresponding faces from the newly created components
    Assign C_S to the new edge
  end for
  $T^0_{\mathcal{S}}$ is assigned the 0-cycle
  Root $T_{\mathcal{S}}$ at the vertex whose region contains the boundary
end if
Negate all cycles as needed so the root region lies to the right of their corresponding sequences

In Algorithm 4, the operation $G \nmid \gamma$ takes $O(n)$ time, so the entire algorithm takes $O(n^2)$ time for each region tree. Therefore, we can preprocess G in $O(n^2 g^2 \log g)$ time. Examples of constructed region trees can be seen in Figures 3 and 4.

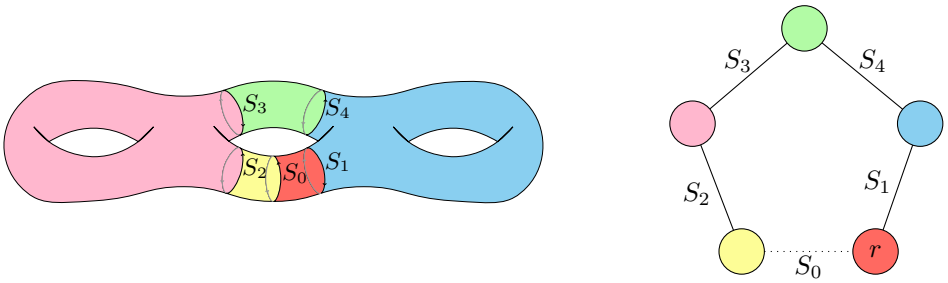


Figure 3 A region tree for a set of edges with non-trivial homology [2]. The root is depicted with r.

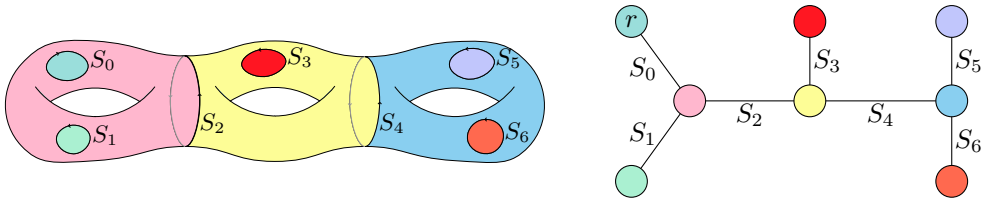


Figure 4 A region tree for a set of edges with null-homologous homology signature [2]. The root is depicted with r.

We can use the region trees to find a shortest cycle that is non-orthogonal modulo p with a given support vector N as shown in Algorithm 5. This algorithm is based on that of Borradaile, et al. [2] with modifications to account for the prime p and the importance of cycle orientations. Given a region tree $T_{\mathcal{S}}$, we start by computing $\langle N, [T^0_{\mathcal{S}}]\rangle$. We then travel the region tree in postorder, computing the inner product for each edge's cycle by adding the contributions from only the components of the cycle's signature that differ from those of the children edge's cycles. Recall the definition of the dual paths c_i used to define the cycle signatures in Section 2. In order to do add inner product contributions consistently, we need to know whether each new cycle C we consider crosses a dual path c_i a different number of times than $T^0_{\mathcal{S}}$. Fortunately, this number changes precisely when we consider the parent edge of the region containing face f_i. And because Algorithm 4 orients the cycles' sequences so the root region of $T_{\mathcal{S}}$ lies to each sequence's right, the net number of crossings and thus the corresponding component of the cycle signature changes by exactly 1. Algorithm 5 considers each face of the graph at most once, so the total runtime to walk up the tree is $O(n)$. We must run this algorithm once for every region tree, so the total runtime to find a non-orthogonal cycle modulo p is $O(ng^2 \log g)$.

In order to obtain a lightest non-orthogonal cycle from a collection of lightest non-orthogonal cycles modulo p, we first pre-compute primes $p_1, \ldots, p_{d/2}$ each of which is at least d. The ring $\mathbb{Z}_{\prod p_i}$ is isomorphic to $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_{d/2}}$, which implies that any non-zero element whose magnitude is less than $\prod_{i=1}^{d/2} p_i$ is mapped to a tuple of values that is not the zero vector. Therefore, if we run our algorithm for cycle searching $d/2$ times, once for each prime we pre-computed, each cycle that is non-orthogonal will also be non-orthogonal for some $p \in \{p_1, \ldots, p_{d/2}\}$. We get a total runtime of $O(n^2 g^2 \log g)$ to find a lightest non-orthogonal cycle for some given support vector.

Algorithm 5 An algorithm find the lightest cycle non-orthogonal to a given support vector N modulo p for a given region tree $T_{\mathcal{S}}$.

Require: Region tree $T_{\mathcal{S}}$
  $m \leftarrow \infty$ ▷ The current minimum weight
  $C \leftarrow$ NULL ▷ The current cycle referring to the minimum weight
  for edge $e \in T_{\mathcal{S}}$ in postorder **do**
    if e goes to a leaf **then**
      $z_e \leftarrow \langle N, [T^0_{\mathcal{S}}] \rangle$
    else
      $z_e \leftarrow 0$
    end if
    for child edge e' of e **do**
      $z_e \leftarrow z_e +_p z_{e'}$
    end for
    for $f_i \in F(\text{bottom}(e))$ **do**
      $z_e \leftarrow z_e +_p N^i$
    end for
    if $z_e \neq 0$ and $w(C^e) < m$ **then** $m \leftarrow w(C^e)$ and $C \leftarrow C^e$
  end for
  return C

3.3 Final analysis

Let $T(k)$ denote the time to run Algorithm 2 in our setting when $h - l + 1 = k$.

$$T(k) = \begin{cases} 2T(k/2) + \tilde{O}(n^2 k^{\omega-1}) & \text{if } k > 1 \\ n^2 g^2 \log g & \text{if } k = 1 \end{cases}$$

This recurrence solves to $O(n^3 g^2 \log g) + \tilde{O}(n^{\omega+1})$. Including the $O(m)$ time needed to guarantee all faces have degree 3 or greater, we get a total running time of of $O(n^3 g^2 \log g + m) + \tilde{O}(n^{\omega+1})$. As in Borradaile et al. [2], any method to improve the speed of selecting support vectors would improve the time required to find a minimum cycle basis.

We conclude with a theorem summarizing our main result.

▶ **Theorem 6.** *Let G be a directed graph with n vertices and m non-negatively weighted edges, cellularly embedded in an orientable surface of genus g. We can deterministically compute a minimum-weight cycle basis of G in $O(n^3 g^2 \log g + n^{\omega+1} + m)$ time.*

References

1 Edoardo Amaldi, Claudio Iuliano, Tomasz Jurkiewicz, Kurt Mehlhorn, and Romeo Rizzi. Breaking the o(m2n) barrier for minimum cycle bases. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 301–312, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

2 Glencora Borradaile, Erin Wolf Chambers, Kyle Fox, and Amir Nayyeri. Minimum cycle and homology bases of surface-embedded graphs. *J. Comput. Geom.*, 8(2):58–79, 2017. doi:10.20382/jocg.v8i2a4.

3 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min *st*-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3):16:1–16:29, 2015. doi:10.1145/2684068.

4 Sergio Cabello, Éric Colin de Verdière, and Francis Lazarus. Finding shortest non-trivial cycles in directed graphs on surfaces. In *Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry*, SoCG '10, pages 156–165, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1810959.1810988.

5 A. C. Cassell, J. C. De C. Henderson, K. Ramachandran, and Alec Westley Skempton. Cycle bases of minimal measure for the structural analysis of skeletal structures by the flexibility method. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 350(1660):61–70, 1976. doi:10.1098/rspa.1976.0095.

6 L. Chua and Li-Kuan Chen. On optimally sparse cycle and coboundary basis for a linear graph. *IEEE Transactions on Circuit Theory*, 20(5):495–503, 1973. doi:10.1109/TCT.1973.1083752.

7 J. Coelho de Pina. *Applications of shortest path methods*. PhD thesis, University of Amsterdam, 1995.

8 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 599–608, USA, 2003. Society for Industrial and Applied Mathematics.

9 Jeff Erickson and Amir Nayyeri. Minimum cuts and shortest non-separating cycles via homology covers. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 1166–1176, USA, 2011. Society for Industrial and Applied Mathematics.

10 Petra Gleiss, Josef Leydold, and Peter Stadler. Circuit bases of strongly connected digraphs. *Santa Fe Institute, Working Papers*, 23, January 2001. doi:10.7151/dmgt.1200.

11 Joshua Evan Greene. On loops intersecting at most once. *Geometric and Functional Analysis*, 29(6):1828–1843, December 2019. doi:10.1007/s00039-019-00517-0.

12 Ramesh Hariharan, Telikepalli Kavitha, and Kurt Mehlhorn. Faster algorithms for minimum cycle basis in directed graphs. *SIAM Journal on Computing*, 38(4):1430–1447, 2008. doi:10.1137/060670730.

13 David Hartvigsen and Russell Mardon. The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. *SIAM J. Discret. Math.*, 7(3):403–418, August 1994. doi:10.1137/S0895480190177042.

14 Joseph Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM J. Comput.*, 16:358–366, April 1987. doi:10.1137/0216026.

15 Telikepalli Kavitha and Kurt Mehlhorn. Algorithms to compute minimum cycle basis in directed graphs. *Theory of Computing Systems*, 40:485–505, June 2007. doi:10.1007/s00224-006-1319-6.

16 Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., USA, 1997.

17 Kurt Mehlhorn and Dimitrios Michail. Minimum cycle bases: Faster and simpler. *ACM Trans. Algorithms*, 6(1), December 2010. doi:10.1145/1644015.1644023.

18 Abhishek Rathod. Fast algorithms for minimum cycle basis and minimum homology basis. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*, volume 164 of *LIPIcs*, pages 64:1–64:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.64.

19 Geetika Tewari, Craig Gotsman, and Steven Gortler. Meshing genus-1 point clouds using discrete one-forms. *Computers & Graphics*, 30:917–926, December 2006. doi:10.1016/j.cag.2006.08.019.

Computing Homomorphisms in Hereditary Graph Classes: The Peculiar Case of the 5-Wheel and Graphs with No Long Claws

Michał Dębski
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Zbigniew Lonc
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Karolina Okrasa
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Marta Piecyk
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Paweł Rzążewski
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Abstract

For graphs G and H, an H-coloring of G is an edge-preserving mapping from $V(G)$ to $V(H)$. In the H-Coloring problem the graph H is fixed and we ask whether an instance graph G admits an H-coloring. A generalization of this problem is H-ColoringExt, where some vertices of G are already mapped to vertices of H and we ask if this partial mapping can be extended to an H-coloring.

We study the complexity of variants of H-Coloring in F-free graphs, i.e., graphs excluding a fixed graph F as an induced subgraph. For integers $a, b, c \geqslant 1$, by $S_{a,b,c}$ we denote the graph obtained by identifying one endvertex of three paths on $a+1$, $b+1$, and $c+1$ vertices, respectively. For odd $k \geqslant 5$, by W_k we denote the graph obtained from the k-cycle by adding a universal vertex.

As our main algorithmic result we show that W_5-ColoringExt is polynomial-time solvable in $S_{2,1,1}$-free graphs. This result exhibits an interesting non-monotonicity of H-ColoringExt with respect to taking induced subgraphs of H. Indeed, W_5 contains a triangle, and K_3-Coloring, i.e., classical 3-coloring, is NP-hard already in claw-free (i.e., $S_{1,1,1}$-free) graphs. Our algorithm is based on two main observations:

1. W_5-ColoringExt in $S_{2,1,1}$-free graphs can be in polynomial time reduced to a variant of the problem of finding an independent set intersecting all triangles, and
2. the latter problem can be solved in polynomial time in $S_{2,1,1}$-free graphs.

We complement this algorithmic result with several negative ones. In particular, we show that W_5-Coloring is NP-hard in P_t-free graphs for some constant t and W_5-ColoringExt is NP-hard in $S_{3,3,3}$-free graphs of bounded degree. This is again uncommon, as usually problems that are NP-hard in $S_{a,b,c}$-free graphs for some constant a, b, c are already hard in claw-free graphs

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Graph algorithms analysis

Keywords and phrases graph homomorphism, forbidden induced subgraphs, precoloring extension

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.14

Related Version *Full Version*: https://arxiv.org/abs/2205.13270 [16]

Funding MD, MP, and PRz were supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062. KO was supported by Polish National Science Centre grant no. 2021/41/N/ST6/01507.

1 Introduction

Many computationally hard problems become tractable when restricted to instances with some special properties. In case of graph problems, typical families of such special instances come from forbidding certain substructures. For example, for a family $\mathcal{F}$ of graphs, a graph G is *$\mathcal{F}$-free* if it does not contain any graph from $\mathcal{F}$ as an induced subgraph. If $\mathcal{F}$ consists of a single graph F, then we write F-free instead of $\{F\}$-free. Classes defined by forbidden induced subgraphs are *hereditary*, i.e., closed under vertex deletion. Conversely, every hereditary class of graphs can be uniquely characterized by a (possibly infinite) set of minimal forbidden induced subgraphs.

Let us define two families of graphs that play a special role as forbidden induced subgraphs. For $t \geqslant 1$, by P_t we denote the path in t vertices. For $a, b, c \geqslant 1$, by $S_{a,b,c}$ we mean the graph consisting of three paths $P_{a+1}, P_{b+1}, P_{c+1}$ with one endvertex identified. Each graph $S_{a,b,c}$ is called a *subdivided claw*. The smallest subdivided claw, i.e., $S_{1,1,1}$ is the *claw*, and the graph $S_{2,1,1}$ is sometimes called the *fork* or the *chair*. Finally, let $\mathcal{S}$ denote the class of graphs in which every connected component is a path or a subdivided claw.

MIS and k-Coloring in F-free graphs. Two problems, whose complexity in hereditary graph classes attracts significant attention, are Max (Weighted) Independent Set (denoted by MIS) and k-Coloring. Let us briefly survey known results, focusing on F-free graphs for connected F. By an observation of Alekseev [2], MIS is NP-hard in F-free graphs, unless $F \in \mathcal{S}$. On the positive side, polynomial-time algorithms are known for some small graphs $F \in \mathcal{S}$. If $F = P_t$, then a polynomial-time algorithm for $t \leqslant 5$ was provided by Lokshtanov et al. [37], which was later extended to $t \leqslant 6$ by Grzesik et al. [28]. The case of $t = 7$ remains open and the general belief is that for every t the problem is polynomial-time solvable. Some evidence is given by the existence of quasipolynomial-time algorithms [25, 45].

The polynomial-time algorithm for MIS in claw-free graphs [41, 46] can be obtained by an extension of the augmenting path approach used for finding largest matchings [17]; note that a maximum matching is precisely a largest independent set in the line graph, and line graphs are in particular claw-free. There are also more modern approaches, based on certain decompositions of claw-free graphs [19, 42]. A polynomial-time algorithm for MIS in $S_{2,1,1}$-free graphs was first obtained by Alekseev [3] (only for the unweighted case), and later an arguably simpler algorithm was provided by Lozin and Milanič [39] (also for the weighted case). Again, the complexity of MIS in F-free graphs for larger subdivided claws F remains open, but all these cases are believed to be tractable. This belief is supported by the existence of a subexponential-time algorithm [12, 40], a QPTAS [12, 13, 40], or a polynomial-time algorithm in the bounded-degree case [1].

If it comes to k-Coloring, then it follows from known results that if F is not a forest of paths, then for every $k \geqslant 3$ the problem is NP-hard in F-free graphs [18, 26, 33, 36]. The complexity of k-Coloring in P_t-free graphs is quite well understood. For $t = 5$, the problem is polynomial-time solvable for every constant k [31]. If $k \geqslant 5$, then the problem is NP-hard already in P_6-free graphs [34]. The case of $k = 4$ is polynomial-time solvable for $t \leqslant 6$ [47] and NP-hard for $t \geqslant 7$ [34]. The case of $k = 3$ is much more elusive. We know a polynomial-time algorithm for P_7-free graphs [5] and the cases for all $t \geqslant 8$ are open. The general belief that they should be tractable is again supported by the existence of a quasipolynomial-time algorithm [45].

Let us mention two generalizations of k-Coloring. In the k-ColoringExt problem we are given a graph G with a subset of its vertices colored using k colors, and we ask whether this partial coloring can be extended to a proper k-coloring of G. In the even more

general List k-Coloring problem, each vertex of the instance graph G is equipped with a subset of $\{1,\ldots,k\}$ called *list*, and we ask whether there exists a proper k-coloring of G respecting all lists. Clearly any tractability result for a more general problem implies the same result for a less general one, and any hardness result for a less general problem implies the same hardness for more general variants. In almost all mentioned cases the algorithms for k-Coloring generalize to List k-Coloring. The only exception is the case $k = 4$ and $t = 6$: the polynomial time algorithm for 4-Coloring in P_6-free graphs can be generalized to 4-ColoringExt [47], but List 4-Coloring in this class is NP-hard [27]

Minimal obstructions. One of the ways of designing polynomial-time algorithms for k-Coloring is to check if the instance graph contains some (hopefully small) subgraph that is not k-colorable. This approach is formalized by the notion of *critical graphs*. A graph G is $(k+1)$-*vertex critical* if it is not k-colorable, but each of its induced subgraphs is. Such graphs can be thought of *minimal obstructions* to k-coloring: a graph G is k-colorable if and only if it does not contain any $(k+1)$-vertex-critical graph as an induced subgraph. Thus if for some hereditary class $\mathcal{G}$ of graphs, the number of $(k+1)$-vertex critical graphs is finite, we immediately obtain a polynomial-time algorithm for k-Coloring in graphs from $\mathcal{G}$. Indeed, it is sufficient to check if the instance graph contains any $(k+1)$-vertex-critical induced subgraph, which can be done in polynomial time by brute force. Such an algorithm, in addition to solving the instance, provides a *certificate* in case of a negative answer – a constant-size subgraph which does not admit a proper k-coloring. Thus the question whether for some class $\mathcal{G}$, the number of $(k+1)$-vertex critical graphs is finite, can be seen as a refined analysis of the polynomial cases of k-Coloring.

The finiteness of the families of $(k+1)$-vertex critical graphs in F-free graphs is fully understood. Recall that the only interesting (i.e., not known to be NP-hard) cases are for F being a forest of paths. Again focusing on connected F, i.e., k-Coloring of P_t-free graphs, we know that the families of minimal obstructions are finite for $t \leqslant 6$ and $k = 3$ [9], and for $t \leqslant 4$ and any k. The latter result follows from the fact that P_4-free graphs are perfect and thus the only obstruction for k-coloring is K_{k+1}. In all other cases there are constructions of infinite families of minimal obstructions [9, 32].

Graph homomorphisms in F-free graphs. A homomorphism from a graph G to a graph H is a mapping from $V(G)$ to $V(H)$ that preserves edges, i.e., the image of every edge of G is an edge of H. Note that if H is the complete graph on k vertices, then homomorphisms to $H = K_k$ are exactly proper k-colorings. For this reason homomorphisms to H are called H-*colorings*, and we will also refer to vertices of H as *colors*. In the H-Coloring problem the graph H is fixed and we need to decide whether an instance graph G admits a homomorphism to H. By the analogy to coloring, we also define more general variants, i.e., H-ColoringExt and List H-Coloring. In the former problem we ask whether a given partial mapping from vertices of an instance graph G to $V(H)$ can be extended to a homomorphism, and in the latter one each vertex of G is equipped with a list which is a subset of $V(H)$ and we look for a homomorphism respecting all lists.

The complexity of H-Coloring was settled by Hell and Nešetřil [30]: the problem is polynomial-time solvable if H is bipartite or has a vertex with a loop, and NP-hard otherwise. The dichotomy is also known for List H-Coloring [22–24]: this time the tractable cases are the so-called *bi-arc graphs*. The case of H-ColoringExt is more tricky. The classification follows from the celebrated proof of the CSP complexity dichotomy [7, 49], but a graph-theoretic description of polynomial cases is unknown.

We are very far from understanding the complexity of variants of H-Coloring in F-free graphs. Chudnovsky et al. [10] showed that List C_k-Coloring for $k \in \{5,7\} \cup [9,\infty)$ is polynomial-time solvable in P_9-free graphs. On the negative side, they showed that for every $k \geqslant 5$ the problem is NP-hard in F-free graphs, unless $F \in \mathcal{S}$. This negative result was later extended by Piecyk and Rzążewski [44] who showed that if H is not a bi-arc graph, then List H-Coloring is NP-hard and cannot be solved in subexponential time (assuming the ETH) in F-free graphs, unless $F \in \mathcal{S}$.

The case of forbidden path or subdivided claw was later investigated by Okrasa and Rzążewski [43]. They defined a class of *predacious graphs* and showed that if H is not predacious, then for every t, the List H-Coloring problem can be solved in quasipolynomial time in P_t-free graphs. On the other hand, for every predacious H there exists t for which List H-Coloring cannot be solved in subexponential time in P_t-free graphs unless the ETH fails. They also provided some partial results for the case of forbidden subdivided claws. Finally, Chudnovsky et al. [11] considered a generalization of List H-Coloring in P_5-free graphs and related classes.

The notion of vertex-critical graphs can be naturally translated to H-colorings. A graph G is a *minimal H-obstruction* if it is not H-colorable, but its every induced subgraph is. Minimal H-obstructions in restricted graph classes were studied by some authors, but the results are rather scattered [4, 8, 35].

Our motivation. Let us point a substantial difference between working with H-Coloring and working with List H-Coloring (with H-ColoringExt being somewhere between, but closer to H-Coloring). The List H-Coloring problem enjoys certain monotonicity: if H' is an induced subgraph of H, then every instance of List H'-Coloring can be seen as an instance of List H-Coloring, where no vertex from $V(H) - V(H')$ appears in any list. Thus any tractability result for List H-Coloring implies the analogous result for List H'-Coloring, while any hardness result for List H'-Coloring applies also to List H-Coloring. In particular, all hardness proofs for List H-Coloring follow the same pattern: first we identify a (possibly infinite) family $\mathcal{H}$ of "minimal hard cases" and then show hardness of List H'-Coloring for every $H' \in \mathcal{H}$. This implies that List H-Coloring is hard unless H is $\mathcal{H}$-free. The lists are also useful in the design of algorithms: for example if for some reason we decide that some vertex $v \in V(G)$ must be mapped to $x \in V(H)$, we can remove from the lists of neighbors of v all non-neighbors of x, and then delete v from the instance graph. This combines well with e.g. branching algorithms or divide-&-conquer algorithms based on the existence of separators.

In contrast, when coping with H-Coloring we need to think about the global structure of H. This makes working with this variant of the problem much more complicated. In particular, hardness proofs often employ certain algebraic tools [6, 30], which in turn do not combine well with the world of F-free graphs. However, note that the complexity dichotomy for H-Coloring *is still monotone* with respect to taking induced subgraphs of H: the minimal NP-hard cases are odd cycles.

An interesting example of non-monotonicity of H-Coloring was provided by Feder and Hell in an unpublished manuscript [21]. For an odd integer $k \geqslant 5$, let W_k denote the *k-wheel*, i.e., the graph obtained from the k-cycle by adding a universal vertex. Feder and Hell proved that W_5-Coloring is polynomial-time solvable in line graphs. This is quite surprising as W_5 contains a triangle and K_3-Coloring, i.e., 3-Coloring, is NP-hard in line graphs [33]. (Note that this implies that List W_5-Coloring is NP-hard in line graphs.) Feder and Hell also proved that for any odd $k \geqslant 7$, the W_k-Coloring problem is NP-hard in line graphs [21].

Our contribution. In this paper we study to which extent the result of Feder and Hell [21] can be generalized. We provide an algorithm and a number of lower bounds, each of a different kind. The main algorithmic contribution of our paper is the following theorem.

▶ **Theorem 1.** *W_5-ColoringExt can be solved in polynomial time in $S_{2,1,1}$-free graphs.*

Let us sketch the outline of the proof. Surprisingly, despite the fact that graph homomorphisms generalize colorings, our approach is much closer to the algorithms for MIS.

Consider any homomorphism φ from G to W_5 and let X be the set of vertices of G mapped by φ to the universal vertex of W_5. We notice that X is independent, and $G - X$ admits a homomorphism to C_5. This is exactly how we look at the problem: we aim to find an independent set X whose removal makes the graph C_5-colorable (and make sure that the precoloring of vertices is respected).

So let us focus on describing the structure of $G - X$, i.e., recognizing C_5-colorable graphs. Note that here we need to use the fact that our graph is $S_{2,1,1}$-free, as C_5-Coloring is NP-hard in general graphs. As a warm-up let us assume that our instance is claw-free and forget about precolored vertices. We observe that every C_5-colorable graph must be triangle-free, as there is no homomorphism from K_3 to C_5. But since $G - X$ is $\{S_{1,1,1}, K_3\}$-free, it must be of maximum degree at most 2, i.e., every component of $G - X$ is a path or a cycle with at least 4 vertices. It is straightforward to verify that such graphs always admit a homomorphism to C_5. Therefore in claw-free graphs, solving W_5-Coloring boils down to finding an independent set intersecting all triangles.

We extend this simple observation in two ways. First, we show that the same phenomenon occurs in $S_{2,1,1}$-free graphs: the only $S_{2,1,1}$-free minimal C_5-obstruction is the triangle. Second, we show that in the same way we can handle precolored vertices: the only no-instances of C_5-ColoringExt in $\{S_{2,1,1}, K_3\}$-free graphs can be easily recognized. Based on this result we show that W_5-ColoringExt in $S_{2,1,1}$-free graphs can be in polynomial time reduced to the Independent Triangle Transversal Extension (ITTE) problem on an induced subgraph of the original instance. Here, the "extension" means that some vertices of our instance can be prescribed to be in X or outside X. Thus from now on we focus on solving ITTE in $S_{2,1,1}$-free graphs. Note that we still need to use the fact that our instances are $S_{2,1,1}$-free, as ITTE is NP-hard in general graphs [20].

We start with the case that our instance graph G is claw-free. We use the result of Chudnovsky and Seymour [14] who show that each claw-free graph admits certain decomposition called a *strip structure*. Roughly speaking, this means that G "resembles" the line graph of some graph D: the vertices of G can be partitioned into sets $\eta(e)$ assigned to the edges e of D, such that (i) for each $e \in E(D)$ the set $\eta(e)$ induces a subgraph of G with a simple structure and (ii) the interactions between sets $\eta(e)$ and $\eta(f)$ for distinct edges e, f are well-defined. Due to property (i), for each edge e of D we can solve the problem in the subgraph of G induced by $\eta(e)$ in polynomial time. Then we can use property (ii) to combine these partial solutions into the final one by finding an appropriate matching in an auxiliary graph derived from D.

As the final step, we lift our algorithm for claw-free graphs to the class of $S_{2,1,1}$-free graphs. We use an observation of Lozin and Milanič [39]: they show that if G is $S_{2,1,1}$-free and *prime*, then every *prime* induced subgraph of the graph obtained from G by removing any vertex and its neighbors is claw-free (the exact definition of prime graphs can be found in Section 3.1). Using this observation we can reduce ITTE in $S_{2,1,1}$-free graphs to the same problem in claw-free graphs, which we already know how to solve in polynomial time.

Combining the reduction from W_5-ColoringExt in $S_{2,1,1}$-free graphs to ITTE in $S_{2,1,1,}$-free graphs and the polynomial-time algorithm for ITTE in $S_{2,1,1}$-free graphs we obtain Theorem 1. Let us point out that the frontier of the complexity of W_5-ColoringExt in $S_{a,b,c}$-free graphs is the same as for MIS: the minimal open cases are $S_{3,1,1}$ and $S_{2,2,1}$.

In the remainder of the paper we investigate several possible generalizations of Theorem 1 and show a number of negative results.

First, one can ask whether a simpler algorithm, at least for W_5-Coloring, can be obtained by showing that the family of minimal $S_{2,1,1}$-free W_5-obstructions is finite. In the following theorem we show that this is not the case.[1]

▶ **Theorem 2** (✡). *For all odd $k \geqslant 5$, there are infinitely many claw-free minimal W_k-obstructions.*

We remark that for every k, the List W_k-Coloring problem can be solved in polynomial time in P_5-free graphs [11]. We show that this result cannot be extended to P_t-free graphs for every fixed t: there exists t such that W_5-Coloring in P_t-free graphs is NP-hard and cannot be solved in subexponential time, unless the ETH fails. Note that this result in particular implies hardness in $S_{a,a,a}$-free graphs for $a = \lceil (t-1)/2 \rceil$. For W_5-ColoringExt we show that it is NP-hard and, assuming the ETH, there is no subexponential-time algorithm already in $S_{3,3,3}$-free graphs of maximum degree 5. This should be contrasted with the fact that for any a, b, c, the MIS problem in $S_{a,b,c}$-free graphs can be solved in subexponential time [12, 40], and even in polynomial time, if the instance is of bounded maximum degree [1]. Consequently, the complexity of W_5-Coloring in $S_{a,b,c}$-free graphs for large a, b, c differs from the complexity of MIS. Furthermore, we find our hardness results quite surprising, as typically problems that are hard in $S_{a,b,c}$-free graphs for some fixed a, b, c are already hard in claw-free graphs.

Finally, we consider the complexity of variants of W_k-Coloring in F-free graphs for other pairs (k, F). Our results are summarized in the following theorem.

▶ **Theorem 3** (✡). *Let F be a connected graph.*

1. *There exist a, b, c, t with the following property. If F is neither an induced subgraph of $S_{a,b,c}$ nor of P_t, then the W_5-Coloring problem is NP-hard in F-free graphs and cannot be solved in subexponential time, unless the ETH fails.*
2. *For every odd $k \geqslant 7$ there exists t with the following property. If F is not an induced subgraph of P_t, then the W_k-Coloring problem is NP-hard in F-free graphs and cannot be solved in subexponential time, unless the ETH fails.*

▶ Remark. The curious reader might wonder why we only consider k-wheels for odd $k \geqslant 5$. The 3-wheel is exactly K_4, and homomorphisms to K_4 are exactly proper 4-colorings. As mentioned above, both 4-Coloring and 4-ColoringExt are well studied in hereditary graph classes and behave substantially differently than W_k-Coloring for odd $k \geqslant 5$. On the other hand, if k is even, then the W_k-Coloring problem is equivalent to the 3-Coloring problem: a graph admits a homomorphism to W_k if and only if it is 3-colorable (this follows from the fact that K_3 is the *core* of W_k [29, Section 1.4]). Thus it only makes sense to consider variants of W_k-Coloring in F-free graphs, where F is a path or a forest of paths. However, as W_k is non-predacious, it follows from the result of Okrasa and Rzążewski [43] that then even List W_k-Coloring is quasipolynomial-time solvable in F-free graphs.

[1] The full proofs of the statements marked with (✡) can be found in the full version of the paper [16].

2 Preliminaries

For an integer k, by $[k]$ we denote the set $\{1, 2, \ldots, k\}$.

Let G be a graph, v be a vertex and X be a set of vertices. By $N_G(v)$ we denote the set of neighbors of v, and by $N_G[v]$ we denote the set $N_G(v) \cup \{v\}$. If G is clear from the context, we omit the subscript, and write, respectively, $N(v)$, and $N[v]$. By $G[X]$ we denote the subgraph induced by X, and by $G - X$ we denote $G[V(G) - X]$. By $\overline{G}$ we denote the complement of G.

We write $\varphi : G \to H$ to indicate that φ is a homomorphism from G to H. We also write $G \to H$ to indicate that some homomorphism from G to H exists.

For a fixed graph H, an instance of H-ColoringExt is a triple (G, U, φ), where G is a graph, U is a subset of $V(G)$, and φ is a function that maps vertices from U to elements of $V(H)$. We ask whether there exists a homomorphism $\psi : G \to H$ such that $\psi|_U = \varphi$.

For a k-wheel W_k, we will always denote the consecutive vertices of the induced k-cycle in W_k by $1, 2, 3, \ldots, k$, and the universal vertex by 0. The following observation is straightforward and will be used implicitly throughout the paper.

▶ **Observation 4.** *Let $k \geqslant 5$ be an odd integer. Let φ be a homomorphism from a graph G to W_k. Let X be the set of vertices of G mapped by φ to 0. Then the following properties are met: (i) G is 4-colorable, (ii) G is K_4-free, (iii) X is an independent set, and (iv) $G - X$ has a homomorphism to C_k, in particular it is 3-colorable and triangle-free.*

For a graph G, a function $\mathfrak{w} : E(G) \to \mathbb{N} \cup \{0\}$, and a set $E' \subseteq E(G)$, we define $\mathfrak{w}(E') := \sum_{e \in E'} \mathfrak{w}(e)$.

Consider a certain variant of the Maximum Weight Matching problem. An instance of Maximum Weight Matching* (MWM*) is a tuple $(G, U, \mathfrak{w}, k)$, where G is a graph, U is a subset of its vertices, $\mathfrak{w} : E(G) \to \mathbb{N} \cup \{0\}$ is an edge weight function, and k is an integer. We ask whether G has a matching M such that $\mathfrak{w}(M) \geqslant k$ and M covers all vertices from U. By a simple reduction to the Maximum Weight Matching problem we show that MWM* can be solved in polynomial time.

▶ **Lemma 5** (☆). *The MWM* problem can be solved in polynomial time.*

3 ITTE in $S_{2,1,1}$-free graphs

In this section we show that in $S_{2,1,1}$-free graphs the W_5-ColoringExt problem can be reduced to a variant of the problem of finding an independent set intersecting all triangles.

We denote the consecutive vertices of C_5 by $1, 2, 3, 4, 5$. We will refer to the vertices of C_5 as colors. We will say that two colors are *neighbors* if they are neighbors on the cycle C_5.

Let G be a graph, let $W \subseteq V(G)$, and let $\varphi : W \to V(C_5)$ be a coloring of vertices of W. We say that a pair of vertices $\{u, v\} \subseteq V(G)$ is *conflicted* if $u, v \in W$ and there is a u-v path P of length at most 3 such that $W \cap V(P) = \{u, v\}$ and $\varphi|_{\{u,v\}}$ cannot be extended to a homomorphism from P to C_5. Equivalently, a pair $\{u, v\} \subseteq W$ of vertices of G is conflicted in a coloring φ of G if and only if

- **(i)** $v \in N_G(u)$, and $\varphi(v)$ is non-adjacent to $\varphi(u)$ in C_5, or
- **(ii)** there exists a path u, w, v in G with $w \notin W$, and $\varphi(v)$ is adjacent to $\varphi(u)$, or
- **(iii)** there exists a path u, w_1, w_2, v in G with $w_1, w_2 \notin W$, and $\varphi(u) = \varphi(v)$.

We say that φ is *conflict-free*, if there is no pair of conflicted vertices in G.

Clearly, being triangle-free and conflict-free are necessary conditions to be a yes-instance of C_5-ColoringExt. It turns out that in $S_{2,1,1}$-free graphs they are also sufficient.

▶ **Lemma 6** (☆). *Let G be an $\{S_{2,1,1}, K_3\}$-free graph, let $W \subseteq V(G)$, and let $\varphi : W \to V(C_5)$. If φ is conflict-free, then φ can be extended to a homomorphism $\psi : G \to C_5$.*

The following auxiliary problem plays a crucial role in our algorithm.

Independent Triangle Transversal Extension (ITTE)	
Instance:	A graph G, sets $X', Y' \subseteq V(G)$, and $E' \subseteq E(G)$
Question:	Is there an independent set $X \subseteq V(G)$, such that (i) $X' \subseteq X$, (ii) for every $e \in E'$ it holds that $e \cap X \neq \emptyset$, (iii) $Y' \cap X = \emptyset$, (iv) $G - X$ is triangle-free?

We observe that if G contains a K_4, then at most one vertex of such a clique can belong to an independent set, and the graph induced by the remaining part is not triangle-free. Thus every yes-instance of ITTE must be K_4-free. We use this observation implicitly throughout the paper.

As the final result of this section we show that W_5-ColoringExt in $S_{2,1,1}$-free graphs can be reduced in polynomial time to ITTE in $S_{2,1,1}$-free graphs.

▶ **Theorem 7.** *The W_5-ColoringExt in $S_{2,1,1}$-free graphs can be reduced in polynomial time to the ITTE problem in $S_{2,1,1}$-free graphs.*

Proof. Let (G, U, φ) be an instance of W_5-ColoringExt. Note that we can assume that there are no vertices $u, v \in U$ such that $uv \in E(G)$ and $\varphi(u)\varphi(v) \notin E(W_5)$, otherwise we immediately report a no-instance (formally, we can return a trivial no-instance of ITTE, e.g., $(K_4, \emptyset, \emptyset, \emptyset)$). This can be verified in polynomial time.

We define the instance (G, X', Y', E') of ITTE as follows. We initialize X', Y' and E' as empty sets. We add to X' every vertex precolored with 0 and we add every vertex precolored with a vertex other than 0 to Y'. To construct E', consider each pair $u, v \in U$ of conflicted vertices with respect to $\varphi : Y' \to [5]$, and let P be a witness of u and v. By our first assumption, $|P| \geqslant 2$. If the consecutive vertices of P are u, w, v, then we add w to X'. If the consecutive vertices of P are u, w_1, w_2, v, then we add the edge w_1w_2 to E'. This completes the construction of the instance (G, X', Y', E') of ITTE. Clearly the reduction is done in polynomial time.

Let us verify that the instance (G, X', Y', E') of ITTE is equivalent to the instance (G, U, φ) of W_5-ColoringExt. First assume that there is a set $X \subseteq V(G)$ that is a solution to the instance (G, X', Y', E') of ITTE. Then $G - X$ is triangle-free. Suppose that there is a conflicted pair of vertices u, v in $G - X$ and let P be the path such that the precoloring of u, v cannot be extended to P. Observe that by the construction of E' at least one vertex of P must be in X, a contradiction. Hence, by calling Lemma 6 on $G - X$, we conclude that the precoloring of $G - X$ can be extended to all vertices of $G - X$ using only colors $1, 2, 3, 4, 5$, and then extended to the whole graph G by coloring every vertex of X with 0.

So now suppose that (G, U, φ) is a yes-instance of W_5-ColoringExt. Then there exists a W_5-coloring ψ of G that extends φ. Define $X := \psi^{-1}(0)$. Let us verify that X satisfies the desired properties. If follows from the definition that X is an independent set. Suppose that $G - X$ contains a triangle. Then $G - X \not\to C_5$, and thus the vertices of $G - X$ cannot be colored using only colors $1, 2, 3, 4, 5$, a contradiction. Consider a vertex $x \in X'$. Then either $\varphi(x) = 0$, or there is a path with consecutive vertices u, x, v with $u, v \in U$, such that φ cannot be extended to x using only colors $1, 2, 3, 4, 5$. Therefore in both cases $\psi(x) = 0$, so $x \in X$. Now consider $y \in Y'$. Then $\varphi(y) \neq 0$, so $y \notin \psi^{-1}(0) = X$. Finally, consider an edge $xy \in E'$. Then there is a path with consecutive vertices u, x, y, v with $u, v \in U$, so that φ cannot be extended to x, y using only colors $1, 2, 3, 4, 5$. We conclude that one of x, y is mapped by ψ to 0, and thus one of x, y is in X. That completes the proof. ◀

3.1 ITTE: basic toolbox

In this section we prove that ITTE behaves well under standard graph decompositions: modular decomposition, clique-cutset decomposition, and tree decomposition.

Modular decomposition and prime graphs. Let G be a graph and let $M \subsetneq V(G)$. We say that M is a *module* of G if for every vertex $v \in V(G) - M$ either v is adjacent to any vertex of M or is non-adjacent to every vertex of M. We say that a module M is *non-trivial* if $|M| > 1$, otherwise M is *trivial*. We say that G is *prime* if every module of G is trivial.

In the next lemma we show that ITTE combines well with modular decompositions.

▶ **Lemma 8** (☆). *Let $\mathcal{X}$ be a hereditary class of graphs and let $\mathcal{X}^*$ be the class of all induced subgraphs of the graphs in $\mathcal{X}$ that are either prime or cliques. If ITTE can be solved in polynomial time on $\mathcal{X}^*$, then it can be solved in polynomial time on $\mathcal{X}$.*

Clique cutsets and atoms. A (possibly empty) set $C \subseteq V(G)$ is a *cutset* in G, if $V(G) - C$ is disconnected. We say that C is a *clique cutset* if C is a cutset and a clique. We say that G is an *atom* if it does not contain clique cutsets. Note that, in particular, every atom is connected. A triple (A, C, B) is a *clique cutset partition* if it is a partition of $V(G)$ such that C is a clique cutset, there is no edge between A and B, and $G[A \cup C]$ is an atom.

We show that we can reduce ITTE to instances (G, X', Y', E') such that G is an atom. The algorithm is a standard recursion that exploits the existence of clique cutsets [48].

▶ **Lemma 9** (☆). *Let $\mathcal{X}$ be a family of graphs, and let $\mathcal{X}^a$ be the family of all atoms in $\mathcal{X}$. If ITTE can be solved in polynomial time in $\mathcal{X}^a$, then ITTE can be solved in polynomial time in $\mathcal{X}$.*

Bounded-treewidth graphs. *Monadic Second-Order Logic* (MSO_2) over graphs consists of formulas with vertex variables, edge variables, vertex set variables, and edge set variables, quantifiers, and standard logic operators. We also have a predicate $\mathsf{inc}(v, e)$, indicating that the vertex v belongs to the edge e.

For a graph G, let $\mathrm{tw}(G)$ denote the treewidth of G. The classic result of Courcelle [15] asserts that problems that can be expressed in MSO_2 can be efficiently solved on graphs of bounded treewidth. As ITTE can be expressed in such a way, we obtain the following.

▶ **Corollary 10** (☆). *ITTE can be solved in polynomial time on bounded-treewidth graphs.*

4 Solving ITTE in claw-free graphs

Let G be a connected graph. A *strip structure of* G is a pair (D, η) that consists of a simple graph D, a set $\eta(xy) \subseteq V(G)$ for every $xy \in E(D)$, and its non-empty subsets $\eta(xy, x), \eta(xy, y) \subseteq \eta(xy)$, satisfying the following conditions:

(S1) $|E(D)| \geqslant 3$, and there are no vertices of degree 2 in D,

(S2) the set $\{\eta(e) : e \in E(D)\}$ forms a partition of $V(G)$,

(S3) if $u \in \eta(e), v \in \eta(f)$, for some $e, f \in E(D)$, then $uv \in E(G)$ if and only if there exists $x \in V(D)$ such that e and f are incident to x, $u \in \eta(e, x)$ and $w \in \eta(f, x)$,

(S4) for every $x \in V(D)$, the set $\bigcup_{y \in N(x)} \eta(xy, x)$ induces a clique in G.

Chudnovsky and Seymour [14] proved that every claw-free graph G is either very simple or admits a strip structure where the subgraphs induced by vertices assigned to a single edge have "simple" structure. If G is additionally of bounded maximum degree, then this result becomes particularly usuful. The following corollary of the result of Chudnovsky and Seymour comes from the paper of Abrishami et al. [1, Corollary 3.5].

▶ **Theorem 11** (Chudnovsky, Seymour [14]). *If G is a connected claw-free graph with maximum degree at most Δ, then either $\mathrm{tw}(G) \leqslant 4\Delta + 3$, or G admits a strip structure (D, η) such that for every $e \in E(D)$, we have $\mathrm{tw}(G[\eta(e)]) \leqslant 4\Delta + 4$. Moreover, (D, η) can be found in time polynomial in $|V(G)|$.*

The main result of this section is the following theorem.

▶ **Theorem 12.** *ITTE can be solved in polynomial time in claw-free graphs.*

Proof. Let (G, X', Y', E') be an instance of ITTE such that G is claw-free. By Lemma 9, we can assume that G is an atom, and, in particular, G is connected.

Recall that we can safely assume that G is K_4-free, as otherwise we deal with a no-instance. This implies that $\Delta(G) \leqslant 5$. Indeed, if G has a vertex v of degree at least six, then v either has three pairwise non-adjacent neighbors (and hence a claw), or three pairwise adjacent neighbors (and hence a K_4).

Since G is claw-free and $\Delta(G) \leqslant 5$, by Theorem 11, either (i) $\mathrm{tw}(G) \leqslant 23$ or (ii) there exists a strip structure (D, η) such that for every $e \in E(D)$ we have $\mathrm{tw}(G[\eta(e)]) \leqslant 24$. By Corollary 10, if (i) holds, it can be checked in polynomial time if G is a yes-instance. Therefore, we will assume that (ii) holds. By Theorem 11 the strip structure (D, η) can be computed in polynomial time. As G is connected, so is D.

Recall that by the definition of a strip structure, if $y_1, \ldots, y_d$ are neighbors of some $x \in V(D)$, the set $\bigcup_{i \in [d]} \eta(xy_i, x)$ induces a clique in G. Hence, as G is K_4-free, we have $\Delta(D) \leqslant 3$, which implies that the vertices of D are either of degree 1 or 3. Moreover, if x is of degree 3, let $N(x) = \{y_1, y_2, y_3\}$ and note that since G is K_4-free, $|\eta(xy_1, x)| = |\eta(xy_2, x)| = |\eta(xy_3, x)| = 1$. In this case, we denote the single member of $\eta(xy, x)$ by $v_{x,y}$.

On the other hand, if D contains a vertex y of degree 1, two things can happen: either D is isomorphic to K_2, or y is adjacent to a vertex x that is of degree 3 in D. In the first case, if we denote by e the edge of D, we have $\mathrm{tw}(G) = \mathrm{tw}(G[\eta(e)]) \leqslant 24$, so again, we can solve the problem in polynomial time by Corollary 10. In the second case, let y_1, y_2 be the other (than y) neighbors of x in D. We observe that $\{v_{x,y_1}, v_{x,y_2}\}$ or $\{v_{x,y}\}$ is a clique cutset in G, a contradiction with G being an atom. Hence, we can assume that D is 3-regular.

For $x \in V(D)$ and $y_1, y_2, y_3 \in N_D(x)$, let $E(x) := \{v_{x,y_1}v_{x,y_2}, v_{x,y_2}v_{x,y_3}, v_{x,y_3}v_{x,y_1}\}$. Recall that by the definition of a strip structure, $G[E(x)]$ is a triangle. The following reduction rule is straightforward.

▶ **Reduction Rule 12.1.** *If $E(x) \subseteq E'$, then (G, X', Y', E') is a no-instance. If $v_{x,y_1}v_{x,y_2}, v_{x,y_2}v_{x,y_3} \in E'$, but $v_{x,y_3}v_{x,y_1} \notin E'$, we can safely add v_{x,y_2} to X' and remove $v_{x,y_1}v_{x,y_2}, v_{x,y_2}v_{x,y_3}$ from E'. Hence, for every $x \in V(D)$ it holds that $|E(x) \cap E'| \leqslant 1$.*

Consider an edge xy of D. Let $G_{xy} := G[\eta(xy)]$, and let $I_{xy} = \{v_{x,y}, v_{y,x}\} - Y'$. Let $\mathcal{A}(xy)$ be the (possibly empty) set of these sets A that satisfy $X' \cap I_{xy} \subseteq A \subseteq I_{xy}$ and

$$\mathcal{I}_A = (G_{xy}, (X' \cap V(G_{xy})) \cup A, (Y' \cap V(G_{xy})) \cup (I_{xy} - A), E' \cap E(G_{xy}))$$

is a yes-instance of ITTE. Observe that if there exists a solution X of ITTE for (G, X', Y', E'), then $X \cap \eta(xy)$ is a solution to $\mathcal{I}_{A'}$ for $A' = \{v_{x,y}, v_{y,x}\} \cap X$, and in this case $A' \in \mathcal{A}(xy)$. We obtain the following.

▶ **Reduction Rule 12.2.** *If $\mathcal{A}(xy) = \emptyset$ for some $xy \in E(D)$, then (G, X', Y', E') is a no-instance.*

Our aim is to reduce the ITTE problem to the MWM* problem. Let $\mathcal{E} := \{xy \in E(D) : \emptyset \notin \mathcal{A}(xy)\}$. We construct an instance $(D', U, \mathfrak{w}, |\mathcal{E}|)$ of MWM* as follows. To obtain D', we take a copy of the set $V(D)$, and for every $xy_1 \in E(D)$ we introduce an additional vertex t_{xy}. Then, for every $xy \in E(D)$, we do the following (below we always use the notation $N_D(x) = \{y, y', y''\}$).

(1) if $\{v_{x,y}, v_{y,x}\} \in \mathcal{A}(xy)$, we add xy to $E(D')$,
(2) if $\{v_{x,y}\} \in \mathcal{A}(xy)$, we add xt_{xy} to $E(D')$,
(3) if $\{v_{y,x}\} \in \mathcal{A}(xy)$, we add yt_{xy} to $E(D')$.
(4) if $\{v_{x,y'}v_{x,y''}\} = E' \cap E(x)$, we remove xy and xt_{xy} from $E(D')$ (if they were introduced).

Let $U = V(D) \subseteq V(D')$, i.e., the set of original vertices of D. For $xy \in E(D)$, we define $T(xy) := \{xy, xt_{xy}, yt_{xy}\} \cap E(D')$. We set $\mathfrak{w} : E(D') \to \{0, 1\}$ to be $\mathfrak{w}(uv) = 1$ if $uv \in T(xy)$ for some $xy \in \mathcal{E}$, and $\mathfrak{w}(uv) = 0$ otherwise. This concludes the construction of $(D', U, \mathfrak{w}, |\mathcal{E}|)$.

By the definition of the strip structure, each edge $uv \in V(G)$ is either contained in $E(G_{xy})$, for some $xy \in E(D)$, or $u = v_{x,y'}$ and $v = v_{x,y''}$ for some $x \in U$. In the first case, we say that uv is of *type I for* xy, in the latter – that uv is of *type II for* $\{xy', xy''\}$. Similarly, each triangle $uvt \in V(G)$ is is either contained in $E(G_{xy})$, for some $xy \in E(D)$ (so it is of *type I for* xy), or $u = v_{x,y}$, $v = v_{x,y'}$ and $t = v_{x,y''}$ for some $x \in U$ (so it is of *type II for* x).

▷ Claim 13 (☆). (G, X', Y', E') is a yes-instance of ITTE if and only if $(D', U, \mathfrak{w}, |\mathcal{E}|)$ is a yes-instance of MWM*.

Sketch of proof of Claim. There is a correspondence between a matching M which is a solution to $(D', U, \mathfrak{w}, |\mathcal{E}|)$ and a set X which is a solution to (G, X', Y', E'). Let us present the intuition.

Consider $xy \in E(D)$ and note that $X \cap \eta(xy)$ is a solution of the instance induced by $\eta(xy)$. Note that the only way how such a partial solution interacts with the rest of the graph is by including (or not) vertices $v_{x,y}$ and $v_{y,x}$ to X. Each of the four possibilities is reflected by the way how M intersects $T(xy)$ (see the definition of $E(D')$). The requirement that M is of weight at least $|\mathcal{E}|$ means that for each $xy \in E(D)$ such that $\emptyset \notin \mathcal{A}(xy)$ we *have* to choose some edge from $T(x, y)$ to M. This is crucial as for these edges there are no solutions of ITTE restricted to $\eta(xy)$ containing neither $v_{x,y}$ nor $v_{y,x}$.

Finally, the set U is used to ensure that the partial solutions chosen for distinct edges of D are compatible with each other. It enforces that for each $x \in V(D)$, the set X intersects the triangle $\bigcup_{xy \in E(D)} \eta(xy, x)$. ◁

Therefore, for an instance $(G, X', Y', |\mathcal{E}|)$ of ITTE, we create an equivalent (by Claim 13) instance $(D', U, \mathfrak{w}, |\mathcal{E}|)$ of MWM*. Then we solve $(D', U, \mathfrak{w}, |\mathcal{E}|)$.

It remains to estimate the running time. We check in polynomial time whether G contains a K_4. As for each $xy \in E(D)$, the set $\eta(xy)$ is non-empty, $|E(D)| \leqslant n$. For every $xy \in E(D)$ we compute $\mathcal{A}(xy)$, which requires solving at most four instances $\mathcal{I}_\emptyset, \mathcal{I}_{\{v_{x,y}\}}, \mathcal{I}_{\{v_{y,x}\}}, \mathcal{I}_{\{v_{x,y}, v_{y,x}\}}$. Each of them consists of a graph G_{xy} with $\mathrm{tw}(G_{xy}) \leqslant 24$ (by Theorem 11), hence this can also be done in polynomial time. Finally, the instance $(D', U, \mathfrak{w}, |\mathcal{E}_N|)$ of MWM* can be constructed and solved in polynomial time by Lemma 5. ◀

5 Solving ITTE in $S_{2,1,1}$-free graphs

In this section we generalize the algorithm from Theorem 12 to the class of $S_{2,1,1}$-free graphs. The main combinatorial tool is the following theorem used to solve MIS in this class [38].

▶ **Theorem 14** (Lozin, Milanič [38, Theorem 4.1]). *Let G be an $S_{2,1,1}$-free prime graph and let $v \in V(G)$. Let G' be an induced prime subgraph of $G - N[v]$. Then G' is claw-free.*

Equipped with Theorem 14, we can present our algorithm.

▶ **Theorem 15.** *ITTE can be solved in polynomial time in $S_{2,1,1}$-free graphs.*

Proof. Let (G, X', Y', E') be an instance of ITTE such that G is $S_{2,1,1}$-free. By calling Lemma 8 for the class of $S_{2,1,1}$-free graphs, it is enough to consider the case that G is either a prime $S_{2,1,1}$-free graph or a clique. If $Y' = V(G)$, then we can verify in polynomial time if $X = \emptyset$ is a solution. So since now we assume that Y' is a proper subset of $V(G)$. Observe that if (G, X', Y', E') is a yes-instance, then there exists a solution X that is non-empty. Indeed, for a solution $X = \emptyset$, we can safely add a vertex $v \in V(G) - Y'$.

For every $v \notin Y'$ we define the instance $\mathcal{I}_v := (G, X' \cup \{v\}, Y', E')$. Now (G, X', Y', E') is a yes-instance if and only if there exists $v \notin Y'$ for which $\mathcal{I}_v$ is a yes-instance. Consider one fixed $v \notin Y'$. Clearly, for any solution X, we have that $X \cap N(v) = \emptyset$, so if there is a vertex in $N(v) \cap X'$ or there is an edge in E' with both endpoints in $N(v)$, we report a no-instance.

Define $G_v := G - N[v]$. We initialize $X'' := X'$ and $Y'' := Y' - N[v]$, and let E'' contain these edges from E' that have both endpoints in $V(G_v)$. Consider a triangle xyz. If $x \in N(v)$ and $y, z \in V(G_v)$ we add yz to E''. If $x, y \in N(v)$ and $z \in V(G_v)$ we add z to X''. Finally, for every edge $uw \in E'$ with $u \in N(v)$ and $w \in V(G_v)$, we add w to X''. Hence, it is enough to focus on the instance (G_v, X'', Y'', E''), as it is clearly equivalent to the instance $\mathcal{I}_v$. Note that this reduction can be performed in polynomial time.

We call again Lemma 8, now for the class $\mathcal{Y} := \{G_w \mid G \in \mathcal{X}, w \in V(G)\}$, so it is enough to solve the problem for the class $\mathcal{Y}^*$ of all induced subgraphs of the graphs in $\mathcal{Y}$ that are either prime or cliques. Clearly, every clique is claw-free. Together with Theorem 14, it implies that every graph in $\mathcal{Y}^*$ is claw-free. By Theorem 12, ITTE can be solved in polynomial time on $\mathcal{Y}^*$, and therefore it can be solved in polynomial time on $\mathcal{Y}$. Since $G_v \in \mathcal{Y}$, the instance (G_v, X'', Y'', E'') can be solved in polynomial time. ◀

Combining Theorem 7 with Theorem 15 we obtain our main algorithmic result.

▶ **Theorem 1.** *W_5-ColoringExt can be solved in polynomial time in $S_{2,1,1}$-free graphs.*

6 W_5-ColoringExt in $S_{3,3,3}$-free graphs is hard

In this section we prove the following hardness result.

▶ **Theorem 16** (☆). *The W_5-ColoringExt problem is NP-hard in $S_{3,3,3}$-free graphs of maximum degree at most 5. Furthermore, it cannot be solved in time $2^{o(n)}$, where n is the number of vertices of the input graph, unless the ETH fails.*

Sketch of proof. Let G be an instance of 3-Coloring such that G is a claw-free graph of maximum degree at most 4 [33]. Note that we can assume that the minimum degree of G is at least 3. Since G is claw-free, this means that every vertex of G belongs to a triangle.

We construct an instance $(\widetilde{G}, U, \varphi)$ of W_5-ColoringExt. We initialize $V(\widetilde{G}) := V(G)$, $E(\widetilde{G}) := E(G)$, and $U := \emptyset$. For each $v \in V(G)$ we proceed as follows. We add to $V(\widetilde{G})$ vertices x_v, y_v, z_v and to $E(\widetilde{G})$ we add $x_v y_v$, $y_v z_v$, and $y_v v$. Moreover we add x_v, z_v to U and set $\varphi(x_v) := 0$ and $\varphi(z_v) := 4$.

Note that the gadget attached to every vertex v of G simulates imposing the list $\{0, 1, 2, 4\}$. However, recall that every vertex of G belongs to a triangle, so it will never receive color 4. Thus $(\widetilde{G}, U, \varphi)$ is a yes-instance of W_5-ColoringExt if and only if G is 3-colorable.

It is straighforward to verify that $\widetilde{G}$ is $S_{3,3,3}$-free: the central vertex of a hypothetical $S_{3,3,3}$ must be a vertex of G, and at least one leg of $S_{3,3,3}$ must be fully contained in the gadget attached to v. Thus this path cannot have three vertices. ◀

7 Conclusion

Let us conclude the paper with pointing out some open questions and directions for future research.

Variants of W_5-Coloring in $S_{a,b,c}$-free graphs. Recall that W_5-ColoringExt is polynomial-time solvable in $S_{2,1,1}$,-free graphs but NP-hard in $S_{3,3,3}$-free graphs. We point out that this leaves an infinite family of open cases, and the minimal ones are $S_{2,2,1}$ and $S_{3,3,3}$.

Initial research shows that Lemma 6 can probably be extended to $S_{2,2,1}$-free graphs, at least without precolored vertices. Thus there is hope to solve W_5-Coloring by a reduction to ITTE. However, an analogue of Lemma 6 does not hold for $S_{2,2,2}$-free and for $S_{3,1,1}$,-free graphs; see Figure 1.

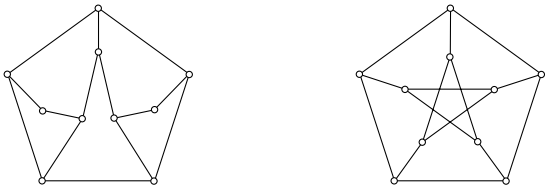


Figure 1 An $\{S_{2,2,2}, K_3\}$-free (left) and an $\{S_{3,1,1}, K_3\}$-free (right) graph that are not C_5-colorable.

Of course this does not mean that some other approach cannot work for W_5-Coloring. However, as we show in Theorem 3, the problem becomes hard if we exclude some long subdivided claw. This leads to a natural question about the boundary between easy and hard cases.

▶ **Question 1.** *For which a, b, c are W_5-Coloring and W_5-ColoringExt polynomial-time solvable in $S_{a,b,c}$-free graphs?*

Minimal obstructions. Recall that by Theorem 2 and Theorem 3, the only connected graphs F for which we can hope for a finite family of F-free minimal W_k-obstructions are paths. This leads to the following question.

▶ **Question 2.** *For which k and t is the family of P_t-free minimal W_k-obstructions finite?*

Let us point out that for all k, the number of P_4-free minimal W_k-obstructions is finite. Indeed, P_4-free graphs are perfect. Thus if G is P_4-free and does not contain K_4, then G is 3-colorable (and thus it admits a homomorphism to W_k). On the other hand, K_4 is a minimal graph that does not admit a homomorphism to W_k. Concluding, K_4 is the only P_4-free (even P_3-free) W_k-obstruction.

On the negative side, recall that there exists an infinite family of P_7-free 4-vertex-critical graphs [9]. An inspection of this family shows that these graphs are minimal W_k-obstructions for every odd $k \geqslant 5$. Note that for each such graph G, the fact that an induced subgraph G' of G has a proper 3-coloring implies that G' has a homomorphism to W_k. However, it still needs to be verified that G itself does not admit a homomorphism to W_k.

Thus the open cases in Question 2 are $t = 5$ and $t = 6$.

References

1 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, and Paweł Rzążewski. Polynomial-time algorithm for maximum independent set in bounded-degree graphs with no long induced claws. In Niv Buchbinder Joseph (Seffi) Naor, editor, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference, January 9-12, 2022*, pages 1448–1470. SIAM, 2022. doi:10.1137/1.9781611977073.61.

2 V.E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982. (in Russian).

3 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1–3):3–16, 2004. doi:10.1016/S0166-218X(02)00290-1.

4 Laurent Beaudou, Florent Foucaud, and Reza Naserasr. Smallest C_{2l+1}-critical graphs of odd-girth $2k+1$. In Manoj Changat and Sandip Das, editors, *Algorithms and Discrete Applied Mathematics - 6th International Conference, CALDAM 2020, Hyderabad, India, February 13-15, 2020, Proceedings*, volume 12016 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2020. doi:10.1007/978-3-030-39219-2_16.

5 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Comb.*, 38(4):779–801, 2018. doi:10.1007/s00493-017-3553-8.

6 Andrei A. Bulatov. H-Coloring dichotomy revisited. *Theor. Comput. Sci.*, 349(1):31–39, 2005. doi:10.1016/j.tcs.2005.09.028.

7 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

8 Paul A. Catlin. Graph homomorphisms into the five-cycle. *J. Comb. Theory, Ser. B*, 45(2):199–211, 1988. doi:10.1016/0095-8956(88)90069-X.

9 Maria Chudnovsky, Jan Goedgebeur, Oliver Schaudt, and Mingxian Zhong. Obstructions for three-coloring graphs without induced paths on six vertices. *J. Comb. Theory, Ser. B*, 140:45–83, 2020. doi:10.1016/j.jctb.2019.04.006.

10 Maria Chudnovsky, Shenwei Huang, Pawel Rzążewski, Sophie Spirkl, and Mingxian Zhong. Complexity of C_k-coloring in hereditary classes of graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 31:1–31:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.31.

11 Maria Chudnovsky, Jason King, MichałPilipczuk, PawełRzążewski, and Sophie Spirkl. Finding large H-colorable subgraphs in hereditary graph classes. *SIAM J. Discret. Math.*, 35(4):2357–2386, 2021. doi:10.1137/20M1367660.

12 Maria Chudnovsky, Marcin Pilipczuk, Michal Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set Problem in H-free graphs. *CoRR*, abs/1907.04585, 2019. arXiv:1907.04585.

13 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set problem in H-free graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2260–2278. SIAM, 2020.

14 Maria Chudnovsky and Paul D. Seymour. Claw-free graphs. v. global structure. *J. Comb. Theory, Ser. B*, 98(6):1373–1410, 2008. doi:10.1016/j.jctb.2008.03.002.

15 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

16 Michał Dębski, Zbigniew Lonc, Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Computing homomorphisms in hereditary graph classes: the peculiar case of the 5-wheel and graphs with no long claws. *CoRR*, abs/2205.13270, 2022. doi:10.48550/arXiv.2205.13270.

17 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

18 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Comb. Probab. Comput.*, 7(4):375–386, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=46667.

19 Yuri Faenza, Gianpaolo Oriolo, and Gautier Stauffer. Solving the weighted stable set problem in claw-free graphs via decomposition. *J. ACM*, 61(4):20:1–20:41, 2014. doi:10.1145/2629600.

20 Alastair Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *Electron. J. Comb.*, 11(1), 2004. URL: http://www.combinatorics.org/Volume_11/Abstracts/v11i1r46.html, doi:10.37236/1799.

21 Tomás Feder and Pavol Hell. Edge list homomorphisms. Unpublished manuscript, available at http://theory.stanford.edu/~tomas/edgelist.ps.

22 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.

23 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.

24 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

25 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

26 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of coloring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84(4):331–363, 2017. doi:10.1002/jgt.22028.

27 Petr A. Golovach, Daniël Paulusma, and Jian Song. Closing complexity gaps for coloring problems on H-free graphs. *Inf. Comput.*, 237:204–214, 2014. doi:10.1016/j.ic.2014.02.004.

28 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6-free graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1257–1271. SIAM, 2019. doi:10.1137/1.9781611975482.77.

29 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*. Oxford University Press, 2004.

30 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

31 Chính T. Hoàng, Marcin Kamiński, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57(1):74–81, 2010. doi:10.1007/s00453-008-9197-8.

32 Chính T. Hoàng, Brian Moore, Daniel Recoskie, Joe Sawada, and Martin Vatshelle. Constructions of k-critical P_5-free graphs. *Discret. Appl. Math.*, 182:91–98, 2015. doi:10.1016/j.dam.2014.06.007.

33 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10:718–720, 1981.

34 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *Eur. J. Comb.*, 51:336–346, 2016. doi:10.1016/j.ejc.2015.06.005.

35 Marcin Kamiński and Anna Pstrucha. Certifying coloring algorithms for graphs without long induced paths. *Discret. Appl. Math.*, 261:258–267, 2019. doi:10.1016/j.dam.2018.09.031.

36 Daniel Leven and Zvi Galil. NP-completeness of finding the chromatic index of regular graphs. *J. Algorithms*, 4(1):35–44, 1983. doi:10.1016/0196-6774(83)90032-9.

37 Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent Set in P_5-free graphs in polynomial time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

38 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 26–30. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109561.

39 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.

40 Konrad Majewski, Tomás Masařík, Jana Novotná, Karolina Okrasa, Marcin Pilipczuk, Paweł Rzążewski, and Marek Sokołowski. Max Weight Independent Set in graphs with no long claws: An analog of the Gyárfás' path argument. *CoRR*, abs/2203.04836, 2022. accepted to ICALP 2022. doi:10.48550/arXiv.2203.04836.

41 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

42 Paolo Nobili and Antonio Sassano. An $O(n^2 \log n)$ algorithm for the weighted stable set problem in claw-free graphs. *Math. Program.*, 186(1):409–437, 2021. doi:10.1007/s10107-019-01461-5.

43 Karolina Okrasa and PawełRzążewski. Complexity of the list homomorphism problem in hereditary graph classes. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 54:1–54:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.54.

44 Marta Piecyk and Paweł Rzążewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.56.

45 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for Independent Set in P_t-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

46 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile. *Discrete Mathematics*, 29(1):53–76, 1980. (in French).

47 Sophie Spirkl, Maria Chudnovsky, and Mingxian Zhong. Four-coloring P_6-free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1239–1256. SIAM, 2019. doi:10.1137/1.9781611975482.76.

48 Robert Endre Tarjan. Decomposition by clique separators. *Discret. Math.*, 55(2):221–232, 1985. doi:10.1016/0012-365X(85)90051-2.

49 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

Computing Palindromes on a Trie in Linear Time

Takuya Mieno
Department of Computer and Network Engineering, University of Electro-Communications, Tokyo, Japan

Mitsuru Funakoshi
Department of Informatics, Kyushu University, Fukuoka, Japan
Japan Society for the Promotion of Science, Tokyo, Japan

Shunsuke Inenaga[1]
Department of Informatics, Kyushu University, Fukuoka, Japan
PRESTO, Japan Science and Technology Agency, Tokyo, Japan

Abstract

A trie $\mathcal{T}$ is a rooted tree such that each edge is labeled by a single character from the alphabet, and the labels of out-going edges from the same node are mutually distinct. Given a trie $\mathcal{T}$ with n edges, we show how to compute all distinct palindromes and all maximal palindromes on $\mathcal{T}$ in $O(n)$ time, in the case of integer alphabets of size polynomial in n. This improves the state-of-the-art $O(n \log h)$-time algorithms by Funakoshi et al. [PSC 2019], where h is the height of $\mathcal{T}$. Using our new algorithms, the eertree with suffix links for a given trie $\mathcal{T}$ can readily be obtained in $O(n)$ time. Further, our trie-based $O(n)$-space data structure allows us to report all distinct palindromes and maximal palindromes in a query string represented in the trie $\mathcal{T}$, in output optimal time. This is an improvement over an existing (naïve) solution that precomputes and stores all distinct palindromes and maximal palindromes for each and every string in the trie $\mathcal{T}$ separately, using a total $O(n^2)$ preprocessing time and space, and reports them in output optimal time upon query.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases palindromes, suffix trees, tries, labeled trees, eertrees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.15

Funding *Mitsuru Funakoshi*: JSPS KAKENHI Grant Number JP20J21147.
Shunsuke Inenaga: JST PRESTO Grant Number JPMJPR1922, JSPS KAKENHI Grant Number JP22H03551.

Acknowledgements We would like to thank anonymous referees for their helpful comments. We also thank Takuya Matsumoto for discussions.

1 Introduction

A string p is called a *palindrome* if p reads the same forward and backward, namely, $p = p^R$ where p^R denotes the reversed string of p. Finding palindromes in a given string is a classical problem in Theoretical Computer Science, with motivations and possible applications in Bioinformatics [11, 17]. A substring palindrome $p = S[i..j]$ in a string S is called a *maximal palindrome* if $S[i-1..j+1]$ is not a palindrome, $i = 1$, or $j = |S|$. Since any substring palindrome in S shares the same center with a unique maximal palindrome, one can essentially obtain all substring palindromes in string S by computing its maximal palindromes.

There are two well-known algorithms that compute all maximal palindromes in a given string S of length m. The algorithm of Manacher [14] finds all $2m-1$ maximal palindromes in S in $O(m)$ time and space. His algorithm scans an input string from left to right, and

[1] Corresponding author

works on a general (unordered) alphabet. The algorithm of Gusfield [11] consists in two steps: In the preprocessing step, it builds the suffix tree [18] of the concatenated string $S\$S^R\#$, where \$ and # are unique symbols not occurring inside S, and then enhances the suffix tree with the lowest common ancestor (LCA) data structure [1] that allows for LCA queries between two nodes in $O(1)$ time after linear-time preprocessing. Then, the algorithm finds all maximal palindromes by applying $2m-1$ *outward* longest common extension (outward LCE) queries in S via the LCA data structure on the suffix tree. Overall, the Gusfield algorithm works in $O(m)$ time and space in the case of *linearly-sortable alphabets*, including constant-size alphabets and integer alphabets of size polynomial in m.

While maximal palindromes tell us all *occurrences* of palindromes in a string S, the other important direction of research is the *vocabularies* of palindromes in S. The latter is called the *distinct palindromes* problem, where the task is to compute the set of all distinct palindromes in S. It is known that any string of length m can contain at most $m+1$ distinct palindromes (including the empty string) [5]. The algorithm of Groult et al. [10] finds all distinct palindromes in $O(m)$ time and space for linearly-sortable alphabets.

We consider extended versions of the above problems, in which the input is a *trie* (i.e. an edge-labeled rooted tree). Tries are a natural generalization of strings at least in two meanings: A trie can be seen as a generalized string with branches; A trie is a compact representation of a set of strings. Funakoshi et al. [7] showed that, for any trie $\mathcal{T}$ with n edges and l leaves, the number of maximal palindromes in $\mathcal{T}$ is exactly $2n-l$ and the number of distinct palindromes in $\mathcal{T}$ is at most $n+1$. In addition, they showed how to find all maximal palindromes in a given trie $\mathcal{T}$ in $O(n \log h)$ time with $O(h)$ space, where h is the height of $\mathcal{T}$. This algorithm is based on the periodicity of palindromes, and works on general alphabets. They also showed how to find all distinct palindromes in a given trie in $O(n \log h)$ time with $O(n)$ space, in the case of linearly-sortable alphabets.

In this paper, we propose the first $O(n)$-time algorithms for computing the maximal palindromes and distinct palindromes in a given trie $\mathcal{T}$ with n edges, in the case of integer alphabets of polynomial size in n. Our algorithm makes heavy use of the suffix tree of a backward trie [13, 3], but its design is quite different from the aforementioned approach by Gusfield [11]. Indeed, no $O(n)$-space $O(1)$-time outward LCE query data structures on tries are known to date (this is because the size of the suffix tree of a forward trie is $\Omega(n^2)$ [12]).

Technically speaking, our algorithm is most related to the suffix-tree based offline algorithm of Rubinchik and Shur (Proposition 4.10 in [15]) that builds the *eertree*, which is a trie-based data structure storing all distinct palindromes in a given string S. In the preprocessing phase of their method, the Manacher algorithm is used to compute all maximal palindromes in S. Our first finding in this paper is that this preprocessing phase can indeed be omitted, and instead one can use the suffix links of the leaves to check the maximality of given substring palindromes. This is helpful in our scenario since the application of the Manacher algorithm to a trie takes $O(n \log h)$ time [7]. The new suffix-link-based method can simultaneously compute all distinct palindromes (or alternatively the eertree) together with all maximal palindromes, in $O(m)$ time and space, for a given string S of length m. However, the time analysis of the suffix-link-based method is due to the fact that each leaf of the suffix tree of a string has exactly one in-coming suffix link (except for the leaf representing the whole string, which has no in-coming suffix link), and the cost of checking an in-coming suffix link to a leaf can be charged to either a unique distinct palindrome or a unique occurrence of a maximal palindrome in S. Unfortunately, in our trie case, there can be at most σ in-coming suffix links to a single leaf of the suffix tree for a trie, where σ is the alphabet size. Since we need to check whether the palindrome in question can be extended with one of at most σ

candidate characters, naïve suffix-link-based methods would require $\Omega(n \log \sigma)$ time for trie inputs, which is prohibitive for large alphabets. Still, we show an $O(n)$-time suffix-link-based method for computing all distinct/maximal palindromes in the input trie.

We also present how our trie-based $O(n)$-size data structure for storing all maximal/distinct palindromes in the input trie $\mathcal{T}$ can be used for reporting all maximal/distinct palindromes in the strings stored in $\mathcal{T}$, in *output optimal* time, without expanding the trie to the plain strings or extracting a string of interest from the trie. Note that the total length of the strings stored in $\mathcal{T}$ can be as large as $O(n^2)$.

2 Preliminaries

2.1 Strings and palindromes

Let Σ be the *alphabet*. An element of Σ^* is called a *string*. The length of a string S is denoted by $|S|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. For any non-negative integer k, let Σ^k denote the set of strings of length k. For a string $S = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of S, respectively. A prefix (resp. suffix) of T of length less than $|S|$ is called a *proper prefix* (resp. *proper suffix*) of S.

For a string S and an integer $1 \leq i \leq |S|$, $S[i]$ denotes the ith character of S, and for two integers $1 \leq i \leq j \leq |S|$, $S[i..j]$ denotes the substring of S that begins at position i and ends at position j. For convenience, let $S[i..j] = \varepsilon$ when $i > j$.

Let S^R denote the reversed string of S, i.e., $S^R = S[|S|] \cdots S[1]$. A string S is called a *palindrome* if $S = S^R$. We remark that the empty string ε is also considered to be a palindrome. Let $\mathbb{P}$ denote the set of all palindromes over Σ.

For any substring palindrome $S[i..j]$ in S, $\frac{i+j}{2}$ is called its *center*. A substring palindrome $S[i..j]$ is said to be a *maximal palindrome* in S if (a) $S[i-1..j+1]$ is not a palindrome (or equivalently $S[i-1] \neq S[j+1]$), (b) $i = 1$, or (c) $j = |S|$. It is clear that for each $c = 1, 1.5, \ldots, |S|$, there is a one-to-one correspondence between $c = \frac{i+j}{2}$ and the maximal palindrome $S[i..j]$. Thus, there are exactly $2|S| - 1$ maximal palindromes in string S.

2.2 Tries and palindromes

A *trie* $\mathcal{T} = (V, E)$ is a rooted tree where each edge in E is labeled by a single character from Σ and the out-going edges of each node are labeled by mutually distinct characters. For any non-root node $\mathbf{u}$ in $\mathcal{T}$, let $\mathsf{parent}(\mathbf{u})$ denote the parent of $\mathbf{u}$. For any node $\mathbf{v}$ in $\mathcal{T}$, let $\mathsf{childchar}(\mathbf{v})$ denote the set of out-going edge labels of $\mathbf{v}$. Let $\mathsf{height}(\mathcal{T})$ denote the height of $\mathcal{T}$. For convenience, we read the path labels in the input trie $\mathcal{T}$ in the leaf-to-root direction. For any path $(\mathbf{u}, \mathbf{v})$, we denote by $\mathsf{str}(\mathbf{u}, \mathbf{v})$ the path label from $\mathbf{u}$ to $\mathbf{v}$. The set $\mathsf{Substr}(\mathcal{T})$ of (reversed) *substrings* in $\mathcal{T}$ is $\mathsf{Substr}(\mathcal{T}) = \{\mathsf{str}(\mathbf{u}, \mathbf{v}) \mid \mathbf{u}, \mathbf{v} \in V, \mathbf{u} \text{ is a descendant of } \mathbf{v}\}$. Note that $\mathbf{v}$ itself is a descendant of $\mathbf{v}$, i.e., the empty string $\varepsilon = \mathsf{str}(\mathbf{v}, \mathbf{v})$ is an element of $\mathsf{Substr}(\mathcal{T})$. For a string w and a node $\mathbf{u}$ in $\mathcal{T}$, the pair $(\mathbf{u}, |w|)$ is called an occurrence of w in $\mathcal{T}$ if w is a prefix of the substring starting at $\mathbf{u}$ in $\mathcal{T}$. Such representations of occurrences allow us to retrieve the substring w in $O(|w|)$ time by taking the unique path from $\mathbf{u}$ towards the root. Also, we enhance the input trie $\mathcal{T}$ with a *level ancestor* data structure [2] so that any character in a given path $(\mathbf{u}, \mathbf{v})$ can be accessed in $O(1)$ time, after linear-time preprocessing.

We can generalize the notions of maximal palindromes and distinct palindromes to tries, as follows. An occurrence of a substring palindrome p which starts at node $\mathbf{u}$ and ends at node $\mathbf{v}$ is called a *maximal palindrome* in $\mathcal{T}$ if (a) $\mathbf{u}$ is not a leaf and $\mathsf{str}(\mathbf{u}', \mathsf{parent}(\mathbf{v}))$ is not a palindrome

with *any* child $\mathbf{u}'$ of $\mathbf{u}$, (b) $\mathbf{u}$ is a leaf, or (c) $\mathbf{v}$ is the root $\mathbf{r}$. Let $\mathsf{MPal}(\mathcal{T})$ be the set of all maximal palindromes in $\mathcal{T}$. For each $1 \leq k \leq \mathsf{height}(\mathcal{T})$, let $\mathsf{MPal}_k(\mathcal{T}) = \{(\mathbf{v}, k) \in \mathsf{MPal}(\mathcal{T})\}$, and let $\mathsf{MPal}_0(\mathcal{T}) = \{\varepsilon\}$. Namely $\bigcup_{0 \leq k \leq \mathsf{height}(\mathcal{T})} \mathsf{MPal}_k(\mathcal{T}) = \mathsf{MPal}(\mathcal{T})$.

For any trie $\mathcal{T}$, let $\mathsf{DPal}(\mathcal{T})$ be the set of all substring palindromes in $\mathcal{T}$, namely $\mathsf{DPal}(\mathcal{T}) = \mathsf{Substr}(\mathcal{T}) \cap \mathbb{P}$. We call the elements of $\mathsf{DPal}(\mathcal{T})$ as *distinct palindromes* in $\mathcal{T}$. For each $0 \leq k \leq \mathsf{height}(\mathcal{T})$, let $\mathsf{DPal}_k(\mathcal{T}) = \mathsf{DPal}(\mathcal{T}) \cap \Sigma^k$. Namely $\bigcup_{0 \leq k \leq \mathsf{height}(\mathcal{T})} \mathsf{DPal}_k(\mathcal{T}) = \mathsf{DPal}(\mathcal{T})$. See Figure 1 for an example of trie $\mathcal{T}$ and its substring palindromes.

▶ **Theorem 1** ([7]). *For any trie $\mathcal{T}$ with n edges and l leaves, $|\mathsf{MPal}(\mathcal{T})| = 2n - l$ and $|\mathsf{DPal}(\mathcal{T})| \leq n + 1$.*

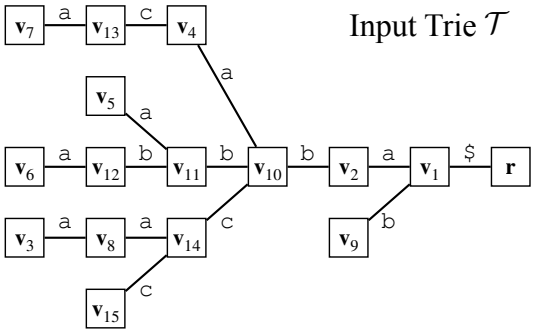


Figure 1 An example for an input trie $\mathcal{T}$ with $n = 15$ edges, in which the path labels are read from the leaves to the root $\mathbf{r}$. The height of $\mathcal{T}$ is $\mathsf{height}(\mathcal{T}) = 6$. Let us focus on the node $\mathbf{v}_{14}$. Then $\mathsf{parent}(\mathbf{v}_{14}) = \mathbf{v}_{10}$, $\mathsf{childchar}(\mathbf{v}_{14}) = \{\mathtt{a}, \mathtt{c}\}$. Also, $\mathsf{str}(\mathbf{v}_{14}, \mathbf{v}_1) = \mathtt{cba}$. The occurrence $(\mathbf{v}_{10}, 1)$ of palindrome $\mathtt{b}$ is not a maximal palindrome since it can be extended to a longer palindrome $\mathsf{str}(\mathbf{v}_4, \mathbf{v}_1) = \mathtt{aba}$. This occurrence $(\mathbf{v}_4, 3)$ of $\mathtt{aba}$ is a maximal palindrome. We have $\mathsf{DPal}(\mathcal{T}) = \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{bb}, \mathtt{cc}, \mathtt{aba}, \mathtt{bbb}, \mathtt{aca}, \mathtt{abba}, \mathtt{abbba}\}$ except for the special character \$.

2.3 Suffix tree and eertree of trie

Suffix tree of a trie. For convenience, we assume that the root $\mathbf{r}$ of the input trie $\mathcal{T}$ has only a single child and its incoming edge from $\mathbf{r}$ is labeled by a special character \$ that does not occur elsewhere in $\mathcal{T}$.

For any node $\mathbf{v}$ in $\mathcal{T}$, let $\mathsf{suf}(\mathbf{v}) = \mathsf{str}(\mathbf{v}, \mathbf{r})$. The set $\mathsf{Suffix}(\mathcal{T})$ of (reversed) suffixes of a given trie $\mathcal{T} = (V, E)$ is $\mathsf{Suffix}(\mathcal{T}) = \{\mathsf{suf}(\mathbf{v}) \mid \mathbf{v} \in V\}$.

The *suffix tree* of $\mathcal{T}$, denoted $\mathsf{STree}(\mathcal{T})$, is a compacted trie such that

- Each edge of $\mathsf{STree}(\mathcal{T})$ is labeled by a non-empty reversed substring of $\mathcal{T}$;
- The labels of the edges from each node of $\mathsf{STree}(\mathcal{T})$ begin with mutually distinct characters;
- There is a one-to-one correspondence between the leaves of $\mathsf{STree}(\mathcal{T})$ and the non-root nodes in $\mathcal{T}$ such that every suffix in $\mathsf{Suffix}(\mathcal{T})$ is represented by a unique leaf in $\mathsf{STree}(\mathcal{T})$.

For any reversed substring w in the input trie $\mathcal{T}$, the *locus* for w in the respective suffix tree $\mathsf{STree}(\mathcal{T})$ is the ending point of the path which spells out w from the root of $\mathsf{STree}(\mathcal{T})$. Note that the locus is either on a node or in the middle of an edge. The loci ending in the middle of edges are called *implicit nodes*, while the loci ending at nodes are called *explicit nodes*. For any implicit or explicit node v of $\mathsf{STree}(\mathcal{T})$, let $\mathsf{str}(v)$ denote the path string from the root of $\mathsf{STree}(\mathcal{T})$ to v. Then, we say that the node v *represents* the substring $\mathsf{str}(v)$.

By the third property of $\mathsf{STree}(\mathcal{T})$, there are exactly n leaves in $\mathsf{STree}(\mathcal{T})$. Since each of the internal explicit nodes has at least two children, there are at most $n-1$ internal nodes in $\mathsf{STree}(\mathcal{T})$. Thus there are at most $2n-1$ nodes and $2n-2$ edges in $\mathsf{STree}(\mathcal{T})$. Each edge label x of $\mathsf{STree}(\mathcal{T})$ is represented by a pair $(\mathbf{u}, \mathbf{v})$ of nodes in $\mathcal{T}$ such that $x = \mathsf{str}(\mathbf{u}, \mathbf{v})$. This allows us to represent $\mathsf{STree}(\mathcal{T})$ in $O(n)$ space. See Figure 2 for an example $\mathsf{STree}(\mathcal{T})$.

In what follows, we use a common assumption that our alphabet is an integer alphabet of polynomial size in n, where $n > 1$ is the number of edges (and thus the number of characters) in the input trie $\mathcal{T}$. We then sort the characters appearing in the input trie $\mathcal{T}$ in $O(n)$ time by radix-sort, and replace each character appearing in the input trie $\mathcal{T}$ with its lexicographical rank. This ensures that we can work on the integer alphabet $\Sigma = [1..\sigma]$, where $\sigma \leq n$ denotes the number of distinct characters in $\mathcal{T}$. Then $\mathsf{STree}(\mathcal{T})$ can be built in linear time:

▶ **Theorem 2** ([16]). *For any trie $\mathcal{T}$ with n edges where edge labels are drawn from an integer alphabet $[1..n]$, $\mathsf{STree}(\mathcal{T})$ can be built in $O(n)$ time and working space.*

We remark that the algorithm of Theorem 2 builds the *edge-sorted* suffix tree for a given trie $\mathcal{T}$, in which the out-going edges of each node are sorted in lexicographical order. Thus, we can naturally assume that the leaves in $\mathsf{STree}(\mathcal{T})$ are arranged in the lexicographical order from left to right. For each $1 \leq i \leq n$, let ℓ_i denote the ith leaf in $\mathsf{STree}(\mathcal{T})$ which represents the lexicographically ith suffix in $\mathsf{Suffix}(\mathcal{T})$. For each $1 \leq i \leq n$, let $\mathbf{v}_i$ denote the node in the input trie $\mathcal{T}$ that corresponds to the leaf ℓ_i, namely $\mathsf{suf}(\mathbf{v}_i) = \mathsf{str}(\ell_i)$ (see Figure 2 for examples).

The *suffix link* of any non-root explicit node v in $\mathsf{STree}(\mathcal{T})$ points to the explicit node u representing the longest proper suffix of $\mathsf{str}(v)$, i.e. $\mathsf{str}(u) = \mathsf{str}(v)[2..|\mathsf{str}(v)|]$, and we denote this by $\mathsf{SL}(v) = u$. Note that the destination explicit node u always exists. Also, by the definition, for each leaf ℓ_i in the suffix tree, $\mathsf{SL}(\ell_i) = \ell_j$ if and only if $\mathsf{parent}(\mathbf{v}_i) = \mathbf{v}_j$. The first character $\mathsf{str}(v)[1]$ that is dropped when moving from v to u is called the *suffix link label* of $\mathsf{SL}(v)$.

In our algorithms to follow, we will use only the suffix links of the leaves of $\mathsf{STree}(\mathcal{T})$. Since every suffix of $\mathcal{T}$ ends with the unique end-marker \$, the suffix link of a leaf points to another leaf, except for the suffix link of the leaf representing \$ which points to the root. For any leaf ℓ_i in $\mathsf{STree}(\mathcal{T})$, let $\mathsf{inSLchar}(\ell_i)$ denote the set of in-coming suffix link labels. Notice that $\mathsf{inSLchar}(\ell_i) = \mathsf{childchar}(\mathbf{v}_i)$ holds since $\mathsf{SL}(\ell_j) = \ell_i$ iff $\mathsf{parent}(\mathbf{v}_j) = \mathbf{v}_i$. See Figure 2 for examples. For any node v in $\mathsf{STree}(\mathcal{T})$, let $\mathsf{subtree}(v)$ denote the subtree rooted at v, and let $\mathsf{leaves}(v)$ denote the set of leaves in $\mathsf{subtree}(v)$. Given $\mathsf{STree}(\mathcal{T})$, it is easy to compute the suffix links of all leaves of $\mathsf{STree}(\mathcal{T})$ in $O(n)$ time. Thus we can also compute $\mathsf{inSLchar}(\ell)$ for every leaf ℓ in $\mathsf{STree}(\mathcal{T})$ in a total of $O(n)$ time.

We extend the notion of $\mathsf{childchar}(\cdot)$ to the suffix tree so that for each node v in $\mathsf{STree}(\mathcal{T})$, $\mathsf{childchar}(v)$ denotes the set of the first characters of the out-going edge labels of v.

Eertree of a trie. We can naturally generalize the *eertree* (a.k.a. *palindromic tree*) which represents the distinct palindromes of a single string [15] to the case of a trie $\mathcal{T}$, as follows: The eertree for a trie $\mathcal{T}$, denoted $\mathsf{eertree}(\mathcal{T})$, is a pair of two rooted tries (the even-tree and odd-tree) where the root of the even-tree represents ε and the root of the odd-tree is an auxiliary node $\perp$. Then, for each $p \in \mathsf{DPal}(\mathcal{T})$, there is a path that spells out $p[\lfloor |p|/2 \rfloor + 1..|p|]$ from the root of the even-tree if $|p|$ is even, or from the root of the odd-tree otherwise. The *suffix link* of a node representing a palindrome p in $\mathsf{eertree}(\mathcal{T})$ points to the node that represents the longest palindrome q, which is a proper suffix of p (the suffix links may bridge the nodes of the even-tree and the nodes of the odd-tree). The suffix links of the two roots ε and $\perp$ both point to $\perp$. See Figure 3 for an example of the eertree for a trie.

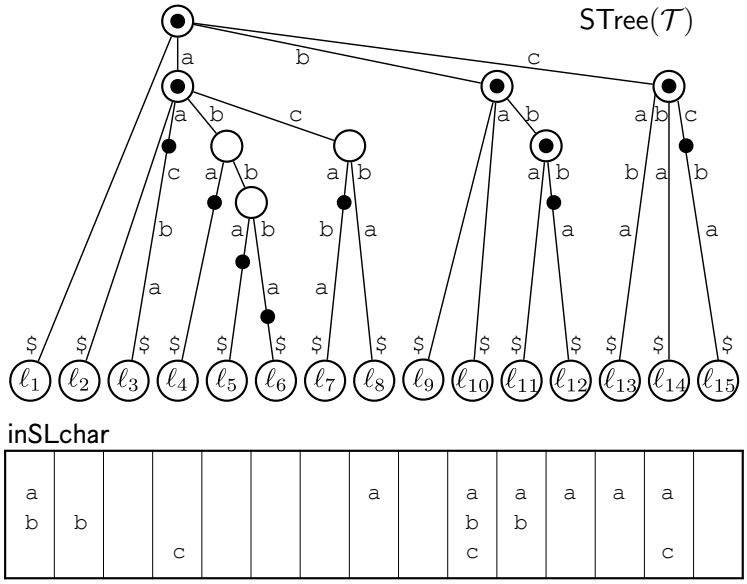


Figure 2 The suffix tree $\mathsf{STree}(\mathcal{T})$ of the input trie $\mathcal{T}$ of Figure 1. The explicit nodes are depicted by the white circles, while the palindromic nodes representing the elements of $\mathsf{DPal}(\mathcal{T})$ are depicted by the black circles. The sets $\mathsf{inSLchar}(\ell_i)$ for all ℓ_i are depicted below the leaves. Each $\mathsf{inSLchar}(\ell_i)$ stores the in-coming suffix link labels of leaf ℓ_i. The suffix links are omitted in this figure, however, we can verify this $\mathsf{inSLchar}$ is correct by looking at trie $\mathcal{T}$ since $\mathsf{inSLchar}(\ell_i) = \mathsf{childchar}(\mathbf{v}_i)$ holds.

3 Algorithm for computing $\mathsf{DPal}(\mathcal{T})$ and $\mathsf{MPal}(\mathcal{T})$

This section presents our algorithm for computing $\mathsf{MPal}(\mathcal{T})$ and $\mathsf{DPal}(\mathcal{T})$ for a given trie $\mathcal{T}$.

3.1 Overview of our algorithm

We use $\mathsf{STree}(\mathcal{T})$ as our main tool for computing $\mathsf{DPal}(\mathcal{T})$ and $\mathsf{MPal}(\mathcal{T})$. An (implicit or explicit) node v on $\mathsf{STree}(\mathcal{T})$ such that $\mathsf{str}(v) \in \mathbb{P}$ is called a *palindromic node*. See also Figure 2, in which the palindromic nodes are depicted by the black circles. By Theorem 1, there are exactly $|\mathsf{DPal}(\mathcal{T})| \in O(n)$ palindromic nodes in $\mathsf{STree}(\mathcal{T})$, and thus, computing $\mathsf{DPal}(\mathcal{T})$ can be reduced to computing all palindromic nodes in $\mathsf{STree}(\mathcal{T})$.

We compute $\mathsf{DPal}(\mathcal{T})$ together with $\mathsf{MPal}(\mathcal{T})$ in increasing order of the lengths of the palindromes. Namely, we visit all palindromic nodes in $\mathsf{STree}(\mathcal{T})$, in the level-wise breadth first manner. Also, for each found palindrome p in increasing order of their length, we check whether it has an occurrence as a maximal palindrome in $\mathcal{T}$. This can be done as follows:

Our strategy for computing $\mathsf{DPal}(\mathcal{T})$ and $\mathsf{MPal}(\mathcal{T})$

- **Base steps:** Start from the root $\mathbf{r}$ of $\mathsf{STree}(\mathcal{T})$. Report ε as a member of $\mathsf{DPal}_0(\mathcal{T})$, and report all single characters occurring in $\mathcal{T}$, as members of $\mathsf{DPal}_1(\mathcal{T})$.
- **Inductive step:** For each increasing $k = 0, 1, \ldots, \mathsf{height}(\mathcal{T}) - 1$:
  - For each palindrome $p \in \mathsf{DPal}_k(\mathcal{T})$, let v be the locus for p in $\mathsf{STree}(\mathcal{T})$.
    - For each character $c \in \mathsf{childchar}(v)$, perform the following:
      - If $cpc \in \mathsf{Substr}(\mathcal{T})$, then report this new palindrome cpc of length $k+2$ as a new member of $\mathsf{DPal}_{k+2}(\mathcal{T})$.
      - For each occurrence of pc that is not preceded by c in the reversed trie $\mathcal{T}$, report this occurrence of p as a new member of $\mathsf{MPal}_k(\mathcal{T})$.

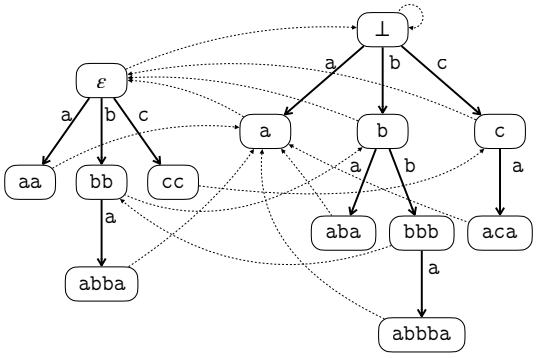


Figure 3 The palindromic tree $\mathsf{eertree}(\mathcal{T})$ for the input trie $\mathcal{T}$ of Figure 1, which represents $\mathsf{DPal}(\mathcal{T}) = \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{bb}, \mathtt{cc}, \mathtt{aba}, \mathtt{bbb}, \mathtt{aca}, \mathtt{abba}, \mathtt{abbba}\}$. The bold arcs represent the edges and the dotted arcs represent the suffix links.

Given a palindrome p of length k together with the corresponding palindromic node v in $\mathsf{STree}(\mathcal{T})$ and $c \in \mathsf{childchar}(v)$, our task is to efficiently check whether or not $cpc \in \mathsf{Substr}(\mathcal{T})$, and if so, then find the locus for cpc in $\mathsf{STree}(\mathcal{T})$. In the sequel, for simplicity, we identify the strings p and pc with their loci (implicit or explicit nodes) in $\mathsf{STree}(\mathcal{T})$.

For a palindrome p and a character c, $cpc \in \mathsf{Substr}(\mathcal{T})$ iff there exists a node $\mathbf{v}_i \in \mathcal{T}$ such that $\mathsf{suf}(\mathbf{v}_i)[1..|p|+1] = pc$ and $\mathbf{v}_i$ has a child that is reachable with character c. Recalling that this is equivalent to that $c \in \mathsf{inSLchar}(\ell_i)$ with the suffix tree leaf ℓ_i that corresponds to $\mathbf{v}_i$, we obtain the two following observations (see also Figure 4).

▶ **Observation 3.** *Let p be a locus in $\mathsf{STree}(\mathcal{T})$ such that $p \in \mathsf{DPal}_k(\mathcal{T})$, and let $c \in \mathsf{childchar}(p)$. Then, $cpc \in \mathsf{DPal}_{k+2}(\mathcal{T})$ iff there exists a suffix tree leaf $\ell \in \mathsf{leaves}(pc)$ such that $c \in \mathsf{inSLchar}(\ell)$.*

▶ **Observation 4.** *Let p be a locus in $\mathsf{STree}(\mathcal{T})$ such that $p \in \mathsf{DPal}_k(\mathcal{T})$, and let $c \in \mathsf{childchar}(p)$. Then, $(\mathbf{v}_i, |p|) \in \mathsf{MPal}_k(\mathcal{T})$ iff $\ell_i \in \mathsf{leaves}(pc)$ and $c \notin \mathsf{inSLchar}(\ell_i)$, where ℓ_i is the suffix tree leaf corresponding to $\mathbf{v}_i$.*

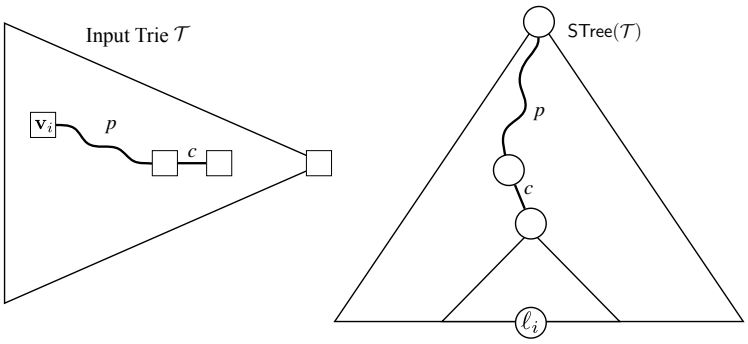


Figure 4 Illustration for Observations 3 and 4.

Following Observation 3 and Observation 4, our task is, given a palindromic node p in $\mathsf{STree}(\mathcal{T})$ and $c \in \mathsf{childchar}(p)$, to perform the following task in *(amortized) constant time.*

(1) Check whether there exists a leaf ℓ under the locus for pc such that $\mathsf{inSLchar}(\ell)$ contains character c, and if so, find it. Then, locate the locus for the extended palindrome cpc in $\mathsf{STree}(\mathcal{T})$.

(2) Check whether there exists a leaf ℓ under the locus for pc such that $\mathsf{inSLchar}(\ell)$ does not contain character c, and if so, retrieve all and only such leaves by skipping the other leaves which do not correspond to the output with respect to pc.

In what follows, we show the details of our algorithm that achieves the above goal, which consists of the preprocessing phase and the computing phase.

3.2 Preprocessing phase

We first enhance $\mathsf{STree}(\mathcal{T})$ with a lowest common ancestor (LCA) data structure in linear preprocessing time so that the LCA of two given nodes can be found in $O(1)$ time [1]. We then compute the *jump-arrays* and some links toward leaves, as follows.

destSL arrays and jump-arrays. For each character c occurring in $\mathcal{T}$, let destSL_c be an array such that $\mathsf{destSL}_c[r] = i$ if the leaf ℓ_i is the lexicographically r-th leaf that has an in-coming suffix link labeled c. For convenience, we add an extra element at the end of destSL_c that stores ∞. Also, we sometimes regard destSL_c as a list of the corresponding leaves ℓ_i. We then define the *jump-array* jump_c which leads us to the ending point of the run of adjacent leaves in $\mathsf{STree}(\mathcal{T})$ for each given leaf ℓ_i in destSL_c. Formally, $\mathsf{jump}_c[i] = i + 1$ if $\mathsf{destSL}_c[i{+}1] - \mathsf{destSL}_c[i] > 1$, and $\mathsf{jump}_c[i] = \min\{j \mid j > i \text{ and } \mathsf{destSL}_c[j{+}1] - \mathsf{destSL}_c[j] > 1\}$ otherwise. See Figure 5 for examples. Note that the total sizes of destSL_c and jump_c for all c occurring in $\mathcal{T}$ is $O(n)$ since these are linear in the number of suffix links of the leaves. Also, we can compute the arrays destSL_c and jump_c for all characters c occurring in $\mathcal{T}$ in linear total time, by radix-sorting all the pairs (c, i) of a character and a leaf-index such that $c \in \mathsf{inSLchar}(\ell_i)$. Simultaneously, we associate each pair (c, i) with its *rank* in destSL_c, i.e., the integer r with $\mathsf{destSL}_c[r] = i$.

Links toward leaves. For each non-leaf node v in $\mathsf{STree}(\mathcal{T})$, we store links from v to the leftmost and the rightmost leaves in $\mathsf{leaves}(v)$. Note that we can compute them easily by performing a depth-first traversal on $\mathsf{STree}(\mathcal{T})$. We further store another link defined below in every non-root node v. For each non-root node v in $\mathsf{STree}(\mathcal{T})$, let c_v be the first character of its in-coming edge label, and further let $\mathsf{leftDest}(v)$ be (some representation of) the leftmost leaf in $\mathsf{leaves}(v)$ such that its leaf-index is in destSL_{c_v} if such a leaf exists, and *nil* otherwise. For technical reasons, we implement $\mathsf{leftDest}(v)$ as the index of the entry of destSL_{c_v} corresponding to the specified leaf. See Figure 5 for examples. Whenever we access an entry of destSL_c, we always take this link from a given node v such that $c_v = c$. Our implementation of $\mathsf{leftDest}(v)$ ensures that we can access the corresponding entry in $O(1)$ time, without the need to search the array $\mathsf{inSLchar}(\ell)$ on some leaf $\ell \in \mathsf{leaves}(v)$ for character c_v, which would use $O(\log \sigma)$ time. Next, we show how to compute the $\mathsf{leftDest}$ for all nodes in linear total time.

As a warm-up, we fix a character c and describe an algorithm which computes $\mathsf{leftDest}(v)$ for all nodes v such that $c_v = c$, in $O(n)$ time. We execute left-to-right depth-first traversal on $\mathsf{STree}(\mathcal{T})$ with a stack st_c storing nodes. When we descend to a node v, push v to st_c if $c_v = c$. When we ascend from a node v, pop v and set $\mathsf{leftDest}(v) \leftarrow nil$ if the top of st_c is v. When we arrive at a leaf ℓ_k, scan $\mathsf{inSLchar}(\ell_k)$ and if $c \in \mathsf{inSLchar}(\ell_k)$ (i.e., $k \in \mathsf{destSL}_c$), pop all nodes v in st_c, and set $\mathsf{leftDest}(v) \leftarrow r$ with $\mathsf{destSL}_c[r] = k$. Recall that such index r has been associated with (c, k) for $c \in \mathsf{inSLchar}(\ell_k)$. During the traversal, st_c stores all nodes v on the path from the root to the current node s.t. $c_v = c$ but $\mathsf{leftDest}(v)$ is not determined yet. Thus, when arriving at a leaf ℓ_k with $c \in \mathsf{inSLchar}(\ell_k)$, we can correctly compute $\mathsf{leftDest}(v)$ for all v s.t. $\mathsf{destSL}_c[\mathsf{leftDest}(v)] = k$.

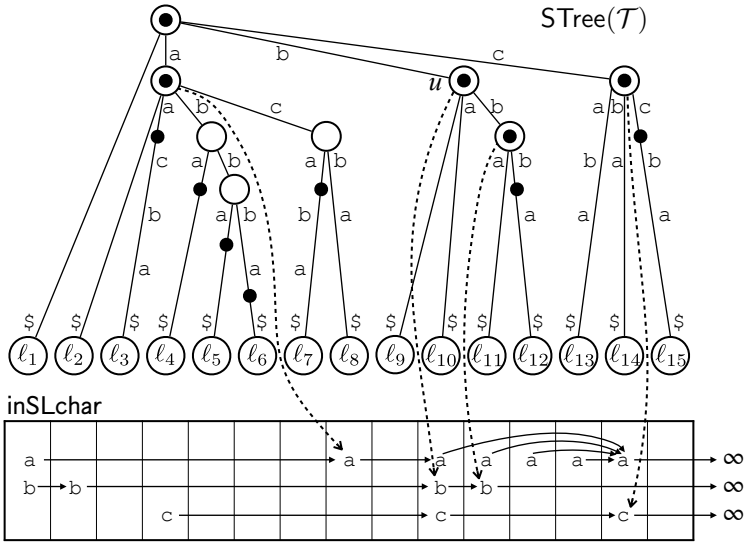


Figure 5 The suffix tree $\mathsf{STree}(\mathcal{T})$ of Figure 2 with some auxiliary data structures. By reading array $\mathsf{inSLchar}$ horizontally for each character c, we can obtain the array destSL_c. Here, $\mathsf{destSL_a} = (1, 8, 19, 10, 11, 12, 13, 14, \infty)$, $\mathsf{destSL_b} = (1, 2, 10, 11, \infty)$, and $\mathsf{destSL_c} = (4, 10, 14, \infty)$. Each element in destSL_c is represented by the character c and is aligned along the corresponding leaf. Also, the arc from each element of destSL_c represents the value stored in the corresponding entry of the jump_c array. Each dotted arrow from an internal node v denotes $\mathsf{leftDest}(v)$. For example, $\mathsf{leftDest}(u)$ of the node u representing string $\mathtt{b}$ points to the third entry of $\mathsf{destSL_b}$, which is represented by the $\mathtt{b}$ under the leaf ℓ_{10}, since the first character c_u of the label of u's parent edge is $c_u = \mathtt{b}$, and the leaf ℓ_{10} is the leftmost leaf in $\mathsf{leaves}(u)$ such that its leaf-index is in $\mathsf{destSL_b}$.

Now, let us remove the constraint of fixing the character c, and perform linear-time computation of $\mathsf{leftDest}(v)$ for all internal nodes v. To compute $\mathsf{leftDest}(v)$ for *all* non-root nodes v, we run the above algorithm for all $c \in \Sigma$ simultaneously in a single depth-first traversal. For this, we use a length-n array $\mathbf{S}$ of stacks in which each entry $\mathbf{S}[c]$ plays the role of st_c as the above. While performing a left-to-right depth-first traversal, when we descend to a node v, push v to $\mathbf{S}[c_v]$, and when we ascend from a node v, pop v and set $\mathsf{leftDest}(v) \leftarrow \mathit{nil}$ if the top of $\mathbf{S}[c_v]$ is v. When we arrive at a leaf ℓ_k, then scan $\mathsf{inSLchar}(\ell_k)$ and for every $c \in \mathsf{inSLchar}(\ell_k)$, pop all nodes v in $\mathbf{S}[c]$ and set $\mathsf{leftDest}(v) \leftarrow r$ with $\mathsf{destSL}_c[r] = k$. The correctness relies on the aforementioned algorithm, and the running time is linear in the size of $\mathsf{STree}(\mathcal{T})$ including the suffix links, that is, $O(n)$.

3.3 Computing phase

In the computing phase, we traverse on $\mathsf{STree}(\mathcal{T})$ and visit the loci representing palindromes in non-decreasing order of the lengths of palindromes. Let us assume that we are now located at a node representing a palindrome p occurring in $\mathcal{T}$. There are the two following cases: (A) node p is an explicit node, and (B) node p is an implicit node.

Algorithm for case (A)

For the case (A), we can enumerate all trie nodes whose length-$|p|$ prefixes are maximal palindromes p in output-sensitive time by using data structures precomputed. Let v be a child of node p and c be the first character of the label of edge (p, v). Let ℓ_{left} and ℓ_{right} be the leftmost and the rightmost leaves in $\mathsf{leaves}(v)$, respectively. Namely, they correspond

to the *left*-th and *right*-th smallest suffixes in $\mathcal{T}$, respectively. We initialize the current-leaf-index $curr \leftarrow \mathsf{destSL}_{c_v}[\mathsf{leftDest}(v)]$ and the previous-leaf-index $prev \leftarrow left - 1$. While $curr \leq right$, output trie nodes $\mathbf{v}_i$ for all i with $prev < i < curr$ if $curr - 1 \notin \mathsf{destSL}_c$, and then set $prev \leftarrow curr$ and $curr \leftarrow \mathsf{destSL}_c[\mathsf{jump}_c[curr]]$. At last, after breaking out from the while-loop condition, output trie nodes $\mathbf{v}_i$ for all i with $prev < i \leq right$ if $curr - 1 \notin \mathsf{destSL}_c$. See also Figure 6 for illustration.

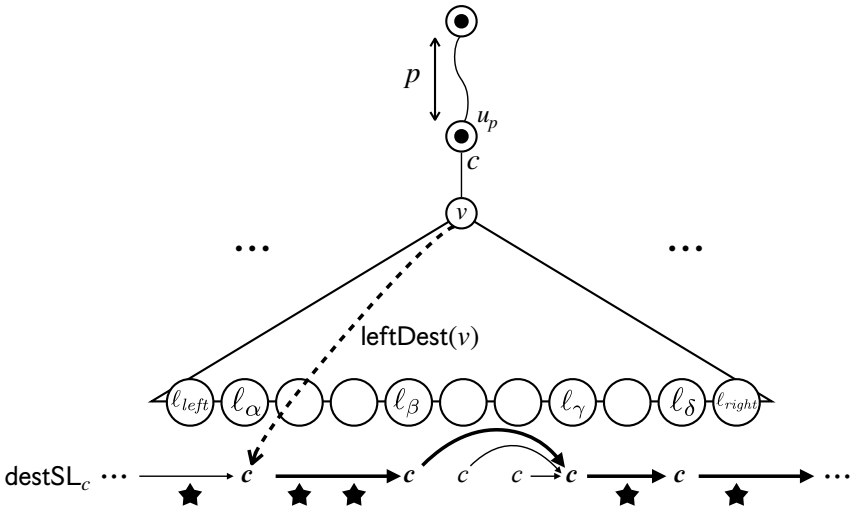


Figure 6 The black stars indicate the leaves in $\mathsf{leaves}(v)$ corresponding to occurrences of palindrome p which are followed by character c but cannot be expanded with c, i.e., occurrences of p as maximal palindromes. In the computing phase, starting from node v, we visit the entries in destSL_c corresponding to the leaves ℓ_α, ℓ_β, ℓ_γ and ℓ_δ in this order, and output all the leaves indicated by black stats. The jump-arrays values that are used in this step are depicted in bold. Note that when we arrive at ℓ_γ, we output nothing since none of the leaves between ℓ_β and ℓ_γ inclusive corresponds to an occurrence of a maximal palindrome.

By Observation 4, this algorithm correctly reports all the occurrences of pc (without duplication) such that the occurrence of p is a maximal palindrome. Next, to evaluate the running time, we show the next lemma:

▶ **Lemma 5.** *In the above algorithm for a node v, when we use jump-pointers three times, at least one maximal palindrome is reported.*

Proof. We assume that we use jump-pointers at least three times on leaves in $\mathsf{subtree}(v)$. Then, at least two jumps are performed inside $\mathsf{subtree}(v)$. (The last jump, which is toward the outside of $\mathsf{subtree}(v)$, is the only exception.)

Let i be the current leaf-index. By the definition of jump-pointers, if $\mathsf{destSL}_c[i+1] - \mathsf{destSL}_c[i] > 1$, then $\mathsf{jump}_c[i] = i+1$. Then, our algorithm reports at least one maximal palindrome since we skip at least one leaf between $\mathsf{destSL}_c[i]$ and $\mathsf{destSL}_c[\mathsf{jump}_c[i]] = \mathsf{destSL}_c[i+1]$ both exclusive. Otherwise, namely if $\mathsf{destSL}_c[i+1] - \mathsf{destSL}_c[i] = 1$, then $\mathsf{jump}_c[i]$ points to the ending point of the run of adjacent leaves in destSL_c. In this case, we report no maximal palindromes. However, in the next step, we track $\mathsf{jump}_c[j]$ where $j = \mathsf{jump}_c[i]$. Then, we fall into the first case $\mathsf{destSL}_c[j+1] - \mathsf{destSL}_c[j] > 1$ since j is the ending position of a run (see also Figure 6). Thus, a maximal palindrome is reported while the second jump. Therefore, performing jumps twice inside $\mathsf{subtree}(v)$ results in a maximal palindrome being reported. ◀

Thus, the running time of the above algorithm for node v is linear in the output size, or is constant if there is no maximal palindrome to output.

After these processes for v, we retrieve the node representing palindrome cpc if it exists. While computing maximal palindromes we can obtain the rightmost leaf ℓ_R such that $\ell_R \in \mathsf{leaves}(v)$ and $R \in \mathsf{destSL}_c$. We move to the source leaves ℓ and ℓ' of the suffix links which respectively point to ℓ_L and to ℓ_R, both labeled by c where $L = \mathsf{destSL}_{c_v}[\mathsf{leftDest}(v)]$ is the leaf-index specified by $\mathsf{leftDest}(v)$. We then compute the LCA u' of ℓ and ℓ'. If the string-depth of u' equals $|cpc|$, the node u' is the desired node corresponding to cpc. Otherwise, the locus on the parent-edge of u' with the string-depth of $|cpc|$ is the (implicit) node corresponding to cpc.

Algorithm for case (B)

For the case (B), the precomputed data structures are not helpful. Instead, we employ the following lemmas for designing an efficient algorithm.

▶ **Lemma 6.** *For any trie $\mathcal{T}$, there is at most one implicit node v on each edge of $\mathsf{STree}(\mathcal{T})$ such that $\mathsf{str}(v)$ is a palindrome.*

Proof. Suppose on the contrary that there are two implicit nodes v and u on the same edge of $\mathsf{STree}(\mathcal{T})$ such that both $\mathsf{str}(v)$ and $\mathsf{str}(u)$ are palindromes. Assume without loss of generality that $|\mathsf{str}(v)| < |\mathsf{str}(u)|$. Since v and u are on the same edge, we have $\mathsf{leaves}(v) = \mathsf{leaves}(u)$ which means that the number of occurrences of $\mathsf{str}(v)$ and $\mathsf{str}(u)$ in $\mathcal{T}$ must be equal. However, since $\mathsf{str}(v)$ is a proper prefix of $\mathsf{str}(u)$ and both of them are palindromes, $\mathsf{str}(v)$ is also a proper suffix of $\mathsf{str}(u)$, meaning that $\mathsf{str}(v)$ occurs at least twice in $\mathsf{str}(u)$. This contradicts that the number of occurrences of $\mathsf{str}(v)$ and $\mathsf{str}(u)$ in $\mathcal{T}$ are equal. Thus there exists at most one implicit node on each edge of $\mathsf{STree}(\mathcal{T})$. ◀

The following lemma states that implicit palindromic nodes have a sort of monotonicity. See also Figure 7 for illustration.

▶ **Lemma 7.** *Let p and q be palindromes such that p is a proper prefix of q. If p corresponds to an implicit node on an edge (u, v) in $\mathsf{STree}(\mathcal{T})$ and its following character is c, then q corresponds to an implicit node on an edge in $\mathsf{subtree}(v)$ and its following character is also c.*

Proof. Since p corresponds to an implicit node, the only character following p is c for all occurrences of p in $\mathcal{T}$. Also, since the palindrome p is a proper prefix of the palindrome q, p occurs in q also as a proper suffix. If we assume that q corresponds to an explicit node, then there have to be two distinct characters following q, and p as well, a contradiction. Thus, q corresponds to an implicit node and its following character is c. Also, by Lemma 6, the implicit node cannot exist on the edge (u, v), and thus, it is in $\mathsf{subtree}(v)$. ◀

Namely, once an implicit node u corresponding to a palindrome is found, any palindrome corresponding to a descendant of u is guaranteed to be followed by the same character (see also Figure 7). Based on this monotonicity, we design an algorithm for the case (B).

At each step of the algorithm, $v.\mathsf{mark} = 1$ means that we already visited an implicit ancestor representing some shorter prefix palindrome. Initially, we set $u.\mathsf{mark} \leftarrow 0$ for every node u. Let v be the shallowest explicit node below p and c be the character following p. If $v.\mathsf{mark} = 0$, every $v' \in \mathsf{subtree}(v)$ is not marked yet since we consider palindromes in non-decreasing order of the lengths. We traverse $\mathsf{subtree}(v)$ and mark all nodes in the subtree. Also, while traversing the subtree, we compute the links $\mathsf{left}(v')$ from each $v' \in \mathsf{subtree}(v')$ defined as follows: $\mathsf{left}(v')$ is the leftmost leaf in $\mathsf{leaves}(v')$ such that its leaf-index is in destSL_c if such a leaf exists, and *nil* otherwise. In other words, $\mathsf{left}(v')$ is a variant of $\mathsf{leftDest}(v')$

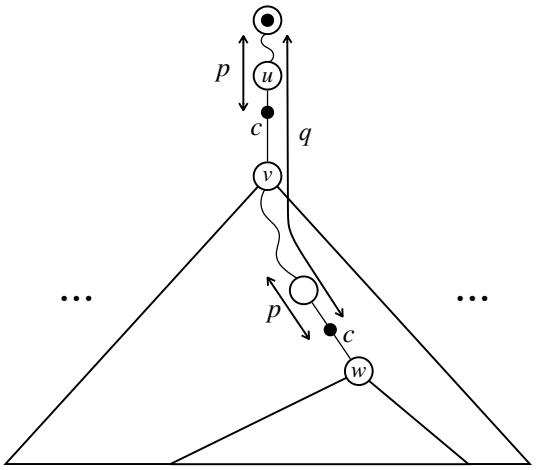


Figure 7 Illustration for the case (B). In our algorithm, we visit the locus of palindrome p before that of palindrome q since q is longer than p. After we have visited the locus of p, each node w' in $\mathsf{subtree}(v)$ have links $\mathsf{left}(w')$ w.r.t. the character c. When we visit the locus of q later, we use $\mathsf{left}(w)$ and destSL_c to report all occurrences of q which are maximal palindromes.

where the character c to be considered is fixed. Traversing $\mathsf{subtree}(v)$ and computing $\mathsf{left}(v')$ for all nodes v' in $\mathsf{subtree}(v)$ can be done in time linear in the size of $\mathsf{subtree}(v)$ plus the total number of incoming suffix links of $\mathsf{leaves}(v)$ as in the computing of $\mathsf{leftDest}$ described in the previous subsection. Once we create the links left, we can compute all occurrences of maximal palindrome p and find the next node for cpc in a similar way to the case (A).

If $v.\mathsf{mark} = 1$, there is an implicit ancestor we already visited. By the above procedures, node v already has link $\mathsf{left}(v)$, and hence, we can output all maximal palindromes p in output linear time. To summarize, the following theorem holds:

▶ **Theorem 8.** *For a given trie $\mathcal{T}$ with n edges over an integer alphabet $\Sigma = [1..n]$, $\mathsf{MPal}(\mathcal{T})$ and $\mathsf{DPal}(\mathcal{T})$ can be computed in $O(n)$ time and space.*

Proof. In the preprocessing phase, $\mathsf{STree}(\mathcal{T})$ can be constructed in $O(n)$ time and space (Theorem 2). An LCA data structure on $\mathsf{STree}(\mathcal{T})$ can be built in $O(n)$ time and space [1], and the remaining data structures can also be built in $O(n)$ time by radix-sort and depth-first traversal on $\mathsf{STree}(\mathcal{T})$.

For the case (A) in the computing phase, most of the procedures can be done in constant time per an edge of $\mathsf{STree}(\mathcal{T})$. The exception is the number of times jump-arrays are used can not be bounded by a constant per an edge. However, by Lemma 5, that can be charged to the maximal palindromes to output, and thus, the total execution time is $O(n + |\mathsf{MPal}(\mathcal{T})|) = O(n)$. For the case (B) in the computing phase, traversing the subtree of a node appears to be the bottleneck. However, the total time to traverse the subtrees is $O(n)$ since each node will be marked at most once throughout the algorithm. The remaining procedures can be done in a total of $O(n)$ time (the analysis is the same as that for the case (A)). ◀

3.4 Computing eertree($\mathcal{T}$)

▶ **Theorem 9.** *For a given trie $\mathcal{T}$ with n edges over an integer alphabet $\Sigma = [1..n]$, the edge-sorted $\mathsf{eertree}(\mathcal{T})$ with suffix links can be computed in $O(n)$ time and space.*

Proof. Since our algorithm computes $\mathsf{DPal}(\mathcal{T})$ in increasing order of the lengths, we can easily compute the tree-topology of $\mathsf{eertree}(\mathcal{T})$ in conjunction with $\mathsf{DPal}(\mathcal{T})$ in $O(n)$ time and space. Recall that the algorithm of Theorem 2 builds the edge-sorted suffix tree. Thus, the edges of the obtained $\mathsf{eertree}(\mathcal{T})$ are already sorted.

Recall that a palindrome q is the longest proper suffix palindrome of another palindrome p iff q is the longest proper prefix palindrome of p. Thus, we can compute the suffix links of the nodes of $\mathsf{eertree}(\mathcal{T})$ by a standard traversal on $\mathsf{STree}(\mathcal{T})$ in which all the palindromic nodes are explicitly inserted as in Figure 2. ◀

4 Answering queries to find palindromes in sub-paths

A trie $\mathcal{T}$ can be regarded as a compact representation of a set $\mathcal{S}_{\mathcal{T}}$ of strings that are the path strings from the leaves to the root of $\mathcal{T}$. This section presents how to report all distinct/maximal palindromes in each string in $\mathcal{S}_{\mathcal{T}}$ upon query, in output-linear time with $O(n)$ space, where n is the size of $\mathcal{T}$. We remark that the total length of the strings in $\mathcal{S}_{\mathcal{T}}$ can be as large as $O(n^2)$, and thus, our $O(n)$-size representation of the palindromes for $\mathcal{S}_{\mathcal{T}}$ is space-efficient. In addition, our algorithms which follow permit us to query on any sub-path for maximal palindromes, and on any suffix-path for distinct palindromes.

4.1 Maximal palindromes in a sub-path

As for maximal palindromes in any path of a given trie, we obtain the following result:

▶ **Theorem 10.** *After $O(n)$-time preprocessing on the input trie $\mathcal{T}$, given a sub-path $(\mathbf{u}, \mathbf{v})$ in $\mathcal{T}$ and a position α that is either a node or an edge on the path $(\mathbf{u}, \mathbf{v})$, the maximal palindrome centered at α in string $\mathsf{str}(\mathbf{u}, \mathbf{v}) \in \mathsf{Substr}(\mathcal{T})$ can be computed in $O(1)$ time.*

Proof. In the preprocessing, we transform the occurrence representation of maximal palindromes into the pair $(\mathbf{s}, \alpha)$ of the starting node $\mathbf{s}$ and its center α, which is either a node or an edge on the suffix-path from $\mathbf{s}$. Such transformation can be done in linear time by traversing $\mathcal{T}$: We maintain an array $\mathcal{N}$ of size $\mathsf{height}(\mathcal{T})$ that stores nodes in the suffix path from the current node so that we can access any ancestor of given depth in constant time. When we visit a node $\mathbf{v}$ and find a maximal palindrome $(\mathbf{v}, k)$, we can compute the $k/2$-th shallower locus α by using array $\mathcal{N}$ in constant time. Simultaneously, we store the pointer from each α to the starting node $\mathbf{s}_\alpha$ of the maximal palindrome centered at α.

Given nodes $\mathbf{u}$, $\mathbf{v}$ and center position α as a query, we first compute the lowest common ancestor of $\mathbf{s}_\alpha$ and $\mathbf{u}$. Let $\mathbf{w}$ be the LCA node. If the center position α is a node $\mathbf{z}$, then the length of the maximal palindrome centered at α in $\mathsf{str}(\mathbf{u}, \mathbf{v})$ is $\min\{2|(\mathbf{w}, \mathbf{z})|, 2|(\mathbf{z}, \mathbf{v})|\}$. Otherwise, namely, if the center position α is an edge $(\mathbf{x}, \mathbf{y})$, then the length of the maximal palindrome centered at α in $\mathsf{str}(\mathbf{u}, \mathbf{v})$ is $\min\{2|(\mathbf{w}, \mathbf{x})|, 2|(\mathbf{y}, \mathbf{v})|\} + 1$. ◀

4.2 Distinct palindromes in a suffix-path

In general, a substring palindrome p in a string S may occur more than once in S. It suffices to report a representative of the occurrences of each palindrome for computing the set of distinct palindromes. The algorithm that computes $\mathsf{DPal}(S)$ for a string S in [10] reports their leftmost occurrences in S. Also, the algorithm that computes $\mathsf{DPal}(\mathcal{T})$ for a trie $\mathcal{T}$ in [7] reports the first occurrence of each palindrome during a depth-first traversal in $\mathcal{T}$. Below, we show how to efficiently report $\mathsf{DPal}(w)$ for any suffix-path string w in the trie $\mathcal{T}$.

▶ **Theorem 11.** *After $O(n)$-time preprocessing on the input trie $\mathcal{T}$, given a node $\mathbf{u}$ in $\mathcal{T}$, all the distinct palindromes in the suffix-path string $\mathsf{suf}(\mathbf{u}) \in \mathsf{Suffix}(\mathcal{T})$ can be enumerated in output-linear time.*

Proof. An occurrence $(\mathbf{v}, |w'|)$ of string w' is called a *rightmost* occurrence in $\mathcal{T}$ if there is no occurrence of w' in $\mathsf{suf}(\mathbf{v})$ other than $(\mathbf{v}, |w'|)$. As the representative of the occurrences of a palindrome p in $\mathcal{T}$, we choose a rightmost one. Our task is to enumerate rightmost occurrences of palindromes in a given suffix $\mathsf{suf}(\mathbf{u}) \in \mathsf{Suffix}(\mathcal{T})$. Let $lpp_{\mathbf{v}}$ be the longest prefix palindrome starting at a trie node $\mathbf{v}$. If $(\mathbf{v}, |p|)$ is a rightmost occurrence of a palindrome p, then $p = lpp_{\mathbf{v}}$ holds (if not, p occurs in $\mathsf{suf}(\mathbf{v})$ as a proper suffix of $lpp_{\mathbf{v}}$ since they are palindromes, a contradiction). Therefore, once we have computed the set $\mathcal{R} = \{\mathbf{v} \mid (\mathbf{v}, |lpp_{\mathbf{v}}|)$ is a rightmost occurrence of $lpp_{\mathbf{v}}$ in $\mathcal{T}\}$ and have *marked* the trie nodes in $\mathcal{R}$, the set $\mathsf{DPal}(w)$ for $w = \mathsf{suf}(\mathbf{u})$ can be computed by searching all the marked nodes on the suffix-path from $\mathbf{u}$ in $\mathcal{T}$ when a query node $\mathbf{u}$ is given. The search can be done in $O(|\mathsf{DPal}(w)|)$ time by recursively querying the nearest marked ancestor (NMA) from $\mathbf{u}$ until it reaches the root. Note that $O(1)$-time static NMA queries can be preprocessed by an $O(n)$-time standard traversal on $\mathcal{T}$.

In the following, we show how to precompute the set $\mathcal{R}$ in linear time. First, we compute $lpp_{\mathbf{v}}$ for each node $\mathbf{v}$. This can be done in linear time by traversing on $\mathsf{STree}(\mathcal{T})$ since $lpp_{\mathbf{v}_i}$ corresponds to the nearest palindromic ancestor node of the leaf ℓ_i in $\mathsf{STree}(\mathcal{T})$. Also, we store a stack in each palindromic node in $\mathsf{STree}(\mathcal{T})$. Then, we check whether $lpp_{\mathbf{v}}$ is the rightmost occurrence for each node $\mathbf{v}$ by performing a depth-first traversal on $\mathcal{T}$ as follows: During the traversal, when descending from node $\mathbf{v}$, we push the node onto the stack of the palindromic node representing $lpp_{\mathbf{v}}$. If the stack is empty just before the push, then $\mathbf{v} \in \mathcal{R}$ holds since $lpp_{\mathbf{v}}$ is the rightmost occurrence, and thus, we mark the node $\mathbf{v}$. When ascending from node $\mathbf{v}$, we pop the node from the stack of the palindromic node representing $lpp_{\mathbf{v}}$. ◀

Longest palindromes in suffix-paths

Let $L(\mathbf{v})$ denote the length of a longest palindrome in $\mathsf{suf}(\mathbf{v})$ in the input trie $\mathcal{T}$. For each non-root node $\mathbf{v}$, $L(\mathbf{v}) = \max\{|lpp_{\mathbf{v}}|, L(\mathsf{parent}(\mathbf{v})\}$ holds. Thus, after computing the lengths of all $lpp_{\mathbf{v}}$, we can compute all $L(\mathbf{v})$ in a top-down manner. Also, we can associate the length $L(\mathbf{v})$ with the corresponding occurrence while computing them. As we proved in Theorem 11, we can compute $lpp_{\mathbf{v}}$ for each node $\mathbf{v}$ in linear time. The next corollary holds:

▶ **Corollary 12.** *We can compute a longest palindrome in each suffix-path $\mathsf{suf}(\mathbf{v}) \in \mathsf{Suffix}(\mathcal{T})$ in $O(n)$ total time.*

5 Conclusions and open questions

This paper proposed the *first* $O(n)$-time algorithm which computes all distinct palindromes and all maximal palindromes in a given trie $\mathcal{T}$ of size n, where the edge labels are drawn from an integer alphabet of size polynomial in n.

In the case of a general ordered alphabet of size σ, one can first sort the edge labels of $\mathcal{T}$ in $O(n \log \sigma)$ time with $O(n)$ space and replace the edge labels with their lexicographical ranks in range $[1..n]$, so our algorithms work in $O(n \log \sigma)$ time and $O(n)$ space.

It is open whether there exists an $O(n)$-time algorithm for computing distinct/maximal palindromes in a trie for general ordered alphabets. To achieve this goal, it is prohibited to construct edge-sorted suffix trees since there is an $\Omega(n \log \sigma)$-time lower bound [6].

It is known that there can be $\Theta(n^{3/2})$ distinct palindromes in an *unrooted* edge-labeled tree of size n [4, 8], and all of them can be computed in $O(n^{3/2} \log n)$ time [9]. It is open whether there is an $O(n^{3/2})$-time solution for this problem.

References

1. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *4th Latin American Theoretical Informatics Symposium, LATIN 2000*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
2. Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.
3. Dany Breslauer. The suffix tree of a tree and minimizing sequential transducers. *Theor. Comput. Sci.*, 191(1-2):131–144, 1998. doi:10.1016/S0304-3975(96)00319-2.
4. Srecko Brlek, Nadia Lafrenière, and Xavier Provençal. Palindromic complexity of trees. In *19th International Conference on Developments in Language Theory, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2015. doi:10.1007/978-3-319-21500-6_12.
5. Xavier Droubay, Jacques Justin, and Giuseppe Pirillo. Episturmian words and some constructions of de Luca and Rauzy. *Theor. Comput. Sci.*, 255(1-2):539–553, 2001. doi:10.1016/S0304-3975(99)00320-5.
6. Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.
7. Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing maximal palindromes and distinct palindromes in a trie. In *Proceedings of the Prague Stringology Conference, PSC 2019*, pages 3–15. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2019. URL: http://www.stringology.org/event/2019/p02.html.
8. Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Tight bound for the number of distinct palindromes in a tree. In *22nd International Symposium on String Processing and Information Retrieval, SPIRE 2015*, volume 9309 of *Lecture Notes in Computer Science*, pages 270–276. Springer, 2015. doi:10.1007/978-3-319-23826-5_26.
9. Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Tight bound for the number of distinct palindromes in a tree. *CoRR*, abs/2008.13209, 2020. doi:10.48550/arXiv.2008.13209.
10. Richard Groult, Élise Prieur, and Gwénaël Richomme. Counting distinct palindromes in a word in linear time. *Inf. Process. Lett.*, 110(20):908–912, 2010. doi:10.1016/j.ipl.2010.07.018.
11. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.
12. Shunsuke Inenaga. Towards a complete perspective on labeled tree indexing: New size bounds, efficient constructions, and beyond. *J. Inf. Process.*, 29:1–13, 2021. doi:10.2197/ipsjjip.29.1.
13. S. Rao Kosaraju. Efficient tree pattern matching (preliminary version). In *30th Annual Symposium on Foundations of Computer Science, FOCS 1989*, pages 178–183. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63475.
14. Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975. doi:10.1145/321892.321896.
15. Mikhail Rubinchik and Arseny M. Shur. EERTREE: an efficient data structure for processing palindromes in strings. *Eur. J. Comb.*, 68:249–265, 2018. doi:10.1016/j.ejc.2017.07.021.
16. Tetsuo Shibuya. Constructing the suffix tree of a tree with a large alphabet. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 86-A(5):1061–1066, 2003. URL: http://search.ieice.org/bin/summary.php?id=e86-a_5_1061.
17. Wing-Kin Sung. *Algorithms in Bioinformatics: A Practical Introduction*. Chapman and Hall/CRC, 2009.
18. Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, SWAT 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

Distortion-Oblivious Algorithms for Scheduling on Multiple Machines

Yossi Azar
Tel Aviv University, Israel

Eldad Peretz
Tel Aviv University, Israel

Noam Touitou
Tel Aviv University, Israel
Amazon, Tel Aviv, Israel[1]

Abstract

We consider the classic online problem of scheduling on multiple machines to minimize total flow time and total stretch where the input consists of estimates on the processing time provided for each job once released. The performance of such algorithms should depend on μ, the error in the estimates of the processing time for that instance (such an algorithm is called a distortion oblivious algorithm). Previously, a distortion oblivious algorithm to minimize flow time was provided only for a single machine. In this paper we extend the work to multiple machines and also consider the total stretch objective. In particular, we design a non-migrative distortion oblivious algorithm to minimize total flow time with a competitive ratio of $O(\mu \log P)$, where P is the ratio between the maximum to minimum processing time. We show that with immediate-dispatching one cannot achieve a competitive ratio which is a function of μ and P; moreover, a competitive ratio which is sub-polynomial in the number of jobs is also impossible. We also present the first distortion-oblivious algorithm for minimizing the stretch time, both on a single and on multiple machines. The competitive ratio of these algorithms are $O(\mu^2)$ which is optimal as we also prove a matching $\Omega(\mu^2)$ lower bound.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online, Scheduling, Predictions, Stretch, Flow Time

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.16

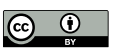
Funding *Yossi Azar*: Supported in part by the Israel Science Foundation (grant No. 2304/20).

1 Introduction

We consider the online scheduling problem on multiple parallel machines, where n jobs arrive over time and the completion of a job requires processing time on the $m \geq 1$ machines. The machines are parallel, so at at any given time a job can be executed on a single machine only. The goal of a scheduling algorithm is to minimize a certain objective function. In this paper, we consider both the *total flow time* objective, which is the sum of the jobs flow-time (time from release to completion), and the *total stretch* objective, which normalizes the flow time of each job by the required processing time for that job. We consider the variant of the problem in which preemption is allowed (i.e., the algorithm is allowed to halt and resume the processing of jobs as desired).

In **classic scheduling**, the processing times of the jobs are exactly known at the job release time. In a single machine setting, it is well known that the algorithm *SRPT* (shortest remaining processing time) [23] is 1-competitive. It has been shown that on multiple machines

[1] This work was done prior to joining Amazon.



the problem is more difficult and depends on the ratio between the largest processing time of a job to the minimum one (denoted by P). Specifically, every online algorithm is $\Omega(\log P)$ competitive. In addition, $SRPT$ algorithm is an optimal online scheduling algorithm which achieves a tight competitive ratio of $O(\log P)$ [15].

However, in practice usually the assumption that the job processing time is known at the job release time does not hold. For example, in computer programs scheduling it is very unlikely to know the job's exact processing time in advance.

Scheduling with estimates, introduced in [4], addresses this lack of processing-time knowledge. In this setting, upon job release the algorithm is provided with an *estimate* of the job's processing time, which might be inaccurate up to a multiplicative error of μ; this parameter μ is called the *distortion* of the input. Naturally, as μ increases, the best achievable competitive ratio becomes worse. A possible result in this model is a *robust family of algorithms* $\{ALG_\mu\}$, such that ALG_μ is tailored to appropriately handle inputs of distortion at most μ. To use such a family, one needs to know the distortion of the input μ in advance to choose the correct algorithm. However, in practical settings, knowing the distortion in advance is infeasible; moreover, guessing the distortion could yield unbounded competitiveness (see [5]). Thus, a preferable result is a single algorithm (rather than a family) that handles any input, while achieving a competitive ratio as a function of the distortion of that input. Such an algorithm is called *distortion-oblivious*. In our paper, we define the parameter P to be the maximum ratio between *estimated* job sizes. We focus on presenting distortion-oblivious algorithms for scheduling with estimates.

In this paper, we consider preemptive **scheduling on multiple machines**. In such scheduling problems, certain properties are desired in a good algorithm. For example, such a property is being *non-migrative*, i.e., once the algorithm decides to process a job on a machine, the job can never be processed on a different machine. Another, stronger property is *immediate dispatching*, in which a job must also be assigned to a machine immediately upon its release.

1.1 Our Results

In this paper, we present the following distortion-oblivious algorithms and lower bounds for scheduling with estimates.

1. We show a non-migrative algorithm with a competitive ratio of $O(\mu \log P)$ with respect to the total flow time objective compared to optimal migrative algorithm (Theorem 2). If migration is allowed we show using similar analysis that lowest class first also achieves a competitive ratio of $O(\min(\mu \log P, \mu \log \mu + \mu \log \frac{n}{m}))$ (Corollary 16).
2. We present a tight non-migrative algorithm for minimizing total stretch on multiple machines with a competitive ratio of $O(\mu^2)$. (Theorem 6)
3. We present a matching lower bound for minimizing total stretch on multiple machines (that even allows migration) of $\Omega(\mu^2)$. (Theorem 11)

In particular, Results 2, 3 also apply for a single machine, and are the first known results in this setting. Therefore, in this paper we completely solve the problem of distortion-oblivious algorithms to minimize stretch time.

We also consider the immediate dispatching property for scheduling with estimates. Here, we show that there exists no algorithm of competitiveness sub-polynomial in the number of jobs, even for arbitrarily-small μ and P; in particular, there is no algorithm with competitive ratio as a function of only μ and P. This lower bound applies to both total flow time and total stretch, and holds even when randomization is allowed. Details appear in Section 5.

1.2 Related Work

A similar setting to the robust setting is nonclairvoyant scheduling, studied in [21, 8, 7, 11, 10]. In this setting, the size of each job is completely unknown at release time, and is only revealed to the algorithm upon completion of the job. In fact, this non-clairvoyant setting is a special case of our model in which the distortion μ is equal to P (i.e., the predicted processing time is meaningless). In this nonclairvoyant model, for multiple machines, the best known result for minimizing flow time is $O(\log n \log \frac{n}{m})$ [8].

In the classic scheduling setting, the exact processing times of jobs are known at release time; this classic model is a special case of the robust setting in which $\mu = 1$. In this setting, [15] presented an $O(\log P)$ algorithm and a matching lower bound of $\Omega(\log P)$. It is also known that the same algorithm achieves an upper bound of $O(\log(n/m))$; without migration, [3] presented an $O(\log n)$-competitive algorithm. A result of similar competitiveness was introduced by [2] for the immediate dispatching variant of this problem. For minimizing stretch, an $O(1)$ non-migrative competitive algorithm is known [9].

Scheduling with distortion falls within the field of algorithms with predictions. In this model, an online algorithm is given somewhat-accurate predictions of the incoming online input. A good algorithm should be able to benefit from these predictions to the degree to which they are precise; its competitive ratio would thus be a function of the predictions' accuracy. Many problems related to scheduling have been addressed using algorithms with predictions; for example, see [22, 13, 19, 12, 6, 16]. Algorithms with predictions have also been used for many other online problems [17, 18, 20]. Additional papers on algorithms with predictions can be found in [1].

1.3 Paper Organization

The non-migrative algorithm for minimizing total flow time is presented and analyzed in Section 3. The non-migrative algorithm for minimizing total stretch is presented and analyzed in Section 4. In Section 5 we show lower bound for immediate dispatching distortion oblivious algorithms, for both the flow time and total stretch objectives. In Appendix A, we show additional results for the total flow time objective; specifically, we analyze the LCF algorithm and give tight examples for the shown algorithms.

2 Preliminaries

In our scheduling problem, jobs are released over time. There are m parallel machines; that means at any given time a job can be processed only on one of the m machines. Each job q must be processed for exactly p_q time (p_q is called the *processing time* of q). For a job q, we denote by r_q its release time and upon the job release time the algorithm gets the job estimated processing time $\tilde{p_q}$; the actual processing time of the job is unknown and revealed only at the job completion time c_q.

We define $\mu_1 = \max_j \frac{p_j}{\tilde{p_j}}$ as the maximum underestimation factor of a job in the input. Similarly, also define $\mu_2 = \max_j \frac{\tilde{p_j}}{p_j}$ as the maximum overestimation factor of a job in the input. Hence, for any job q:

$$\frac{\tilde{p_q}}{\mu_2} \leq p_q \leq \tilde{p_q} \cdot \mu_1$$

Finally, we define the distortion parameter $\mu := \mu_1 \cdot \mu_2$. It is natural to assume that $\mu_1, \mu_2 \geq 1$, although it is not required for our proofs.

Let P be the ratio between the maximal job estimated size and the minimal job estimated size in the input, $P = \frac{\max_q \tilde{p_q}}{\min_q \tilde{p_q}}$.

Note that since P is a function of the input the algorithm has no prior knowledge on P. For the same reason, unless the algorithm is not distortion oblivious it also has no knowledge on the parameters μ_1, μ_2 and μ.

Objectives. In our paper we discuss 2 objectives: total flow time and total stretch. The total flow time objective is one of the most basic performance measures in multiprocessor scheduling problems, and is defined as the overall time the jobs are spending in the system. This includes the delay of waiting for processing as well as the actual processing time. Formally, using the defined notations the total flow time objective is:

$$F = \sum_q c_q - r_q$$

Another natural objective is the total stretch objective. The total stretch objective is defined as follows:

$$W = \sum_q \frac{c_q - r_q}{p_q}.$$

To refer to the objective of an algorithm A, we use the superscript A, e.g., F^A or W^A. Unlike the classic setting, in scheduling with estimates the total stretch objective is *not* a special case of the total weighted time objective since the algorithm does not know the actual processing times of the alive jobs and therefore does not know their weights.

Throughout the paper, we often use the following definition of a job class:

▶ **Definition 1** (job class). *We define the* class *of a job q, denoted ℓ_q, to be the unique integer i such that $\tilde{p}_q \in [2^i, 2^{i+1})$.*

This definition refers to the *estimated* processing times provided in the input (rather than actual processing times, or remaining processing times). In particular, the class of a job does not change over time and the definition does not depend on the algorithm (either it is an algorithm or the optimal solution the job classes are defined the same).

Notations. In the paper we use notations that have been defined by previous work:

- $V^A(t)$ is the remaining time of alive jobs for algorithm A at time t
- $\delta^A(t)$ is the number of alive jobs for algorithm A at time t.
- $\gamma^A(t)$ is the number of non-idle machines for algorithm A at time t.

Where the algorithm is clear from context, we sometimes omit the superscript for these definitions. In addition, for a function f (e.g., V or δ) we define the following notations:

- $\Delta f(t) = f^A(t) - f^{OPT}(t)$, the difference at f value between the algorithm A and the optimal solution at time t.
- $f_{=k}(t)$ is the value of f at time t for class k.
- $f_{\leq k}(t)$ is the value of f at time t over all jobs of class at most k. Similarly, $f_{\geq k}(t)$ is the value of f at time t over all jobs of class at least k

We also use * to denote the optimal solution. For example, $V^*(t)$ is the remaining time of alive jobs for the optimal solution at time t.

3 Minimizing Flow Time

In this section, we describe and analyze a distortion-oblivious algorithm for minimizing total flow time for the non-migrative setting with competitive ratio of $O(\mu \log P)$ against even an migrative optimal algorithm. Note that when μ is a constant this is the best known upper bound.

3.1 Algorithm for the Non-Migrative Setting

The algorithm maintains a pool of jobs that are released and have not been processed at all. In addition, it maintains at every machine a stack of alive jobs that have already been processed by that machine. At any time, each machine processes the job at the top of its stack. The algorithm handles two events: job release and job completion.

At job release, the algorithm checks whether there exists a machine that is empty or currently processing a job of higher class. If such a machine exists, the algorithm places the job at the top of that machine's stack. Otherwise, the job is added to the pool.

Upon job completion, the algorithm pops the completed job from its associated machine stack. Then, it observes the job q' currently at the top of that stack. If the lowest-class job q'' in the pool has a lower class than q' (or if q' does not exist) then q'' is moved to the top of the stack of that machine.

The online algorithm is given as Algorithm 1.

Algorithm 1 Distortion Oblivious Non Migrative Scheduling Algorithm.

```
Event Function UponJobRelease(q)
    if Exists an idle machine or a machine that currently processes a job of class
     higher than ℓ_q then
        Insert q to the top of that machine stack.
    else
        Insert q to the pool.
Event Function UponJobCompletion(q)
    Denote by m_j the machine q was processed on.
    Pop q from the stack of m_j, and let q' be the next job in that stack.
    if the job with lowest class in the pool q'' has class strictly less than that of q' or
     q' does not exist then
        Remove q'' from the pool and insert it to the top of the stack of m_j.
```

The following theorem (Theorem 2) shows that Algorithm 1 has a bounded competitive ratio against an optimal offline solution for the input; in Appendix A.2, we show that Theorem 2 is in fact tight for $m \geq 2$.

▶ **Theorem 2.** *Algorithm 1 is $O(\mu \log P)$-competitive for inputs with distortion μ.*

3.2 Proof of Theorem 2

A useful and classic concept that we use in the proof, is called *local competitiveness*. An algorithm is locally competitive if at every point in time t, the number of living jobs in the algorithm at t is at most some factor from that of the optimal solution at t. A classic observation is that a locally c-competitive algorithm is in particular (globally) c-competitive; this observation results simply from integrating over time. Observation 3 states this formally.

▶ **Observation 3.** *If for an input I it holds that at every time t we have $\delta^A(t) \leq c(I) \cdot \delta^*(t)$ then the algorithm A is $c(I)$-competitive with respect to total flow time on input I.*

Note that at any time every machine stack has at most one job of each class. There are $\lceil \log P \rceil$ classes, therefore at any time the number of partial jobs is at most $m \log P$. Also note that if at time t there is an idle machine (i.e., $\gamma(t) < m$) then the pool is empty and the the total number of alive jobs of the algorithm at time t is $\delta^{ALG}(t) \leq m \log P$.

We define T to be the set of times t that $\gamma^{ALG}(t) = m$; that is, the set of times in which all machines are busy.

We start by proving the following lemma, that for every k bounds the total remaining processing time difference on jobs of class at most k between the algorithm and the optimal solution.

▶ **Lemma 4.** *If $t \in T$ then $\Delta_{\leq k} V(t) \leq m\mu_1 \cdot 2^{k+2}$.*

Proof. Let t_0 be the earliest time such that $[t_0, t) \subset T$. We define $t_k \in [t_0, t)$ as the latest time in $[t_0, t)$ that the algorithm has processed a job of class greater than k (if no such job was processed during this interval then $t_k = t_0$). At time t_k we know that there are no pending jobs of class at most k in the pool. Therefore, all jobs of class at most k are already assigned to a machine. Since every machine process at most one job of any class, there are at most m jobs of any class at most k in the system. The worst case actual processing time of a job of class i is $m\mu_1 2^{i+1}$, thus

$$\Delta_{\leq k} V(t_k) \leq \sum_{i=1}^{k} m\mu_1 2^{i+1} \leq m\mu_1 2^{k+2}.$$

In $[t_k, t) \subset T$ we processed only jobs of class at most k, implies that

$$\Delta_{\leq k} V(t) \leq \Delta_{\leq k} V(t_k) \leq m\mu_1 2^{k+2}.$$

Note that arrival of new jobs adds to both V^* and V^{ALG} the same amount and therefore does not affect the proof. ◀

▶ **Lemma 5.** *If $t \in T$ then $\delta^{ALG}(t) \leq (\mu + 3)\gamma^{ALG}(t) \log P + 2\mu\delta^*(t)$*

Proof.

$$\begin{aligned}
\delta^{ALG}(t) &= \sum_{i=k_{min}}^{k_{max}} \delta^{ALG}_{=i}(t) \\
&\leq \sum_{i=k_{min}}^{k_{max}} \left(\frac{V^{ALG}_{=i}(t)}{2^i/\mu_2} + m \right) \\
&\leq \sum_{i=k_{min}}^{k_{max}} \frac{V^*_{=i}(t) + \Delta V_{=i}(t)}{2^i/\mu_2} + m \log P \\
&\leq \sum_{i=k_{min}}^{k_{max}} \frac{\delta^*_{=i}(t)\mu_1 2^{i+1}}{2^i/\mu_2} + \sum_{i=k_{min}}^{k_{max}} \frac{\Delta V_{\leq i}(t) - \Delta V_{\leq i-1}(t)}{2^i/\mu_2} + \gamma^{ALG}(t) \log P \\
&\leq 2\mu\delta^*(t) + \sum_{i=k_{min}}^{k_{max}} \frac{\Delta V_{\leq i}(t)}{2^{i+1}/\mu_2} + \frac{\Delta V_{\leq k_{max}}(t)}{2^{k_{max}+1}/\mu_2} + \gamma^{ALG}(t) \log P \\
&\leq 2\mu\delta^*(t) + \gamma^{ALG}(t)\mu \log P + 2\mu\gamma^{ALG}(t) + \gamma^{ALG}(t) \log P \\
&\leq (\mu + 3)\gamma^{ALG}(t) \log P + 2\mu\delta^*(t)
\end{aligned}$$

where the first inequality is since there are at most m partial jobs at each class. The third inequality is since $t \in T$ thus $\gamma^{ALG}(t)$ equals m. The fourth inequality is since $\delta^*(t) = \sum_{i=k_{min}}^{k_{max}} \delta^*(t)$. The fifth inequality is applying Lemma 4. ◀

Now we turn to prove Theorem 2:

Proof of Theorem 2. Using the analysis above, we can now bound the total flow time of the algorithm.

$$\begin{aligned}
F^{ALG} &= \int_t \delta^{ALG}(t)dt \\
&= \int_{t\in T} \delta^{ALG}(t)dt + \int_{t\notin T} \delta^{ALG}(t)dt \\
&\leq \int_{t\in T} (\mu+3)\gamma^{ALG}(t)\log(P) + 2\mu\delta^*(t)dt + \int_{t\notin T} \gamma^{ALG}(t)\log(P)dt \\
&\leq (\mu+3)\log P \int_t \gamma^{ALG}(t)dt + 2\mu \int_t \delta^*(t)dt \\
&\leq (\mu+3)\log P \cdot F^* + 2\mu \cdot F^* \\
&= O(\mu \log P)F^*
\end{aligned}$$

where the first inequality follows by applying Lemma 5. The last inequality follows since $F^* = \int_t \delta^*(t)dt$ and $\int_t \gamma^{ALG}(t)dt \leq F^*$. ◀

4 Minimizing Total Stretch

In this section, we study the distortion-oblivious, non-migrative Algorithm 1 for minimizing total stretch. We show that Algorithm 1 achieves the best possible competitive ratio for non-migrative algorithms. We show that Algorithm 1 competitive ratio is $O(\mu^2)$ against even an migrative optimal algorithm (Theorem 6). In addition, we also prove a matching lower bound of $\Omega(\mu^2)$ even for the migrative case. Those results concludes that Algorithm 1 is optimal (Theorem 11) for minimizing the total stretch objective in the online setting.

The weight of a job q is inversely proportional to q's (actual) processing time; we denote this weight by $w_q := \frac{1}{p_q}$. Similarly, for a set of jobs Q we define w_Q as the sum of the weights $w_Q := \sum_{q\in Q} w_q$.

In addition, we define $W^A(t)$ to be the total weight of the currently-alive jobs of an algorithm A at time t (and omit A whenever the algorithm is clear from context). We also use a subscript predicate to restrict this notation to specific classes; for example, $W_{=i}(t)$ is the total weight of living jobs of class exactly i in the algorithm at time t.

4.1 Non-Migrative $O(\mu^2)$ Scheduling on Parallel Machines

In this subsection, we prove the following theorem.

▶ **Theorem 6.** *Algorithm 1 is $O(\mu^2)$-competitive for inputs with distortion μ to minimize total stretch.*

Since stretch scheduling involves weights, here local competitiveness means that at every time t, the total living weight in the algorithm is bounded by the total living weight in the optimal solution. Through integration over time, it is easy to see that local-competitiveness implies competitiveness, as stated in Observation 7.

▶ **Observation 7.** *If for any input I, at any time t, $W^A(t) \leq c(I) \cdot W^*(t)$ then the algorithm A is $c(I)$-competitive with respect to total stretch.*

We start with the following notation:

- Let $c_j(t)$ be the currently processed job of machine j at time t (i.e., the job at the top of machine j stack).
- Let $R(t) = \{c_j(t) | j \in [m]\}$ is the set of jobs being processed at time t.
- Let $S_j(t)$ denote the the set of alive jobs that are on machine j stack, excluding the currently processed job $c_j(t)$. Let $S(t) = \bigcup_j S_j(t)$ the set of alive jobs in one of the machines stack and not currently processed.
- Let $P(t)$ denote the set of alive jobs in the pool at time t.

▶ **Observation 8.** *For any scheduling algorithm A, $\int_t \sum_{x \in R^A(t)} w_x = n$ from the definition of the stretch objective, since every job q is executed exactly p_q time in the algorithm. Therefore, both OPT and Algorithm 1 has total stretch of at least n ($W^* \geq n$).*

We will split the proof into two lemmas. The first lemma (Lemma 9) will show that the total stretch added from jobs being in the machines stacks is bounded by $2n\mu$ and therefore from Observation 8 we get that this part is $O(\mu)$ competitive with OPT. The second lemma (Lemma 10) shows that the rest of the jobs time in the system add at most $4\mu^2 W^* + 12\mu^2 n$ stretch, which is $O(\mu^2)$ competitive with OPT. From this two lemmas, Theorem 6 follows.

▶ **Lemma 9.** $\int_t \sum_{x \in S(t)} w_x \leq 2n\mu$.

Proof. At every machine stack, there is at most one job of each class. Every job $x \in S_j(t)$ is of class strictly greater than of $c_j(t)$.

$$w_{S_j(t)} = \sum_{x \in S_j(t)} w_x \leq \sum_{i=\ell_{c_j(t)}}^{k_{max}} \frac{\mu_2}{2^i} \leq \frac{\mu_2}{2^{\ell_{c_j(t)}}}.$$

The weight of the currently running job of machine j is $w_{c_j(t)} \geq \frac{1}{2^{\ell_{c_j(t)}+1}\mu_1}$. Hence,

$$w_{S_j(t)} \leq 2\mu w_{c_j(t)}.$$

Therefore, $w_{S(t)} \leq 2\mu \cdot w_{R(t)}$ and since $\int_t w_{R(t)} = n$ we get $w_{S(t)} \leq 2\mu n$ that concludes the proof. ◀

Let $V^P_{=i}(t)$ denote the volume of jobs in the pool of class i, $V^P_{\leq i}(t) = \sum_{z \leq i} V^P_{=z}(t)$ and $\Delta V^P_{\leq i}(t) = V^P_{\leq i}(t) - V^*_{\leq i}(t)$. Let $\rho(t)$ denote $\max_{x \in R(t)} \ell_x$, the maximal class of a running job at time t. From algorithm definition, all jobs in the pool are of class at least $\rho(t)$, therefore $V^P_{<\rho(t)}(t) = 0$. We also define $\alpha(t)$ to be the largest class of an alive job. We now turn to prove the lemma that bounds the weight of jobs in the pool.

▶ **Lemma 10.** $\sum_{x \in P(t)} w_x \leq 6\mu^2 W^*(t) + 12\mu^2 \sum_{x \in R(t)} w_x$.

Proof. Note that in case $\gamma(t) < m$, then the pool is empty and the lemma is trivially true; since the left hand side is 0. Denote by C_i the set of jobs of class i. The number of jobs in $C_i \cap P(t)$ is upper bounded by $\frac{\mu_2 V^P_{=i}(t)}{2^i}$ since the jobs in the pool aren't processed yet. Each job has a weight at most $\frac{\mu_2}{2^i}$. Hence we get that

$$\begin{aligned}\sum_{i=\rho(t)}^{\alpha(t)} \sum_{x \in C_i \cap P(t)} w_x \leq & \sum_{i=\rho(t)}^{\alpha(t)} \mu_2 \frac{V^P_{=i}(t)}{2^i} \cdot \frac{\mu_2}{2^i} \\ =& \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^P_{=i}(t)}{2^i \cdot 2^i} \\ \leq & \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t) + \Delta V^P_{=i}(t)}{2^i \cdot 2^i} \\ \leq & \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)}{2^i \cdot 2^i} + \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i \cdot 2^i}.\end{aligned}$$

First, we bound $\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)}{2^i \cdot 2^i}$. At time t, OPT has at least $\frac{V^*_{=i}(t)}{2^{i+1}\mu_1}$ jobs of class i with weight at least $\frac{1}{2^{i+1}\mu_1}$. Therefore:

$$W^*_{=i}(t) \geq \frac{V^*_{=i}(t)}{2^{i+1}\mu_1} \cdot \frac{1}{2^{i+1}\mu_1}$$

which is equivalent to

$$\mu_2^2 \cdot \frac{V^*_{=i}(t)}{2^i \cdot 2^i} \leq 4\mu^2 W^*_{=i}(t)$$

and concludes that

$$\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)}{2^i \cdot 2^i} \leq 4\mu^2 W^*(t).$$

It remains to show that $\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i \cdot 2^i} \leq 6\mu^2 \sum_{x \in R(t)} w_x + 2\mu^2 W^*(t)$:

$$\begin{aligned}\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i \cdot 2^i} \leq & \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t) - \Delta V^P_{\leq i-1}(t)}{2^i \cdot 2^i} \\ =& \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)}{2^{i+1} \cdot 2^i} + \mu_2^2 \frac{\Delta V_{\leq \alpha(t)}(t)}{2^{\alpha(t)+1} \cdot 2^{\alpha(t)}} - \mu_2^2 \frac{\Delta V^P_{\leq \rho(t)-1}(t)}{2^{\rho(t)+1} \cdot 2^{\rho(t)}} \\ =& \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)}{2^{i+1} \cdot 2^i} + \mu_2^2 \frac{\Delta V_{\leq \alpha(t)}(t)}{2^{\alpha(t)+1} \cdot 2^{\alpha(t)}} + \mu_2^2 \frac{V^*_{\leq \rho(t)-1}(t)}{2^{\rho(t)+1} \cdot 2^{\rho(t)}} \\ \leq & \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)}{2^{i+1} \cdot 2^i} + \mu_2^2 \frac{\Delta V_{\leq \alpha(t)}(t)}{2^{\alpha(t)+1} \cdot 2^{\alpha(t)}} + \mu_2^2 \frac{W^*_{\leq \rho(t)-1}(t) \cdot \mu_1^2 \cdot 2^{2\rho(t)+2}}{2^{\rho(t)+1} \cdot 2^{\rho(t)}} \\ \leq & \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{(m-1)\mu_1}{2^i} + \mu_2^2 \frac{4(m-1)\mu_1}{2^{\alpha(t)}} + 2\mu^2 W^*(t) \\ \leq & 6\mu_2^2 \mu_1 (m-1) \cdot \frac{1}{2^{\rho(t)}} + 2\mu^2 W^*(t) \\ \leq & 6m\mu^2 \sum_{x \in R(t)} w_x + 2\mu^2 W^*(t)\end{aligned}$$

where the second inequality is since $\delta^*_{\leq\rho(t)}(t) \geq \frac{V^*_{\leq\rho(t)}}{\mu_1 \cdot 2^{\rho(t)+1}}$ and the minimal weight of a job of class at most $\rho(t)$ is $\frac{1}{\mu_1 \cdot 2^{\rho(t)+1}}$. The first two terms of the third inequality are according to Lemma 4 of previous section (and the fact that $\Delta V^P_{\leq i}(t) \leq \Delta V_{\leq i}(t)$). The right most term of the third inequality is since $W^*_{\leq\rho(t)}(t) \leq W^*(t)$. The last inequality is since for every $x \in R(t)$, $w_x \geq \frac{1}{2^{\rho(t)}\mu_1}$ we get that $6\mu_2^2\mu_1(m-1)\frac{1}{2^{\rho(t)}} \leq 6m\mu^2 \sum_{x\in R(t)} w_x$. We showed that $\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i \cdot 2^i} \leq 6\mu^2 \sum_{x\in R(t)} w_x + 2\mu^2 W^*(t)$ which concludes the proof. ◀

Now we can prove the algorithm is $O(\mu^2)$ competitive:

Proof of Theorem 6.

$$\begin{aligned}
\int_t \sum_{x\in Jobs(t)} w_x &= \int_t \sum_{x\in S(t)} w_x + \int_t \sum_{x\in P(t)} w_x \\
&\leq 3n\mu + \int_t 6\mu^2 W^*(t) + 6\mu^2 \sum_{x\in R(t)} w_x \\
&\leq 3n\mu + 6\mu^2 W^* + 6\mu^2 n \\
&\leq 15\mu^2 W^*
\end{aligned}$$

where the second inequality is since $\int_t \sum_{x\in R(t)} w_x = n$ (Observation 8) and the last inequality is since $n \leq W^*$ (Observation 8). ◀

4.2 Total Stretch Lower Bound

We show $\Omega(\mu^2)$ lower bound for online distortion oblivious scheduling on multiple machines even if job migrations is allowed. This lower bound proves that the online algorithm presented in this section is optimal for total stretch minimization.

▶ **Theorem 11.** *For every number of parallel machines m and for every constant distortion μ, every deterministic online algorithm ALG to minimize total stretch time has a competitive ratio of $\Omega(\mu^2)$.*

Proof. Without loss of generality the algorithm is non-idle; since every algorithm with idle-time can be transformed to a non-idle algorithm by simply processing a arbitrary job during the idle time. Also assume that $\mu > 2$. Let $t = \frac{\mu^5}{2}$. We analyze the local competitive ratio of the algorithm at time t and then extend the result to (global) competitive ratio.

At time 0 release $m \cdot \mu^4$ jobs of estimated size 1 (their actual size is in $[1, \mu]$). For every job q denote by x_q the amount of time ALG processes job q until time t. For every job q, define the actual processing time of q to be:

$$p_j = \min(\max(1, x_j + \epsilon), \mu)$$

where $\epsilon = \frac{1}{\mu}$. Note that the sum of the jobs processing time is at least $m \cdot t$, since we assume the algorithm is non-idle and there is enough volume to be processed. Denote:

1. D is the set of jobs that $p_j = \mu$.
2. F is the set of jobs that $x_j \leq 1 - \epsilon$. In particular for all $j \in F$, $p_j = 1$.
3. P is the set of all the other jobs.

The above implies that:

$$|D| + |F| + |P| = m \cdot \mu^4.$$

OPT processes the jobs with *Shortest Time First*. Therefore, OPT finishes all the jobs of size at most $\frac{\mu}{2}$ (in particular, all the jobs in F). Note that the total remaining volume for all jobs at t in the algorithm is at most $\epsilon(|P|+|D|)+|F|$. Since the optimal solution is non-idle, this is also an upper bound for the total remaining volume in the optimal solution at t. However, all pending jobs in the optimal solution at t (with the exception of at most m jobs currently being processed) have a remaining processing time of at most $\frac{\mu}{2}$. Thus, the total number of pending jobs in the optimal solution at t is at most $m + \frac{2}{\mu} \cdot (\epsilon(|P|+|D|)+|F|)$, which is $O(m \cdot \mu^2 + \frac{|F|}{\mu})$. Those jobs are in P and D and of size at least $\frac{\mu}{2}$. Therefore the total weight of the optimal solution at time t is:

$$W^*(t) = O\left(m \cdot \mu + \frac{|F|}{\mu^2}\right).$$

We turn to analyze the weight of the algorithm at time t. Note that $|D| \leq \frac{m \cdot \mu^4}{2-2\frac{\epsilon}{\mu}} \leq \frac{m \cdot \mu^4}{1.5}$ since otherwise there must be a job $q \in D$ that $x_q < \mu - \epsilon$. The algorithm ALG, remains with the jobs in the sets P and F. Their combined size is $|P|+|F| = m \cdot \mu^4 - |D| \geq \frac{m \cdot \mu^4}{3}$. Since the algorithm ALG does not complete any of the jobs in F, it remains with weight of at least $|F|$ since the weight each job in F is 1. Therefore:

$$W^{ALG}(t) \geq |F|.$$

Analyzing in different way, the algorithm remains with at least $\frac{m \cdot \mu^4}{3}$ jobs (of the sets P and F) where the minimal weight of a job is $\frac{1}{\mu}$. Therefore, the weight of the algorithm ALG at time t is:

$$W^{ALG}(t) \geq \Omega\left(\frac{m \cdot \mu^4}{3} \cdot \frac{1}{\mu}\right) = \Omega(m \cdot \mu^3).$$

That means:

$$W^{ALG}(t) = \Omega(m \cdot \mu^3 + |F|).$$

Combining all, the local competitiveness at time t is:

$$\Omega\left(\frac{m \cdot \mu^3 + |F|}{m \cdot \mu + \frac{|F|}{\mu^2}}\right)$$

which is $\Omega(\mu^2)$ local competitive ratio. To finish the proof, we use the standard "bombardment" technique: at time t, we start releasing m jobs of size $\epsilon = \frac{1}{\mu}$ every ϵ time. The total weight of the released jobs in each interval is $m\mu$. This is easily seen to extend the $\Omega(\mu^2)$ lower bound from local competitiveness to general (global) competitiveness. ◀

5 Impossibility of online Immediate-Dispatching algorithms for Robust Scheduling

In this part, we will show lower bounds on immediate dispatching algorithms in the distortion-oblivious setting. We show that for both total flow time and total stretch objectives, there are polynomial lower bounds in terms of the number of jobs n for arbitrarily close to 1 values of μ and P, even if randomization is allowed. We first prove the lower bound version for deterministic algorithms (Theorem 12). Then we extend the proof to allow algorithms that use randomness (Theorem 13).

▶ **Theorem 12.** *For $m > 1$, every online immediate dispatching deterministic algorithm ALG, for every constant distortion factor μ, has a competitive ratio of $\Omega\left(\sqrt{\frac{n}{m}}\right)$ with respect to the total flow time objective.*

Proof. Theorem 12. At the beginning (i.e., time 0) we release k jobs of estimated size 1, the value of k will be determined later. Let x_i and be the number of jobs assigned to machine i by the algorithm ALG.

$$\sum_{i\in[m]} x_i = k.$$

Let j be the machine with the largest number of assigned jobs by the algorithm (i.e., $j = argmax_{i\in[m]} x_i$). For jobs assigned to machine j we set their actual size to be μ, and for the rest of the we set their actual size to 1.

The optimal solution, completes all the released jobs by time $t = \frac{x_j\mu+(m-1)x_j}{m}$, with *round robin* algorithm. The total flow time of the jobs until time t is $O(\frac{k^2}{m}\cdot\mu)$.

By that time, the algorithm remains with volume of at least $x_j\mu - t$ of remaining jobs assigned to machine j. Plug in the value of t and the remaining volume is:

$$x_j\mu - t = \Omega\left(\frac{k}{m}\cdot(\mu-1)\right).$$

From time t until time $t+(k/m)^2\cdot\mu$ we release in intervals of μ (i.e., $t, t+\mu, t+2\mu, t+3\mu, ...$) m jobs of actual processing time μ and estimated processing time 1. In total, during the construction we release $k + m\cdot(k/m)^2$ jobs.

At each interval, the optimal solution assigns the m released jobs to the m machines and completes them by the end of the interval. Thus the optimal solution total flow time is

$$F^* = m\cdot(k/m)^2\cdot\mu + O\left(\frac{k^2}{m}\cdot\mu\right) = O\left(\frac{k^2}{m}\cdot\mu\right).$$

The algorithm at time t remains with $\Omega\left(\frac{k}{m}(\mu-1)\right)$ volume (of jobs assigned to machine j), therefore, since the maximum processing time of a job is μ, at any tine between t and $t+(k/m)^2\cdot\mu$, the algorithm has at least $\Omega\left(\frac{k}{m}(1-\frac{1}{\mu})\right)$ alive jobs. Hence, the algorithm ALG total flow time is

$$F^{ALG} \geq (k/m)^2\cdot\mu\cdot\left(\frac{k}{m}\cdot\left(1-\frac{1}{\mu}\right)+m\right) + \Omega\left(\frac{k^2}{m}\cdot\mu\right).$$

Therefore the algorithm ALG is $\Omega(\frac{k}{m}(1-1/\mu))$-competitive with the optimal solution. The number of released jobs during execution is $n = k + m\cdot(k/m)^2 = O\left(\frac{k^2}{m}\right)$ and therefore, in other terms the algorithm ALG competitive ratio is $\Omega\left(\sqrt{\frac{n}{m}}(1-1/\mu)\right)$, which for a constant μ is $\Omega\left(\sqrt{\frac{n}{m}}\right)$. ◀

Now we prove the second theorem, Theorem 13 with the same technique we used at the proof of Theorem 12.

▶ **Theorem 13.** *For $m > 1$, every online immediate dispatching algorithm ALG that can use randomness, for every constant distortion factor μ, has a competitive ratio of $\Omega\left(\frac{n^{1/4}}{\sqrt{m}}\right)$ with respect to the total flow time objective.*

Proof. Theorem 13. We use Yao's principle and describe a lower bound for deterministic algorithms on a given distribution over the input. This yields a lower bound for randomized online algorithms.

Release k jobs of estimated size 1 at the beginning, $k/2$ of them with actual size 1 and the rest with actual size μ. The algorithm ALG assigns the jobs to the machines. Let x_i be the number of jobs that the algorithm assigned to machine i. Since the algorithm does not know the actual sizes at the assignment time, in expectation, every machine has $\frac{x_i}{2}$ jobs of actual size 1 and $\frac{x_i}{2}$ jobs of actual size μ.

We denote by z_i the number of jobs of actual size μ that are assigned to machine i by the algorithm. Similarly, we denote by y_i the number of jobs of actual size 1 that are assigned to machine i by the algorithm. Note that for every machine i, $z_i + y_i = x_i$.

The optimal solution, can perform *round robin* algorithm and complete all the released jobs at time $t = \frac{k(\mu+1)}{2m}$. The total flow time of the jobs until time t is $O\left(\frac{k^2}{m} \cdot \mu\right)$.

Let j be the machine with the largest number of assigned jobs by the algorithm (i.e., $j = argmax_{i\in[m]} x_i$). As before, $x_j \geq \frac{k}{m}$. The random variable z_j is a binomial random variable, $z_j \sim B(x_j, 0.5)$. Let G be the event that $z_j \geq \frac{k}{m} + \sqrt{\frac{k}{m}}$, note that the event G happens with probability at least $\frac{1}{4}$ (i.e., $P(G) \geq \frac{1}{4}$). We analyze only the case that G happens, in any other case we let the total flow time of the algorithm to be 0.

If G happens, then the algorithm ALG at time t is left with $\Omega\left(\sqrt{\frac{k}{m}} \cdot (\mu - 1)\right)$ of volume of jobs that assigned to machine j.

Repeating the last part of the previous proof, from time t until time $t + (k/m)^2 \cdot \mu$ we release in intervals of μ (i.e., $t, t+\mu, t+2\mu, t+3\mu, ...$) m jobs of actual processing time μ and estimated processing time 1. In total, during the construction we release $k + m \cdot (k/m)^2$ jobs.

At the beginning of each interval, the optimal solution assigns the m released jobs to the m machines and completes them at the end of each interval. Thus the optimal solution total flow time is

$$F^* = m \cdot (k/m)^2 \cdot \mu + O\left(\frac{k^2}{m} \cdot \mu\right) = O\left(\frac{k^2}{m} \cdot \mu\right).$$

At the time between t to $t + (k/m)^2 \cdot \mu$, ALG has at least $\frac{k}{m} \cdot (1 - 1/\mu) + m$ alive jobs. Hence, the algorithm ALG total flow time is

$$E[F^{ALG}] \geq P(G) \cdot \left((k/m)^2 \cdot \mu \cdot \left(\sqrt{\frac{k}{m}} \cdot (1 - 1/\mu) + m\right) + \Omega\left(\frac{k^2}{m} \cdot \mu\right)\right).$$

Therefore the algorithm ALG is $\Omega\left(\sqrt{\frac{k}{m}}(1 - 1/\mu)\right)$-competitive with the optimal solution. The number of released jobs during execution is $n = k + m \cdot (k/m)^2$ and therefore, the algorithm ALG competitive ratio is $\Omega(\frac{n^{1/4}}{\sqrt{m}}(1 - 1/\mu))$. ◀

The lower bound also applies for stretch since all the jobs are in the range of $[1, \mu]$, therefore considering the weights of the jobs can only decrease the competitive ratio by factor of μ, which is negligible related to the number of jobs n.

6 Discussion and Open Problems

In this paper, we present the first distortion-oblivious algorithms for total stretch, which have optimal competitive ratio. This provides an optimal deterministic distortion-obliviousness algorithm for the total stretch objective on both single and multiple machines. We also present

nearly-optimal, distortion-oblivious algorithms for total flow time on multiple machines. We also show that with immediate dispatching no algorithm with sub-polynomial competitive ratio exists. It would be interesting to close the gap between the presented algorithms for flow time, that achieve $O(\mu \log P)$-competitiveness, to the previously known lower bound of $\Omega(\mu + \log P)$. Another direction of future work would be randomized distortion-oblivious algorithms for minimizing total stretch.

References

1 Algorithms with predictions. https://algorithms-with-predictions.github.io.

2 Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 11–18, New York, NY, USA, 2003. ACM. doi:10.1145/777412.777415.

3 Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. *SIAM Journal on Computing*, 31(5):1370–1382, 2002. doi:10.1137/S009753970037446X.

4 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1070–1080. ACM, 2021. doi:10.1145/3406325.3451023.

5 Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 252–274. SIAM, 2022. doi:10.1137/1.9781611977073.13.

6 Eric Balkanski, Tingting Ou, Clifford Stein, and Hao-Ting Wei. Scheduling with speed predictions, 2022. doi:10.48550/ARXIV.2205.01247.

7 Nikhil Bansal, Kedar Dhamdhere, Jochen Könemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004. doi:10.1007/s00453-004-1115-0.

8 Luca Becchetti and Stefano Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 94–103, 2001.

9 Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 84–93, 2001. doi:10.1145/380752.380778.

10 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *Journal of the ACM (JACM)*, 65(1):1–33, 2017.

11 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 531–540. IEEE, 2014.

12 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, pages 285–294, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3409964.3461790.

13 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, New Orleans, LA, USA, January 5 - 8, 2020.*, 2020.

14 Stefano Leonardi. *A Simpler Proof of Preemptive Total Flow Time Approximation on Parallel Machines*, pages 203–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11671541_7.

15 Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 110–119, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258562.

16 Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. *arXiv*, 2022. doi:10.48550/arXiv.2202.10199.

17 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3302–3311, 2018. URL: http://proceedings.mlr.press/v80/lykouris18a.html.

18 Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1858–1866, 2017. URL: http://papers.nips.cc/paper/6782-revenue-optimization-with-approximate-bid-predictions.

19 Michael Mitzenmacher. Scheduling with Predictions and the Price of Misprediction. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2020.14.

20 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint*, 2020. doi:10.48550/arXiv.2006.09123.

21 Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994.

22 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

23 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi:10.1002/nav.3800030106.

A Total Flow Time Appendix

A.1 LCF algorithm analysis for total flow time

In this setting, we use the LCF (Lowest Class First) algorithm. The algorithm maintains a list of jobs L; first sorted by jobs' class and then by release time. At any time the algorithm processes the m first jobs of the list L.

▶ **Theorem 14.** *LCF is $O(\mu \log P)$ competitive for inputs with distortion μ.*

▶ **Theorem 15.** *LCF is $O(\mu \log \mu + \mu \log \frac{n}{m})$ competitive for inputs with distortion μ.*

▶ **Corollary 16.** *For every μ, LCF is $O(\min(\mu \log P, \mu \log \mu + \mu \log \frac{n}{m}))$-competitive for inputs with distortion μ.*

Proof of Theorem 14. We skip most of the proof since it is almost similar to the proof of Theorem 2. The proof is based on the following two lemmas, which their proof is omitted.

▶ **Lemma 17.** *If $t \in T$ then $\Delta_{\leq k} V(t) \leq m\mu_1 2^{k+1}$.*

▶ **Lemma 18.** *If $t \in T$ then $\delta_{\geq k_1 \leq k_2}(t)^{LCF} \leq m(\mu+3)(k_2 - k_1 + 2) + 2\delta^*_{\leq k_2}(t)$.*

Similar steps as in the proof of Theorem 2 and plugging in Lemma 18 concludes the proof. ◀

Proof of Theorem 15. Lets define the *group* of a job q as the unique integer i such that $p_q \in [2^i, 2^{i+1})$, differently from the class of the job, the *group* is defined over the actual processing time rather than the estimated.

Let $\bar{k}$ be the maximum integer k such that for some time $t \in T$, $\delta^{LCF}_{\geq k}(t) \geq m$ (note that $\delta^{LCF}_{\geq k}(t) \geq m$ is the number of alive jobs of *class* at least k, not *group*). If such integer does not exists, then the set T is empty. In addition, for this proof we define k_{min} differently; as the lowest *group* of a job. Let $T_j \subset T$, $j = k_{min}...\bar{k}$, be the set of times that all the machines are busy and the maximal currently processed job *group* is j. Let $T_{\bar{k}+1} \subset T$ be the set of times when all machines are busy and at least one machine is processing a job of *group* higher than $\bar{k}$. Observe that $T_{k_{min}}...T_{\bar{k}+1}$ defines a partition of T . We can write the total flow time of LCF as:

$$\begin{aligned} F^{LCF} &= \int_{t\notin T} \delta^{LCF} dt + \int_{t\in T} \delta^{LCF} dt \\ &= \int_{t\notin T} \gamma^{LCF}(t) dt + \sum_{j=k_{min}}^{\bar{k}+1} \int_{t\in T_j} \delta^{LCF}(t) dt. \end{aligned}$$

Note that $\forall t \in T_j$ we have $\delta_{<j-\lceil \log \mu \rceil}(t) < m$ since the algorithm currently process a job of *group* j. Also note that for any time t we have $\delta_{>\bar{k}}(t) < m$ from the definition of $\bar{k}$. By these two observations we get that for any group j, $\forall t \in T_j$ $\delta^{LCF}(t) \leq 2m + \delta^{LCF}_{\geq j-\lceil \log \mu \rceil, \leq \bar{k}+1}(t)$.

$$F^{LCF} \leq \int_{t\notin T} \gamma^{LCF}(t) dt + \sum_{j=k_{min}}^{\bar{k}+1} \int_{t\in T_j} (2m + \delta^{LCF}_{\geq j-\lceil \log \mu \rceil, \leq \bar{k}+1}(t)) dt.$$

By plugging in Lemma 18:

$$F^{LCF} \leq \int_{t\notin T} \gamma^{LCF}(t) dt + \sum_{j=k_{min}}^{\bar{k}+1} \int_{t\in T_j} (2m + m\mu(\bar{k} - j + 3 + \lceil \log \mu \rceil) + 2\delta^*_{\leq \bar{k}+1}(t)) dt$$

which implies:

$$\begin{aligned} F^{LCF} \leq & \int_{t\notin T} \gamma^{LCF}(t) dt + (3\mu + 2) \int_{t\in T} m dt + \sum_{j=k_{min}}^{\bar{k}+1} m\mu(\bar{k} - j)|T_j| \\ & + \sum_{j=k_{min}}^{\bar{k}+1} |T_j| m\mu \lceil \log \mu \rceil + 2 \int_{t\in T} \delta^*_{\leq \bar{k}+1}(t) dt \end{aligned}$$

where the first two terms $\int_{t\notin T} \gamma^{LCF}(t) dt + (3\mu + 2) \int_{t\in T} m dt$ are at most $(3\mu + 2)F^*$ since $\int_t \gamma^{ALG}(t) \leq F^*$ and for any $t \in T$ we have $\gamma^{ALG}(t) = m$. The fourth term, $\sum_{j=k_{min}}^{\bar{k}+1} |T_j| m\mu \lceil \log \mu \rceil$ equals $\mu \lceil \log \mu \rceil \int_{t\in T} \gamma^{ALG}(t) dt$ since T_j defines a partition of T. Using the fact $\int_{t\in T} \gamma^{ALG}(t) dt \leq F^*$ we have that the fourth term is bounded by $\mu \lceil \log \mu \rceil F^*$. The right most term, $\int_{t\in T} \delta^*_{\leq \bar{k}+1}(t) dt \leq F^*$, from the definition of local competitiveness. In addition, for $j = \bar{k}$ the mid term is negative. Therefore we can write:

$$F^{LCF} \leq (\mu \lceil \log \mu \rceil + 3\mu + 4)F^* + \mu \sum_{j=k_{min}}^{\bar{k}} m(\bar{k} - j)|T_j|.$$

Now we have the following lemma, which its proof is adapted from [14] (lemma 8). Their lemma was proved for SRPT. We show that it also holds for LCF. Note that in our proof, the sets T_j are defined according to the original *group* while in the original proof the sets T_j are defined according to the remaining processing time *group*.

▶ **Lemma 19.** $F(n) = \sum_{j=k_{min}}^{\bar{k}} m(\bar{k}-j)|T_j| = O(F^* \cdot \log \frac{n}{m})$.

Proof of Lemma 19. Let T_j^l be the set of times that machine l processes a job of *group* j. At every time $t \in T_j$ is also a part of m sets $T_{j_l}^l$ where j_l denotes the *group* of the job processed on machine ℓ. Also, at every time $t \in T_j$ the maximal group of a job is j, therefore $\bar{k} - j \leq \bar{k} - j_\ell$ for any machine ℓ and the following inequality follows:

$$F(n) = \sum_{j=k_{min}}^{\bar{k}} m(\bar{k}-j)|T_j| \leq \sum_{k_{min}}^{\bar{k}} \sum_{j \in [m]} (\bar{k}-j)|T_j^l|.$$

Every job of *group* i gives a contribution to the above equation of at most $(\bar{k}-i)2^{i+1}$. Let n_j denote the number of jobs of *group* j, therefore:

$$F(n) \leq \sum_{j=k_{min}}^{\bar{k}} (\bar{k}-j) n_j 2^{j+1}.$$

Note that this is only possible because we consider the *group* of the job, and therefore the maximal processing time of the job is 2^{j+1} rather than $2^{j+1} \cdot \mu_1$. Let $I_i = n_{\bar{k}-i} \cdot 2^{\bar{k}-i}$, for $i = k_{min}, ..., \bar{k}$. Then the sum becomes:

$$F(n) \leq 2 \cdot \sum_{i=0}^{\bar{k}-k_{min}} i \cdot I_i$$

In order to bound the function $F(n)$, we maximize the function subject to two obvious constraints:

$$\sum_{i=0}^{\bar{k}-k_{min}} I_i \leq \sum_q p_q$$

$$\sum_{i=0}^{\bar{k}-k_{min}} 2^i \cdot I_i \leq n \cdot 2^{\bar{k}}.$$

To complete the proof we use the following lemma proved in [14].

▶ **Lemma 20.** *Given a sequence $a_1, a_2, ...$ of non-negative numbers such that $\sum_{i \geq 1} a_i \leq A$ and $\sum_{i \geq 1} 2^i \cdot a_i \leq B$ then $\sum_{i \geq 1} i \cdot a_i \leq \log(\frac{4B}{A}) \cdot A$.*

We use Lemma 20 by setting $a_i = I_i$ for $i = 0, ..., \bar{k} - k_{min}$, $A = \sum_q p_q$ and $B = n \cdot 2^{\bar{k}}$.

$$\sum_{i=0}^{\bar{k}-k_{min}} i \cdot I_i \leq \log(\frac{4n \cdot 2^{\bar{k}}}{\sum_q p_q}) \cdot \sum_q p_q$$

By the definition of $\bar{k}$ and that $\sum_q p_q \leq F^*$ we get that:

$$F(n) \leq \sum_{i=0}^{\bar{k}-k_{min}} i \cdot I_i = O(F^* \cdot \log \frac{n}{m})$$

and that concludes the proof. ◀

Note that Lemma 19 concludes the proof, since:

$$F^{LCF} \leq (\mu \lceil \log \mu \rceil + 3\mu + 4) \cdot F^* + O(\mu \log \frac{n}{m}) \cdot F^* = O(\mu \log \mu + \mu \log \frac{n}{m}) \cdot F^*$$

as required. ◀

A.2 LCF and Algorithm 1 are $\Omega(\mu \log P)$-competitive

We argue that the theorems Theorem 14 and Theorem 2 are tight for $m \geq 2$. We show that by a single construction for both algorithms, that is built using $L = \lceil \log_\mu P \rceil - 1$ phases. We first describe the input construction, prove for the LCF algorithm and then explain that on this input both algorithms act exactly the same and the proof holds for both. For simplicity we assume that m and μ are even. Denote by r_i the time that phase i begins and is defined as follows:

$$P_i = \frac{P}{(2\mu)^i}.$$
$$r_0 = 0 \ ; \ r_i = r_{i-1} + 2P_i$$

For $i = 0, 1, ..., L-1$; at time r_i, we release $\frac{3m}{2}$ jobs of actual processing time P_i, their estimated processing time is also P_i (i.e., with no estimation error). Let $e_i = r_i + \frac{3P_i}{2}$. At time e_i we release $\frac{m\mu}{2}$ jobs of processing time $\frac{P_i}{\mu}$ with estimated time P_i.

The optimal solution can perform *round robin* algorithm and completes all the jobs that has been released at time r_i by time e_i and the jobs released at time e_i it completes by time r_{i+1}. Therefore at time r_L, the optimal solution completes all of the released jobs.

The algorithm LCF will always prefer jobs released at phase $i+1$ over phase i. Therefore, by time e_i LCF completes precisely m jobs that released at time r_i and the remaining $m/2$ jobs are processed precisely for $\frac{P_i}{2}$ time. In addition, LCF also process jobs of previous phases. *Inductively*, we show that for $i \geq 1$, those jobs are with remaining processing time of at least $2P_i$ at time r_i (and therefore also at $r_i + P_i$). Hence, the jobs from previous phases will be left with at least P_i processing time until phase $i+1$. Therefore neither of the remaining jobs at any phase until i is ever finished.

At the time between e_i and r_{i+1}, LCF continues to process the $m/2$ jobs released at time r_i and on the other $m/2$ machines it processes the jobs released by time e_i. LCF completes $\frac{m\mu}{4}$ and remains with $\frac{m\mu}{4}$ with processing time $\frac{P_i}{\mu}$ (unprocessed). Since $\frac{P_i}{\mu} \geq 2P_{i+1}$ this completes the induction.

At every phase, LCF added $\frac{m\mu}{4}$ jobs, that as discussed are not completed, and therefore

$$\delta^{LCF}(r_L) = \Omega(L \cdot m\mu) = \Omega(m\mu \log_\mu(P))$$

while the optimal solution remains with no jobs. To finish the proof, we use the standard "bombardment" technique: at time r_L, we start releasing m unit-jobs. This is easily seen to extend the $\Omega(\mu \log_\mu P)$ lower bound from local competitiveness to general (global) competitiveness.

Note that on this instance, since LCF does not need to do any migration and therefore Algorithm 1 actually behaves the same and the analysis also shows that Algorithm 1 analysis is tight.

Efficiently Reconfiguring a Connected Swarm of Labeled Robots

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany

Peter Kramer
Department of Computer Science, TU Braunschweig, Germany

Christian Rieck
Department of Computer Science, TU Braunschweig, Germany

Christian Scheffer
Faculty of Electrical Engineering and Computer Science, Bochum University of Applied Sciences, Germany

Arne Schmidt
Department of Computer Science, TU Braunschweig, Germany

Abstract

When considering motion planning for a swarm of n labeled robots, we need to rearrange a given start configuration into a desired target configuration via a sequence of parallel, continuous, collision-free robot motions. The objective is to reach the new configuration in a minimum amount of time; an important constraint is to keep the swarm connected at all times. Problems of this type have been considered before, with recent notable results achieving *constant stretch* for not necessarily connected reconfiguration: If mapping the start configuration to the target configuration requires a maximum Manhattan distance of d, the total duration of an overall schedule can be bounded to $\mathcal{O}(d)$, which is optimal up to constant factors. However, constant stretch could only be achieved if *disconnected* reconfiguration is allowed, or for scaled configurations (which arise by increasing all dimensions of a given object by the same multiplicative factor) of *unlabeled* robots.

We resolve these major open problems by (1) establishing a lower bound of $\Omega(\sqrt{n})$ for connected, labeled reconfiguration and, most importantly, by (2) proving that for scaled arrangements, constant stretch for connected reconfiguration can be achieved. In addition, we show that (3) it is NP-hard to decide whether a makespan of 2 can be achieved, while it is possible to check in polynomial time whether a makespan of 1 can be achieved.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Motion path planning

Keywords and phrases Motion planning, parallel motion, bounded stretch, makespan, connectivity, swarm robotics

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.17

Related Version *Full Version*: `https://arxiv.org/abs/2209.11028` [20]

1 Introduction

Motion planning for sets of objects is a theoretical and practical problem of great importance. A typical task arises from relocating a large collection of agents from a given start into a desired goal configuration, while avoiding collisions between objects or with obstacles. Previous work has largely focused on achieving reconfiguration via sequential schedules, where one robot moves at a time; however, reconfiguring *efficiently* requires reaching the target configuration in a timely or energy-efficient manner, with a natural objective of minimizing the time until completion, called *makespan*. Achieving minimum makespan for reconfiguring a swarm of labeled robots was the subject of the 2021 Computational Geometry Challenge [19]; see [8, 25, 37] for successful contributions.

Exploiting parallelism in a robot swarm to achieve an efficient schedule was studied in recent seminal work by Demaine et al. [11]: Under certain conditions, a labeled set of robots can be reconfigured with bounded *stretch*, i.e., there is a collision-free motion plan such that the makespan of the schedule remains within a constant of the lower bound that arises from the maximum distance between origin and destination of individual robots; see also the video by Becker et al. [4] that illustrates these results.

A second important aspect for many applications is *connectivity* of the swarm throughout the reconfiguration, because disconnected pieces may not be able to regain connectivity, and also because of small-scale swarm robots (such as catoms in claytronics [22]), which need connectivity for local motion, electric power and communication; see the video by Bourgeois et al. [5]. Connectivity is not necessarily preserved in the schedules by Demaine et al. [11]. In more recent work, Fekete et al. [18] presented an approach that does achieve constant stretch for *unlabeled* swarms of robots for the class of *scaled* arrangements; such arrangements arise by increasing all dimensions of a given object by the same multiplicative factor and have been considered in previous seminal work on self-assembly, often with unbounded or logarithmic scale factors (along the lines of what has been considered in self-assembly [31]). The method by Fekete et al. [18] relies strongly on the exchangeability of indistinguishable robots, which allows a high flexibility in allocating robots to target configurations, which is not present in labeled reconfiguration.

These results have left two major open problems.

1. Can efficient reconfiguration be achieved in a *connected* manner for a swarm of *labeled* robots in a not necessarily scaled arrangement?
2. Is it possible to achieve constant stretch for connected reconfiguration of scaled arrangements of *labeled* objects?

1.1 Our contributions

We resolve both of these open problems.

1. We show that connected reconfiguration of a swarm of n labeled robots may require a stretch factor of at least $\Omega(\sqrt{n})$.
2. We present a framework for achieving *constant* stretch for connected reconfiguration of scaled arrangements of labeled objects.
3. In addition, we show that it is NP-hard even to decide whether a makespan of 2 for labeled connected reconfiguration can be achieved.

1.2 Related work

Algorithmic efforts for multi-robot coordination date back to the seminal work by Schwartz and Sharir [30] from the 1980s. Efficiently coordinating the motion of many agents arises in a large spectrum of applications, such as air traffic control [9], vehicular traffic networks [17, 29], ground swarm robotics [27, 28], or aerial swarm robotics [7, 36]. In both discrete and geometric variants of the problem, the objects can be *labeled*, *colored* or *unlabeled*. In the *labeled* case, the objects are all distinguishable and each object has its own, uniquely defined target position. In the *colored* case, the objects are partitioned into k groups and each target position can only be covered by an object with the right color; see Solovey and Halperin [32]. In the *unlabeled* case, objects are indistinguishable and target positions can be covered by any object; see Kloder and Hutchinson [23], Turpin et al. [35], Adler et al. [1], and Solovey et al. [34]. On the negative side, Solovey and Halperin [33] prove that the unlabeled multiple-object motion planning problem is PSPACE-hard. Calinescu, Dumitrescu, and Pach [6] consider the

sequential reconfiguration of objects lying on vertices of a graph. They give NP-hardness and inapproximability results for several variants, a 3-approximation algorithm for the unlabeled variant, as well as upper and lower bounds on the number of sequential moves needed.

We already described the work by Demaine et al. [11] for achieving constant stretch for coordinated motion planning, as well as the recent practical CG Challenge 2021 [8, 19, 25, 37]. None of these approaches satisfy the crucial connectivity constraint, which has previously been investigated in terms of decidability and feasibility by Dumitrescu and Pach [14] and Dumitrescu, Suzuki, and Yamashita [16]. Furthermore, these authors have also proposed efficient patterns for fast swarm locomotion in the plane using sequential moves that allow preservation of connectivity [15]. A closely related body of research concerns itself with sequential pivoting moves that require additional space around moving robots, limiting feasibility and reachability of target states, see publications by Akitaya et al. [2, 3].

Very recently, Fekete et al. [18] presented a number of new results for connected, but unlabeled reconfiguration. In addition to complexity results for small makespan, they showed that there is a constant c^* such that for any pair of start and target configurations with a (generalized) *scale* of at least c^*, a schedule with constant stretch can be computed in polynomial time. The involved concept of scale has received considerable attention in self-assembly; achieving constant scale has required special cases or operations. Soloveichik and Winfree [31] showed that the minimal number of distinct tile types necessary to self-assemble a shape, at some scale, can be bounded both above and below in terms of the shape's Kolmogorov complexity, leading to unbounded scale in general. Demaine et al. [13] showed that allowing to destroy tiles can be exploited to achieve a scale that is only bounded by a logarithmic factor, beating the linear bound without such operations. In a setting of recursive, multi-level *staged* assembly with a logarithmic number of stages (i.e., "hands" for handling subassemblies), Demaine et al. [10] achieved logarithmic scale, and constant scale for more constrained classes of polyomino shapes; this was later improved by Demaine et al. [12] to constant scale for a logarithmic number of stages. More recently, Luchsinger et al. [26] employed repulsive forces between tiles to achieve constant scale in two-handed self-assembly.

For further related work see Demaine et al. [11].

1.3 Preliminaries

We consider *robots* at nodes of the (integer) infinite grid $G = (V, E)$, where two nodes are connected if and only if they are in unit distance. A *configuration* is a mapping $C : V \to \{1, \ldots, n, \perp\}$, i.e., each node is mapped injectively to one of the n labeled robots, or to $\perp$ if the node is empty. For a robot ℓ, $C^{-1}(\ell) = (x_\ell, y_\ell)$ refers to its x- and y-coordinate. The configuration C is *connected* if the subgraph $H \subset G$ induced by occupied nodes in C is connected. The *silhouette* of a configuration C is the respective unlabeled configuration, i.e., C without labeling. Unless stated otherwise, we consider labeled connected configurations.

Two configurations *overlap*, if they have at least one occupied position in common. A configuration C is *c-scaled*, if H is the union of $c \times c$ squares of vertices. The *scale* of a configuration C is the maximal c such that C is c-scaled. This corresponds to objects being composed of pixels at a certain resolution; note that this is a generalization of the uniform pixel scaling studied in previous literature (which considers a c-grid-based partition instead of an arbitrary union), so it supersedes that definition and leads to a more general set of results. Two robots are *adjacent* if their positions v_1, v_2 are adjacent, i.e., $\{v_1, v_2\} \in E(H)$.

A robot can move in discrete time steps by changing its location from a grid position v to an adjacent grid position w; denoted by $v \to w$. Two moves $v_1 \to w_1$ and $v_2 \to w_2$ are *collision-free* if $v_1 \neq v_2$ and $w_1 \neq w_2$. Note that a *swap*, i.e., two moves $v_1 \to v_2$ and $v_2 \to v_1$, causes

a collision and is not allowed in our model. A *transformation* between two configurations C_1 and C_2 is a set of collision-free moves $\{v \to w \mid C_1(v) = C_2(w) \neq \bot \wedge |v - w| \leq 1\}$. Note that a robot is allowed to hold its position. For $M \in \mathbb{N}$, a *schedule* is a sequence $C_1 \to \cdots \to C_{M+1}$ (also abbreviated as $C_1 \rightrightarrows C_{M+1}$) of transformations, with a *makespan* of M. A *stable* schedule $C_1 \rightrightarrows_\chi C_{M+1}$ uses only connected configurations. In the context of this paper, we use these notations equivalently.

Let C_s and C_t be two connected configurations with equally many robots called *start* and *target configuration*, respectively. The *diameter* d of the pair (C_s, C_t) is the maximal Manhattan distance between a robot's start and target position. The *stretch (factor)* of a schedule is the ratio between its makespan M and the diameter d of (C_s, C_t).

2 Fixed makespan

Given two labeled configurations, it is easy to see that it can be determined in linear time whether there is a schedule with a makespan of 1 that transforms one into the other: For every robot, check whether its target position is in distance at most 1; furthermore, check that no two robots want to swap their positions. This involves $O(1)$ checks for every robot, thus $O(n)$ checks in total. We obtain the following.

▶ **Theorem 1.** *It can be decided in $O(n)$ time whether there is a schedule $C_s \rightrightarrows_\chi C_t$ with makespan 1 for any pair (C_s, C_t) of labeled configurations, with n robots each.*

On the other hand, Fekete et al. [18] showed that it is already NP-hard to decide whether a schedule with a makespan of 2 can be achieved, if the robot swarm is unlabeled. Due to the desired makespan, the respective target position of every robot is highly restricted. Thus, it is straightforward to provide a suitable labeling of the configurations such that the very same construction shows NP-hardness for the variant of labeled robot swarms.

For technical details, see the full version [20].

▶ **Theorem 2.** *It is NP-hard to decide whether there is a schedule $C_s \rightrightarrows_\chi C_t$ with makespan 2 for any pair (C_s, C_t) of labeled configurations, with n robots each.*

3 Lower bound on stretch factor

We sketch a lower bound of $\Omega(\sqrt{n})$ on the stretch factor. For this, we consider the pair of configurations (C_s, C_t) shown in Figure 1(a), both consisting of n robots. The difference between both configurations is that adjacent robots need to swap their positions. Thus, the diameter of (C_s, C_t) is $d = 1$. Because swaps are not allowed within the underlying model, some robots have to move orthogonally.

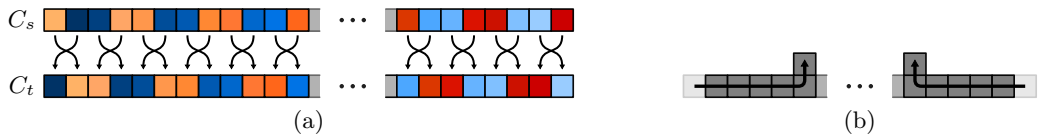


Figure 1 Pairs of robots must swap their positions (a), using moves that involve all robots (b).

Any robot occupying a position adjacent to the line after moving orthogonally can be utilized to perform swaps between robots still contained within the line itself. However, for each robot we move out of the line, we have to simultaneously perform a "shrinking"

motion along the line itself to preserve connectivity, see Figure 1(b). Acquiring λ such robots therefore takes $\lambda/2$ steps. To perform n pairwise swaps, we then need roughly n/λ steps. This sum has a minimum at $\lambda = \sqrt{n}$, and thus we obtain the following theorem.

▶ **Theorem 3.** *There are pairs of labeled configurations C_s and C_t, each with n vertices, so that every schedule $C_s \rightrightarrows_\chi C_t$ has a stretch factor of at least $\Omega(\sqrt{n})$.*

4 Schedules of constant stretch

In this section, we describe an approach to determine stable schedules with constant stretch for labeled configurations of sufficient scale. In particular, we show the following result.

▶ **Theorem 4.** *There is a constant c^* such that for any pair (C_s, C_t) of labeled configurations with n robots each and scale of at least c^*, there exists a constant stretch schedule $C_s \rightrightarrows_\chi C_t$.*

4.1 Overview of the algorithm

Our approach works in different phases; Figure 2 provides an overview. Based on (1) preprocessing steps (Section 4.4) in which we determine the scale c and the diameter d of the configurations, we (2) build a tile-based scaffolding structure (Section 4.5) that guarantees connectivity during the reconfiguration. Then the actual reconfiguration (3+4) consists of shifting robots between adjacent tiles based on flow computations (Sections 4.6–4.9), and (5) reconfiguring tiles in parallel (Sections 4.3 and 4.10). Finally, the deconstruction of the scaffold yields the target configuration. The phases can be summarized as follows.

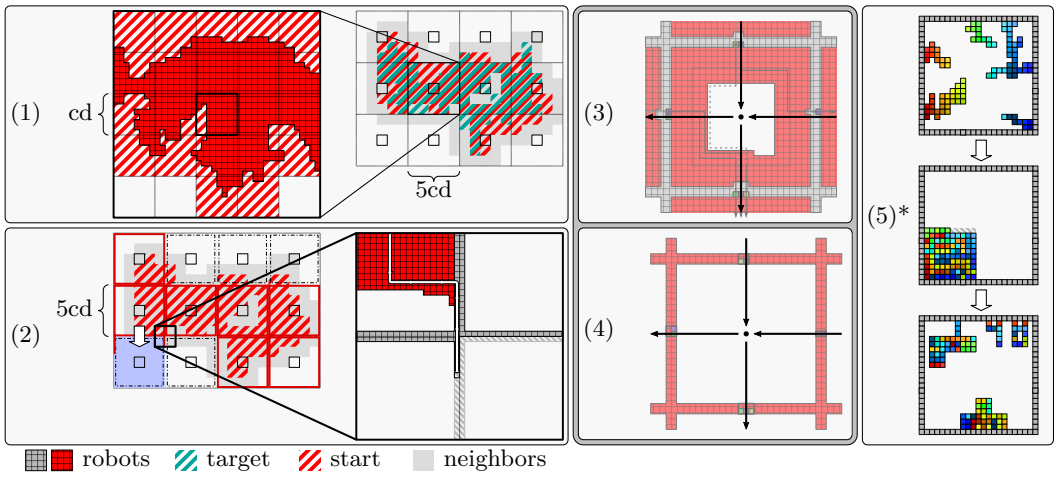


Figure 2 Overview of our algorithm. *Note that ideas of Phase (5) are also used as a subroutine in Phases (2) to (4).

Phase (1) Preprocessing: Compute scale c, diameter d, a tiling $\mathcal{T}_1$ of the grid into squares of size $cd \times cd$, and a larger tiling $\mathcal{T}_5$ covering all non-empty tiles of $\mathcal{T}_1$.

Phase (2) Scaffold construction: Construct a scaffold along the edges of $\mathcal{T}_5$, resulting in a tiled configuration, guaranteeing connectivity during reconfiguration.

Phase (3) Interior flow: Create a flow graph $G_{\mathcal{T}_5}$ to model the movement of interior (non-scaffold) robots between adjacent tiles of $\mathcal{T}_5$. Convert the flow into a series of moves, placing all interior robots in their target tiles.

Phase (4) Boundary flow: Analogously to Phase (3), create a flow for the robots that were used to construct a scaffold in Phase (2). Afterwards, move all of those robots into the boundary of their target tiles.

Phase (5) Local tile reconfiguration: Locally reconfigure all tiles of the resulting tiled configuration.

Phase (6) Scaffold deconstruction: Reverse of Phase (2).

In the remainder of this section we provide details of the different phases. We start by providing some preliminaries needed for the detailed descriptions.

4.2 Preliminaries for the algorithm

In the following we give definitions that are fundamental for the understanding of the algorithm. We give additional definitions in each section as needed.

As an intermediate result for the labeled case, Demaine et al. [11] proposed an algorithm that computes schedules with stretch factors that are linear in the dimensions of a fully occupied rectangular area that is to be reconfigured.

▶ **Lemma 5.** *Let C_s and C_t be two labeled configurations of an $n_1 \times n_2$ rectangle with $n_1 > n_2 \geq 2$. There is a schedule $C_s \rightrightarrows_\chi C_t$ with makespan $O(n_1 + n_2)$.*

In a follow-up paper, Fekete et al. [18] considered arbitrary unlabeled configurations, computing stable schedules of constant stretch. Note that stretch in the unlabeled case is defined via a bottleneck matching of robots between the start and target configuration. They make use of so-called *tilings*, *neighborhoods* of tiles, *layers* in tiles, a *scaffold* based on these layers, and *tiled configurations*, which we define as follows.

An m-tiling is a subdivision of a configuration's underlying grid into squares (called *tiles*) of side length $m \in \mathbb{N}$, each of them anchored at coordinates that are multiples of m in both dimensions. With scale c, diameter d, and $m = cd$, we refer to this tiling as $\mathcal{T}_1$. For a tiling $\mathcal{T}$ and a subset of tiles $\mathcal{T}' \subseteq \mathcal{T}$, the *$k$-neighborhood* $N_k[\mathcal{T}']$ is the set of all tiles from $\mathcal{T}$ with Chebyshev distance at most k to any tile $T \in \mathcal{T}'$. The *boundary* of a tile consists of all nodes in the underlying grid graph that are immediately adjacent to its edge. The first *layer* of the tile is then the set of inward neighboring nodes of the boundary. This relationship applies to successive higher-order layers as well. The *scaffold* of a tiling is the union of all boundaries of its tiles. A *tiled configuration* is a configuration that is a subset of the given tiling and a superset of its scaffold. The *interior* of a tiled configuration is the set of all robots not part of the scaffold.

As an intermediate result they showed the following.

▶ **Lemma 6.** *Let C_s and C_t be two tiled* unlabeled *configurations such that C_s and C_t contain the same number of robots in the interior of each tile T. There is a schedule $C_s \rightrightarrows_\chi C_t$ with makespan $O(d)$.*

4.3 Subroutine: Single tile reconfiguration

A key insight for our approach is that we can efficiently exploit a globally connected structure locally. Before explaining how we achieve this global structure, we show how to exploit it locally. This provides a fundamental subroutine that is used to locally transform tiled configurations within a makespan of $O(d)$. In particular, we obtain the following.

▶ **Theorem 7.** *For any two tiled connected configurations C_s and C_t for which each tile consists of the same robots, there is a stable schedule of makespan $O(d)$ that transforms one into the other.*

In the remainder of this subsection we provide several lemmas that yield a proof of Theorem 7, see full version [20] for details. To this end, we first show that the interior of a tile can be reconfigured arbitrarily, followed by a description of how this can be adapted to include the reconfiguration of the boundary, as well as arbitrary exchanges of robots between a tile's interior and its boundary.

We start by showing how the interior of a tile can be transformed arbitrarily. As moves are reversible, proving that a canonical configuration is reachable is sufficient. Using a technique outlined by Fekete et al. [18], we can gather all robots in a corner of a tile. A straightforward compacting process then yields a configuration that is a subset of a square. Overall this takes $O(d)$ transformations. We thus obtain the following lemma.

▶ **Lemma 8.** *For any two connected configurations consisting of an immobile $m \times m$ boundary for $m \in O(d)$ and $k \leq (m-2)^2$ interior robots, there is a stable schedule of makespan $O(d)$ that moves all interior robots into a subset of a $\lceil\sqrt{k}\rceil \times \lceil\sqrt{k}\rceil$ square.*

By applying Lemma 5 to this square we can rearrange the robots in $O(d)$ steps. We obtain the following lemma; see Figure 2, Phase (5) for the resulting procedure.

▶ **Lemma 9.** *Given $k \in [1, (m-2)^2]$ robots arranged in a subset of a $\lceil\sqrt{k}\rceil \times \lceil\sqrt{k}\rceil$ square, the robots may be arbitrarily rearranged by a stable schedule of makespan $O(d)$.*

Note that if the $\lceil\sqrt{k}\rceil \times \lceil\sqrt{k}\rceil$ square is not completely occupied, we apply Lemma 5 at most three times on different parts of the square to obtain any desired configuration. See the full version for details [20].

We now show how to exchange robots between the tiles' interiors and their boundaries.

▶ **Lemma 10.** *For any tiled configuration of $m \times m$-tiles for $m \in O(d)$, it is possible to exchange any number of robots from each tile's interior with its boundary in $O(d)$ steps.*

Proof. Place at most $4(m-4)$ robots that need to be swapped into the boundary, adjacent to the respective boundary robots that need to swap into the interior. Afterwards apply Lemma 5 to the disjoint areas in parallel. As there are fewer positions on a successive layer, we have to repeat this process at most once. As this operation is performed entirely within a single tile, it may be applied to all tiles simultaneously. ◀

It remains to show that the scaffolding structure can be reconfigured arbitrarily, without modifying its silhouette. To reorder any given tile's boundary, we make use of the $2 \times d$ scaffold robots that separate it from a neighbor's interior. By doing a full revolution of the boundary, we can collect any subset of d robots in that portion of the scaffold, which we then reorder using Lemma 5, see Figure 3. Repeating this at most four times yields a sorted boundary, so we obtain the following lemma. Details can be found in the full version [20].

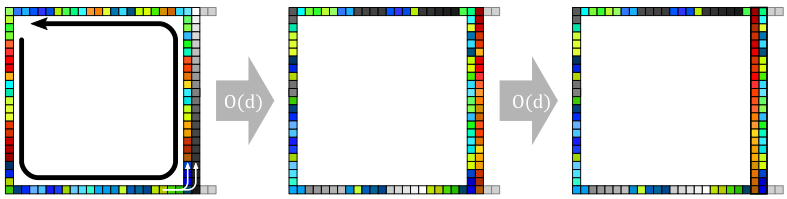


Figure 3 An illustration of the first iteration of the boundary reconfiguration approach.

▶ **Lemma 11.** *The boundaries of all tiles within a tiled configuration that consists of $m \times m$-tiles with $m \in O(d)$ may be locally reordered by a schedule of makespan $O(d)$.*

4.4 Phase (1): Preprocessing

Consider start and target configurations C_s and C_t. Let c refer to the minimum of the two configurations' scales and let d be the diameter of the pair (C_s, C_t). Without loss of generality, assume that the configurations overlap in at least one position. Otherwise, we move the target configuration, such that it overlaps with the start configuration. This can be done in $O(d)$ steps, and results in a new diameter $d' \leq 2d$.

Using c and d, we define a cd-tiling $\mathcal{T}_1$ of the underlying grid. Let $\mathcal{T}_1(C_s)$ and $\mathcal{T}_1(C_t)$ refer to the set of tiles in $\mathcal{T}_1$ that contain robots in C_s and C_t, respectively (see red and green area in Figure 4). We observe that $N_1[\mathcal{T}_1(C_s) \cup \mathcal{T}_1(C_t)]$ is both a subset of $N_2[\mathcal{T}_1(C_s)]$ and of $N_2[\mathcal{T}_1(C_t)]$, because $\mathcal{T}_1(C_s)$ and $\mathcal{T}_1(C_t)$ are fully contained in each other's 1-neighborhoods.

Demaine et al. [11] exploited an $O(d)$-tiling to guarantee that every robot's target position is either within its starting tile, or an immediate neighbor. In an extension of this approach, Fekete et al. [18] employed a cd-tiling; they constructed a scaffold along the tiling boundaries, achieving connectivity during the reconfiguration process. For configurations of sufficient scale, they drew robots from 2-neighborhoods in order to construct each tile's boundary. In our case, robots have individual target positions, so we must take their heading into account when constructing a scaffold. As a result, we consider a higher-resolution tiling, which allows us to ensure that the diameter of the instance does not increase due to scaffold construction.

Based on the non-empty tiles in $\mathcal{T}_1$, we compute the higher-resolution cover of the relevant area by a grid of 5×5 squares of cd-tiles, see Figure 4. Let $T_1, \ldots, T_n$ be n tiles of $\mathcal{T}_1$, such that the distance between any two of them is a multiple of 5 on both the x- and y-axis, and $N_1[\mathcal{T}_1(C_s) \cup \mathcal{T}_1(C_t)] \subseteq N_2[T_1] \cup \cdots \cup N_2[T_n]$. We define the tiling $\mathcal{T}_5 := \{N_2[T_1], \ldots, N_2[T_n]\}$.

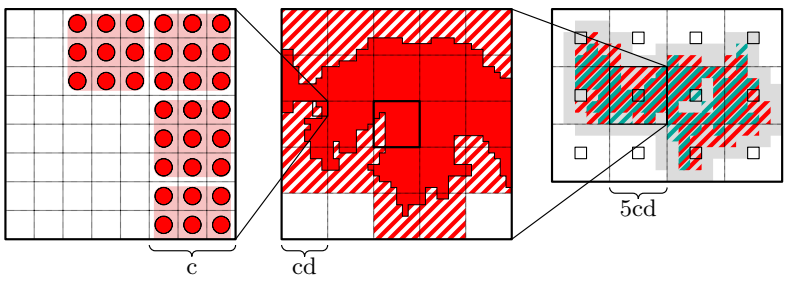


Figure 4 We derive a cd-tiling $\mathcal{T}_1$ (center) from the scale c (left) and the diameter of the instance. We then cover the neighborhood (right, in gray) of non-empty start and target tiles (dashed red and green, respectively) of $\mathcal{T}_1$ by a larger tiling $\mathcal{T}_5$ around which we construct a scaffold for stability.

This concludes the theoretical foundation of our approach. We will now proceed with the first transformation phase, which constructs the fundamental scaffold.

4.5 Phase (2): Scaffold construction

Having determined a cover of C_s and C_t in shape of the $5cd$-tiling $\mathcal{T}_5$, a scaffold spanning this cover is to be constructed. For this purpose, a robot is placed at every boundary position in $\mathcal{T}_5$, which requires $(5 \cdot 4cd - 4)$ robots per tile. We show that there are sufficiently many robots to build the scaffold. For this, we refer to the *locally available material* for a given tile $T \in \mathcal{T}_5$ as the set of robots contained within T itself and its immediate neighborhood $N_1[T]$ over $\mathcal{T}_5$.

▶ **Lemma 12.** *There is a constant c, such that for all C_s and C_t with scale at least c, there is sufficient locally available material to construct a boundary around all tiles of $\mathcal{T}_5$.*

As $\mathcal{T}_5$ is a cover of $N_1[\mathcal{T}_1(C_s) \cup \mathcal{T}_1(C_t)]$, every tile $T \in \mathcal{T}_5$ contains at least one $T' \in N_2[\mathcal{T}_1(C_s)]$. We can thus guarantee sufficient locally available material for the boundary of T if we can show that it exists in the 4-neighborhood of any such T'.

To achieve this, we distinguish *donor* and *recipient* tiles. A donor tile contains enough robots to construct a boundary around itself and all eight immediate neighbors – any tile that cannot do this will be referred to as a recipient. Suppose there is a recipient T such that $N_1[T]$ does not contain a donor. Due to connectivity, we can show that there must be a path that connects T with some tile non-adjacent tile $T' \notin N_1[T]$. Because the path must cross $N_1[T]$, one of the neighboring tiles must have at least $(c^2d)/4$ robots and is thus a donor tile, contradicting the premise. This directly implies that the 1-neighborhood $N_1[T]$ of any $T \in \mathcal{T}_5$ contains at least one donor.

Because every tile is a donor or has a donor as its neighbor, we can construct the scaffold as follows: First, construct the boundary of each donor. Afterwards, each donor can push out robots to neighboring recipients to construct their boundaries. By a careful analysis we can guarantee that the boundary of a recipient T only consists of robots that have their target position in T or in an adjacent tile. Overall we obtain the following.

▶ **Lemma 13.** *Constructing the scaffold takes no more than $O(d)$ transformations.*

A detailed description with a proof of this construction are described in the full version [20].

With the help of this global scaffold structure, connectivity is ensured during the actual reconfiguration. It remains to show how we shift robots between tiles, and reconfigure robot arrangements within the constructed scaffold. The latter has been already described in Section 4.3, so we describe how to relocate robots between tiles. This is modeled as a supply-demand flow for interior robots in three subphases: Phase (3.1) – Interior flow computation; Phase (3.2) – Interior flow partition; and Phase (3.3) – Interior flow realization. A similar approach is used to model the flow for boundary robots, see Section 4.9. We give descriptions of the different phases, and start with *Phase (3.1): Interior flow computation.*

4.6 Phase (3.1): Interior flow computation

Given any tile $T \in \mathcal{T}_5$, its interior robots either need to stay within it or move into a neighboring tile. The anticipated motion is represented as a supply-demand flow $G_{\mathcal{T}_5} := (\mathcal{T}_5, E_{\mathcal{T}_5}, f_{\mathcal{T}_5})$ of the dual graph of $\mathcal{T}_5$. The flow value of an edge $f_{\mathcal{T}_5}(e)$ corresponds to the cardinality of the set of robots that need to move from one tile into another. A tile $T \in \mathcal{T}_5$ is a *source* (*sink*) if and only if the sum of flow values of incoming edges is smaller (larger) than the sum of flow values of outgoing edges. Otherwise, we call T *flow-conserving*. The difference of weights is called *supply* and *demand* for sources and sinks, respectively. If the flow value of every edge within a given flow graph is bounded from above by some value k, we refer to it as a *k-flow*. For simplicity, we consider the number of robots in the flow model, rather than the specific robots themselves. We observe that the flow value $f_{\mathcal{T}_5}(e)$ of each edge $e \in E_{\mathcal{T}_5}$ is bounded from above by the interior space of the tiles, as each may contain at most $(5cd-2)^2 < 25c^2d^2$ robots.

We say that a schedule *realizes* a flow graph $G_{\mathcal{T}_5} = (\mathcal{T}_5, E_{\mathcal{T}_5}, f_{\mathcal{T}_5})$ if for each pair $v, w \in \mathcal{T}_5$ of tiles, the number of robots moved by it from their start tile v to their target tile w is $f_{\mathcal{T}_5}((v,w))$, where we let $f_{\mathcal{T}_5}((v,w)) = 0$ if $(v,w) \notin E_{\mathcal{T}_5}$. Additionally, we define an (a,b)-*partition* of a flow graph as a set that contains b many a-flows that sum up to the original

flow. By construction, the realization of a flow like the one above never requires us to fill a tile over capacity or to remove robots from an empty tile, as this would imply an invalid start or target configuration.

▶ **Lemma 14.** *It is possible to efficiently compute a stable schedule of makespan $O(d)$ that realizes $G_{\mathcal{T}_5}$.*

In the remainder of this section we provide an overview of the properties of $G_{\mathcal{T}_5}$ along with a detailed description of additional measures that split the graph into an acyclic and another totally cyclic component. For each of these components, we will provide an algorithm which computes a $(d, O(d))$-partition of it. Finally, we discuss a set of unified movement patterns that realize each such flow partition through a schedule of makespan $O(d)$.

Note that the initial flow graph may contain diagonal, bidirectional, or crossing edges. By exchanging robots between neighboring tiles, we obtain a configuration with a planar, unidirectional flow graph $G_{\mathcal{T}_5}$. See the full version for details [20].

4.7 Phase (3.2): Interior flow partition

$G_{\mathcal{T}_5}$ is now a planar graph, but neither guaranteed to be acyclic nor totally cyclic. Due to the standard result from the theory of network flows (e.g., see [21, 24]) the algorithm partitions $G_{\mathcal{T}_5}$ into two flows $G_{\rightarrow}$ and $G_{\bigcirc}$, one being acyclic and the other totally cyclic.

▶ **Lemma 15.** *It is possible to efficiently compute a partition of $G_{\mathcal{T}_5}$ into an acyclic component $G_{\rightarrow}$ and a totally cyclic component $G_{\bigcirc}$.*

For the case of configurations that do not have to be connected, Demaine et al. [11] considered tiles of side length $24d$. Thus, they obtained a totally cyclic flow graph $G_{\bigcirc}$ with an upper bound of $24d \cdot 24d = 576d^2$ for the flow value of each edge. Furthermore, they showed that it is possible to compute a $(d, O(d))$-partition of $G_{\bigcirc}$. In our case, we have to keep configurations connected, resulting in tiles of side length cd. Thus, we extend the peeling algorithm from [11], resulting in a specific flow partition to a more general version.

▶ **Lemma 16.** *A $(d, O(d))$-partition of the totally cyclic $k \cdot d^2$-flow $G_{\bigcirc}$ for $k \in \mathbb{N}$ into totally cyclic flows can be computed efficiently.*

In the context of unlabeled robots, Fekete et al. [18] proposed an algorithm for computing a $(O(d^2), 28)$-partition of $G_{\rightarrow}$. Due to the much more complex situation of labeled robots, we employ a number of more refined ideas to provide an algorithm that guarantees the following.

▶ **Lemma 17.** *A $(d, O(d))$-partition of the acyclic $k \cdot d^2$-flow $G_{\rightarrow}$ for $k \in \mathbb{N}$ into acyclic flows can be computed efficiently.*

See the full version [20] for detailed proofs of above lemmas and the respective algorithms.

4.8 Phase (3.3): Interior flow realization

In order to exchange robots between tiles as modeled by the flow $G_{\mathcal{T}_5}$, we have to determine a collision-free protocol that allows robots to pass through the scaffold and into adjacent tiles. To this end, we describe a set of movement patterns for the robots of a single tile; these realize a single d-subflow in a stable manner within $O(d)$ steps. To achieve a compact concatenation of these movement patterns, an invariant type of local tile configuration is of significant importance. Using this invariant, we then provide more compact movement patterns that realize up to d such d-subflows via a schedule of makespan $O(d)$.

These invariant configuration of a tile $T \in \mathcal{T}_5$ are *push-stable* (with respect to a flow $G_{\mathcal{T}_5}$), defined by the following connectivity conditions for every robot r on an ith layer of the interior of T. If $i \leq d$ then r is connected to the closest boundary robot by a straight line of robots. Otherwise, r is connected to a robot r' on layer d by a path of robots in higher-order layers than d, such that the closest side to r' of the boundary of T rests on an edge without outgoing flow.

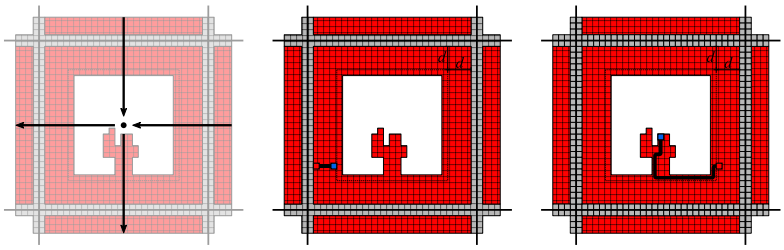


Figure 5 An example of a push-stable configuration of a tile with two incoming and two outgoing edges (left). Highlighted are a robot on layer d (center) and on a higher-order layer (right), with paths that ensure their connectivity.

A *total sink* (*total source*) is a tile T that has four incoming (outgoing) edges of non-zero value over $G_{\mathcal{T}_5}$. Conversely, a *partial sink* (*partial source*) is a tile T that is not flow-conserving over $G_{\mathcal{T}_5}$, but has no more than three incoming (outgoing) edges of non-zero value. Note that by definition, total sources can never be configured in a push-stable manner. As a consequence, we handle both total sinks and total sources separately.

We briefly sketch our approach that realizes a single subflow.

▶ **Lemma 18.** *Consider a d-subflow $H_{\mathcal{T}_5} \subseteq G_{\mathcal{T}_5}$ and a tile $T \in \mathcal{T}_5$ that is flow-conserving with respect to $H_{\mathcal{T}_5}$. There is a schedule of makespan $O(d)$ that realizes the flow at its location.*

For details, we refer the reader to the full version. However, the high-level idea is to use Theorem 7 to construct a push-stable configuration and connect incoming and outgoing robots in a crossing-free manner by using higher-order layers, see Figure 6(c). We then move every robot along its respective path if it lies on layer at most d, or if the partial path from the incoming position to the robot contains no empty position; see the dashed lines in Figure 6(c). By a careful analysis we obtain the following lemma.

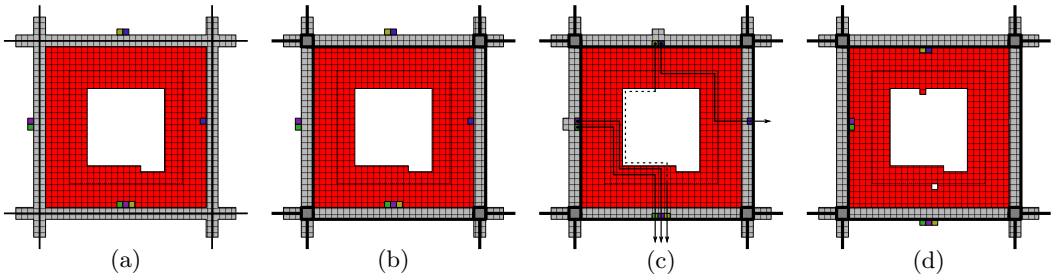


Figure 6 A tile $T \in \mathcal{T}_5$ during the realization of a single d-subflow. The dashed portions of paths indicate that no motion actually occurs in this segment due to the described pushing behavior.

▶ **Lemma 19.** *The realization of a d-subflow can be performed in a way that results in another push-stable configuration of T.*

By slightly modifying this approach, mainly by leaving a specific set of positions empty (partial sources) or letting them become empty (partial sinks) during a sequence of parallel motions, this method becomes applicable for non-flow-conserving tiles.

▶ **Lemma 20.** *Consider a d-subflow $H_{\mathcal{T}_5} \subseteq G_{\mathcal{T}_5}$ and a tile $T \in \mathcal{T}_5$ that is not a total sink or total source with respect to $G_{\mathcal{T}_5}$. There is a schedule of makespan $O(d)$ that realizes the flow at its location and yields another push-stable configuration.*

The previously described movement patterns may be compactly combined into a single schedule of makespan $O(d)$ that realizes up to d many d-subflows at once. Intuitively, this can be achieved by stacking the outgoing robots against the target tile's boundary during the push-stable construction, see Figure 7. Because there are at most d subflows, these stacks of robots have height at most d. Therefore, we can successively realize each flow after another $O(1)$ steps per subflow.

▶ **Lemma 21.** *Consider a tile $T \in \mathcal{T}_5$ that is not a total sink or total source with respect to $G_{\mathcal{T}_5}$ and a sequence $S_{\mathcal{T}_5} := (H^1_{\mathcal{T}_5}, \dots, H^\ell_{\mathcal{T}_5})$ of d-subflows of $G_{\mathcal{T}_5}$ with $\ell \leq d$. There exists a stable schedule of makespan $O(d)$ that realizes all ℓ many d-subflows.*

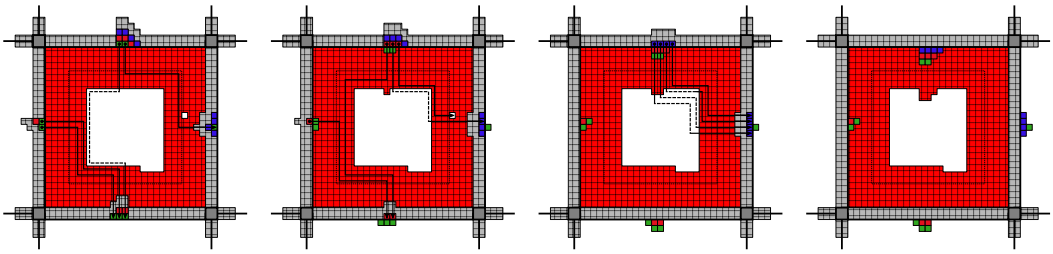


Figure 7 A tile $T \in \mathcal{T}_5$ during the realization of ℓ d-subflows.

Total sinks and sources (as defined above) form a special case and are handled separately from the described push-stable patterns. Every total sink has at least as many empty positions as the number of incoming robots. By carefully rearranging the interior of the respective tile, we can realize each subflow by filling empty positions with robots. For the set of total sources, we consider the reverse operation.

▶ **Lemma 22.** *Consider a tile $T \in \mathcal{T}_5$ that is a total sink or total source with respect to $G_{\mathcal{T}_5}$ and a sequence $S_{\mathcal{T}_5} := (H^1_{\mathcal{T}_5}, \dots, H^\ell_{\mathcal{T}_5})$ of d-subflows of $G_{\mathcal{T}_5}$ with $\ell \leq d$. There exists a stable schedule of makespan $O(d)$ that realizes all ℓ many d-subflows.*

By applying Lemmas 21 and 22, up to d many d-subflows of $G_{\mathcal{T}_5}$ may be realized in $O(d)$ transformations. As we created $O(d)$ such subflows in Section 4.7, all of them may be realized through $^{O(d)}/_{d} = O(1)$ repetitions. These repetitions require a total of $O(d)$ transformations.

4.9 Phase (4): Boundary flow

The next phase of our algorithm deals with the movement of robots that are part of the scaffold. As described in Section 4.5, the robots forming the scaffold in a tiled configuration must remain in the 1-neighborhood of their target tile over $\mathcal{T}_5$. The intermediate mapping step in the construction process guarantees that this remains the case even after that phase concludes. However, the scaffold does not necessarily consist of the same robots in the tiled configurations C'_s and C'_t. We observe that for any given tile $T \in \mathcal{T}_5$, up to $20cd - 4$ robots exist in its neighborhood $N_1[T]$ that need to become part of its boundary structure.

This observation induces another supply-demand flow graph similar to the one in Section 4.6, with each edge's flow value bounded from above by $20cd - 4$. Using the techniques discussed in Sections 4.6 and 4.7, we obtain a $(d, O(1))$-partition of this flow graph. This requires minor modifications to the involved methods. See full version [20] for details.

Each of the computed subflows can then be realized by a schedule of makespan $O(d)$. As no tile can ever end up with fewer than $20cd - 4$ total robots in the tiled target configuration, we can simply rearrange all tiles according to Theorem 7, placing outgoing robots adjacent to their target tile before pushing them into the boundary of their target tile and displacing part of the scaffold in the process. While this creates "gaps" in the boundary of all involved tiles, such movement never causes a disconnection in the full scaffold, as the receiving tile's boundary preserves connectivity (see Figure 8). The resulting gap can thus be safely patched up with incoming robots. We repeat this process until each tile contains the correct robots for its scaffold.

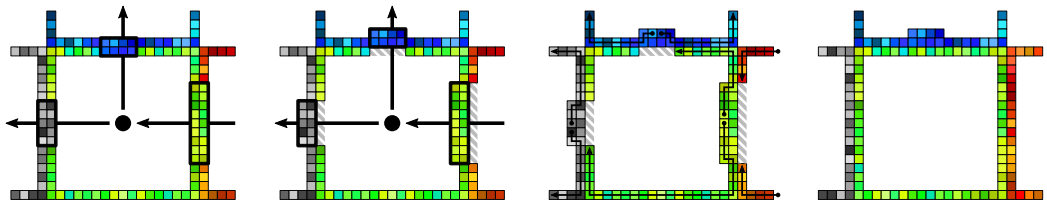


Figure 8 We can employ the displayed movement patterns to exchange boundary robots between adjacent tiles.

4.10 Phase (5): Local tile reconfiguration

Once Phase (4) concludes, we have reached a tiled configuration in which every tile contains precisely the robots that it would in a tiled configuration C'_t of the target configuration. This means that we can reconfigure into C'_t by a single application of Theorem 7, which forms the entirety of Phase (5).

As Phase (6) is a reverse of Phase (2), this concludes the description of the algorithm. Because each phase takes $O(d)$ transformation steps, this proves Theorem 4.

5 Conclusion and future work

We resolved two major open problems for connected reconfiguration. Some open problems still remain. Does any connected arrangement of n robots with diameter $D \leq n$ allow a makespan of $O(\sqrt{D})$? Can the involved constants of our methods be significantly reduced? Of interest are also implementations of largely distributed computation and motion control for efficient parallel motion schedules.

References

1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015. doi:10.1109/TASE.2015.2470096.

2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The O(1) musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

3 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 4:1–4:19, 2022. doi:10.4230/LIPIcs.SWAT.2022.4.

4 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. Coordinated motion planning: The video. In *Symposium on Computational Geometry (SoCG)*, pages 74:1–74:6, 2018. Video at https://www.ibr.cs.tu-bs.de/users/fekete/Videos/CoordinatedMotionPlanning.mp4. doi:10.4230/LIPIcs.SoCG.2018.74.

5 Julien Bourgeois, Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Benoît Piranda, Christian Rieck, and Christian Scheffer. Space Ants: Episode II – Coordinating Connected Catoms. In *Symposium on Computational Geometry (SoCG)*, pages 65:1–65:6, 2022. doi:10.4230/LIPIcs.SoCG.2022.65.

6 Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008. doi:10.1137/060652063.

7 Soon-Jo Chung, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar. A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018. doi:10.1109/TRO.2018.2857475.

8 Loïc Crombez, Guilherme Dias da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso. Shadoks approach to low-makespan coordinated motion planning. In *Symposium on Computational Geometry (SoCG)*, pages 63:1–63:9, 2021. doi:10.4230/LIPIcs.SoCG.2021.63.

9 Daniel Delahaye, Stéphane Puechmorel, Panagiotis Tsiotras, and Eric Féron. Mathematical models for aircraft trajectory design: A survey. In *Air Traffic Management and Systems*, pages 205–247. Springer, 2014. doi:10.1007/978-4-431-54475-3_12.

10 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane Souvaine. Staged self-assembly: Nanomanufacture of arbitrary shapes with O(1) glues. *Natural Computing*, 7(3):347–370, 2008. doi:10.1007/s11047-008-9073-0.

11 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Christian Scheffer, and Henk Meijer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.

12 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017. doi:10.1016/j.tcs.2016.11.020.

13 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the Kolmogorov bound with small scale factor. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 201–212, 2011. doi:10.4230/LIPIcs.STACS.2011.201.

14 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006. doi:10.1007/s00373-005-0640-1.

15 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Formations for fast locomotion of metamorphic robotic systems. *International Journal of Robotics Research*, 23(6):583–593, 2004. doi:10.1177/0278364904039652.

16 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration. *Transactions on Robotics*, 20(3):409–418, 2004. doi:10.1109/TRA.2004.824936.

17 Sándor P. Fekete, Björn Hendriks, Christopher Tessars, Axel Wegener, Horst Hellbrück, Stefan Fischer, and Sebastian Ebers. Methods for improving the flow of traffic. In *Organic Computing – A Paradigm Shift for Complex Systems*. Springer, 2011. doi:10.1007/978-3-0348-0130-0_29.

18 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 9:1–9:16, 2021. doi:10.4230/LIPIcs.ISAAC.2021.9.

19 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG:SHOP Challenge 2021, 2021. arXiv:2103.15381.

20 Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficiently reconfiguring a connected swarm of labeled robots, 2022. arXiv:2209.11028.

21 Lester R. Ford and Delbert R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

22 Seth C. Goldstein and Todd C. Mowry. Claytronics: A scalable basis for future robots, 2004. URL: http://www.cs.cmu.edu/~claytronics/papers/goldstein-robosphere04.pdf.

23 Stephen Kloder and Seth Hutchinson. Path planning for permutation-invariant multi-robot formations. *IEEE Transactions on Robotics and Automation*, 22(4):650–665, 2006. doi:10.1109/TRO.2006.878952.

24 Bernhard H. Korte and Jens Vygen. *Combinatorial optimization*. Springer, 2011.

25 Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k-opt. In *Symposium on Computational Geometry (SoCG)*, pages 64:1–64:8, 2021. doi:10.4230/LIPIcs.SoCG.2021.64.

26 Austin Luchsinger, Robert T. Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, 18(1):93–105, 2019. doi:10.1007/s11047-018-9707-9.

27 Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014. doi:10.1126/science.1254295.

28 Erol Şahin and Alan F. T. Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2-4):69–72, 2008. doi:10.1007/s11721-008-0020-6.

29 Michael Schreckenberg and Reinhard Selten. *Human Behaviour and Traffic Networks*. Springer, 2013. doi:10.1007/978-3-662-07809-9.

30 Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.

31 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. doi:10.1137/S0097539704446712.

32 Kiril Solovey and Dan Halperin. k-color multi-robot motion planning. *International Journal of Robotics Research*, 33(1):82–97, 2014. doi:10.1177/0278364913506268.

33 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *International Journal of Robotics Research*, 35(14):1750–1759, 2016. doi:10.1177/0278364916672311.

34 Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion planning for unlabeled discs with optimality guarantees. In *Robotics: Science and Systems*, 2015. doi:10.15607/RSS.2015.XI.011.

35 Matthew Turpin, Nathan Michael, and Vijay Kumar. Trajectory planning and assignment in multirobot systems. In *Algorithmic Foundations of Robotics X – Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 175–190. Springer, 2013. doi:10.1007/978-3-642-36279-8_11.

36 Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*, 37(4):401–415, 2014. doi:10.1007/s10514-014-9412-1.

37 Hyeyun Yang and Antoine Vigneron. A simulated annealing approach to coordinated motion planning. In *Symposium on Computational Geometry (SoCG)*, pages 65:1–65:9, 2021. doi:10.4230/LIPIcs.SoCG.2021.65.

Entropy Matters: Understanding Performance of Sparse Random Embeddings

Maciej Skorski
University of Luxembourg, Luxembourg

Abstract

This work shows how the performance of sparse random embeddings depends on the Renyi entropy-like property of data, improving upon recent works from NIPS'18 and NIPS'19.

While the prior works relied on involved combinatorics, the novel approach is simpler and modular. As the building blocks, it develops the following probabilistic facts of general interest:

(a) a comparison inequality between the linear and quadratic chaos

(b) a comparison inequality between heterogenic and homogenic linear chaos

(c) a simpler proof of Latala's strong result on estimating distributions of IID sums

(d) sharp bounds for binomial moments in all parameter regimes.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Theory of computation → Random projections and metric embeddings

Keywords and phrases Random Embeddings, Sparse Projections, Renyi Entropy

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.18

1 Introduction

The celebrated result due to Johnson and Lindenstrauss [38] states that random linear mappings are perfect embeddings: they *nearly preserve distances* of input data points, while mapping them into a *much lower dimension*. This enables accomplishing otherwise computationally demanding tasks, by running on the *reduced yet representative data*. Formally, the lemma states that for any distortion $\epsilon > 0$ and confidence parameter $0 < \delta < 1$, with the embedding dimension $m = \Theta(\log(1/\delta)\epsilon^{-2})$ and the $m \times n$ matrix A sampled from the appropriately scaled normal or Rademacher distribution, for every vector $x \in \mathbb{R}^n$

$$(1-\epsilon)\|x\|_2 \leqslant \|Ax\|_2 \leqslant (1+\epsilon)\|x\|_2 \quad \text{with probability } 1-\delta. \tag{1}$$

For modest but practically meaningful distortion and confidence parameters ϵ, δ and large data dimensions n we obtain $m \ll n$, that is a *significant dimension reduction*; on the other hand nearly-preserving distances (up to a relative factor of ϵ) translates into nearly-preserving scalar products and thus the internal data geometry, making it *representative for many tasks*. Indeed, over the years variants of the *Johsnon-Lindenstrauss Lemma* have found important applications to text mining and image processing [7], approximate nearest neighbor search [35, 3], learning mixtures of Gaussians [22], sketching and streaming algorithms [43, 47], approximation algorithms for clustering high dimensional data [6, 12, 54], speeding up linear algebraic computations [57, 61, 16], analyzing combinatorial properties of graphs [28, 52] and even to privacy [9, 42]; on the pure theory side, it is worth mentioning the importance for understanding Hilbert spaces in functional analysis [39].

Although the embedding dimension m is optimal [40, 37], the costly matrix-vector product can be optimized by the use of *sparse matrices*. The long line of research [1, 21, 51, 3, 55, 41, 18] have finally established the same guarantees for matrices A with only $s = \Theta(\log(1/\delta)\epsilon^{-1})$

entries per column[1]. Optimal in the worst-case, these results were far away from the performance empirically observed on real-world data, particularly the remarkable accuracy of *feature hashing* [65] which uses only $s = 1$ (!). This leaded to the following intriguing question:

> Why extremely sparse random projections work better-than-expected?

A careful reader notices that so far we have been speaking of *data-oblivious* results, that is under no assumption on the data structure. Indeed, the relevant research in [65, 21, 41, 30, 36] has finally established [30, 36] that the certain metric which captures *data dispersion*, more precisely the ratio $v = \|x\|_\infty / \|x\|_2$, allows for setting the matrix sparsity to[2]

$$s = \Theta(v^2 \epsilon^{-1}) \cdot \max\left\{ \log \frac{1}{\delta}, \frac{\log^2 \frac{1}{\delta}}{\log^2 \frac{1}{\epsilon}} \right\} \tag{2}$$

while keeping the optimal dimension $m = \Theta(\log(1/\delta)\epsilon^{-2})$. This offers an additional improvement by a factor of $1/v^2$. In simple terms: the more data is dispersed, the better matrix sparsity works. This breakthrough result still suffers from the following limitations:

1. *Unsatisfactory definition of data dispersion.* The ratio ℓ_∞-to-ℓ_2 is a crude notion: on the unit sphere $\|x\|_2 = 1$ it depends on the heaviest element and so is not smooth enough. It suffers particularly from "spikes" that are naturally present in real-world data (such as features produced in text-mining [4]) and due to pairwise vector differences studied in multi-vector setup (uniform guarantees for multiple vectors are obtained by looking at pairwise differences $x - x'$, which leads to "spikes" for example in images [48]). This motivates further research for a *more accurate notion of dispersion.*
2. *High proof complexity and lack of modern toolkit.* Proofs in prior works [30, 36] suffer from being lengthy and convoluted, mostly in supplementary materials, which results in numerical mistakes as well as gaps not immediately fixable (see Appendix A). These works did admirable efforts on presenting the self-contained proof, yet did not utilize the modern probability toolkit to the full extent. Their strategy is to see Equation (1) as the concentration of the *quadratic form* $x \to \|Ax\|_2^2$, and quantify its tails by controlling high-order moments estimated via multinomial expansions coupled with combinatorial arguments. However, this does not leverage tools to control quadratic random forms, namely the modern techniques of the *Hanson-Wright inequality* [31, 60, 67] such as decoupling of quadratic forms [63, 24]. Furthermore, it re-develops a variant of the sharp result from [49] on moment estimation and certain known facts from *high-dimensional probability* on sub-gaussian distributions [11, 10]. Finally, while [36] develops its technical lemmas for symmetric random variables, this condition is not satisfied which leaves a gap. Thus, further effort in *revisiting and modernizing the toolkit* used in recent state-of-art works [30, 36] is well-motivated. Indeed, simplifying proofs and developing novel techniques for the JL Lemma is an independent and valued line of research [28, 29, 23, 19], as these have been historically difficult (the original result used sophisticated geometric approximations, while the sparse variant [21] relied on correlation inequalities [27]).

[1] As shown by [18] one can reduce further sparsity s by $B > 1$ at the cost of exponentially increasing the dimension m by a factor of $2^{\Theta(B)}$. However, in practice, sub-optimal dimension is less interesting.

[2] The formula arises from rearranging Theorem 1.5 in [36]

2 Our Contribution

This work offers a solution to the two problems discussed above: we *strengthen* and to a great extent *simplify* the state-of-art results from prior works.

2.1 Performance of Sparse Random Projections

We introduce the following (novel) notion of the *data dispersion*:

$$v_d(x) \triangleq \sup_{|I|<d/2} \left(\frac{\sum_{i\notin I} |x|_i^d}{\sum_{i\notin I} x_i^2} \right)^{\frac{1}{d-2}} / \|x\|_2, \quad d>2. \tag{3}$$

where I are taken as strict subsets of the support of x.

The matrix A is sampled from the sparisfied Rademacher distribution, as in prior works:

Algorithm 1 Sparse Random Projections: Matrix Sampler.

Data: data dimension n, embedding dimension m, matrix sparsity s
Result: $A \in \mathbb{R}^{n\times m}$
- for every column i, select s positions at random (without replacement)
- set randomly ± 1 on the selected positions
- scale the matrix by $1/\sqrt{s}$

For the matrix as in Algorithm 1 above, we prove the following result.

▶ **Theorem 1.** *Let $d=\log(1/\delta)$, then the JL Lemma, that is* (1)*, holds for the dimension*

$$m = \Theta(d\epsilon^{-2}) \tag{4}$$

and any sparsity s such that

$$v_d(x) \leqslant \Theta(s\epsilon)^{1/2} \min(\log(m\epsilon/d)/d, 1/d^{1/2}). \tag{5}$$

We now discuss the result in detail in the series of remarks below.

▶ Remark 2 (Intuition). We give the following rationale for one could conjecture a result like the one above: the analysis of sparse random projections establishes that the performance depends on the d-th moment of the error expression, where $d=\log(1/\delta)$ is relatively small; it seems reasonable to expect that the assumptions on the data should not include moments higher than of order d, particularly bounding $\|x\|_\infty$ seems to be an overshooting.

▶ Remark 3 (Comparison with previous bounds). Since $v_d(x) \leqslant \|x\|_\infty/\|x\|_2$, we obtain the previous state-of-art bounds from [36], by rearranging Equation (5) to Equation (2). This approximation is however rather crude, as it merely replaces the d-th norm $\|\cdot\|_d$ by $\|\cdot\|_\infty$, and our bound can do much better. Consider the more explicit example where $x_i^2=(n/d)^{-1/d}$ for d values of i and $x_i^2 = 1-(n/d)^{-1/d}/(n-d)$ otherwise. We then have $v_d(x)=\Theta(n^{-\frac{2}{d-2}})$ while $\|x\|_\infty/\|x\|_2=\Theta(n^{-\frac{1}{d}})$). Since the best possible sparsity s is roughly proportional to $v_d(x)^{-2}$, our gain over the previous approach is by a factor of $n^{\frac{4}{d-2}-\frac{2}{d}}$ which is huge for moderate values of d and large n (that is, in a typical application regime).

▶ Remark 4 (Relation to Renyi Entropy). Let's introduce the probability measure $w_i \sim x_i^2$, then $(\sum_i |x_i|^d/\sum_i x_i^2)^{\frac{1}{d-2}}/\|x\|_2 = (\sum_i w_i^{\frac{d}{2}})^{\frac{1}{d-2}} = 2^{H_{d/2}((w_i))/2}$ where the Renyi entropy [58] of the distribution w is defined as $H_d(w) \triangleq \frac{1}{1-d}\sum_i w_i^d$ and $H_\infty(w) \triangleq -\log\max_i w_i$ when

$d = \infty$. Under the mild assumption that x such that $\sum_{i \notin I} x_i^2 = \Theta(\|x\|_2^2)$ for all $|I| \leqslant d$ we can thus compare the sparsity achieved in Theorem 1 and the result in [36] as low-order Renyi entropy versus min-entropy. More precisely, our bound on s is better by a factor of $2^{H_{d/2}((w_i)) - H_\infty((w_i))}$, that is the gain is *exponential in entropy deficiency* understood as $H_{d/2}((w_i)) - H_\infty((w_i))$. The well-known bounds from information-theory [14] show that this gap can be as big as $\frac{1}{d/2-1} H_{d/2}((w_i))$ (which is unbounded without some restrictions on x).

▶ Remark 5 (Dimension-Sparsity Tradeoffs). It is possible to improve the sparsity parameter s by a factor of B at the expense of making the dimension worse by a factor of $\mathrm{e}^{\Theta(B)}$, exactly as in [36]. However, this tradeoff does not seem to be interesting from the application-oriented point of view (the whole idea of random projections is to keep the low dimension).

2.2 Techniques of Independent Interest

2.2.1 From Quadratic to Linear Chaos

One important novelty in our approach is that we get rid of analyzing quadratic forms, which appear due to considering the expression $\|Ax\|_2^2$, by an elegant reduction to their linear analogues. Although quadratic chaoses of symmetric random variables have been studied in the past [49, 46], the generic bounds were found intractable to analyze by the authors of prior works [30, 36] and other workarounds have been proposed. It has been not clear if one can get rid of these complicated methods. Indeed, we show that we can:

▶ **Lemma 6.** *Let X_i be independent zero-mean random variables, with possibly different distributions. Then for even $d \geqslant 2$ we have*

$$\|\sum_{i \neq j} X_i X_j\|_d \leqslant 32 \|\sum_i X_i\|_d^2.$$

▶ Remark 7. The result is fairly general, not requiring symmetry or identical distributions. In fact, the constant reduces to 4 if X_i are already symmetric.

This bound allows for reducing a bulk of technical calculations, and almost directly applying existing *tractable bounds* for linear forms such as those in [50]. The proof uses *decoupling* [63] which allows for upper-bounding the moments of the quadratic form $\sum_{i \neq j} X_i X_j$ by the moments of bilinear form $\sum_{i \neq j} X_i X_j'$, and *symmetrization* [64] which allows for replacing X_i by their symmetrized versions $X_i - X_i'$ at the expense of a constant factor.

2.2.1.1 Heterogenic Sparse Rademacher Chaos

Although we reduce the problem to studying linear forms, they are not IID sums. More precisely in our case we will be interested in sums of form $\sum_i x_i X_i$ where X_i are symmetric and IID, but the given weights x_i can be very different. Such sums are notoriously difficult to analyze, the best example being probably the classical Khintchine's inequality which seeks to bound $\|\sum_i x_i \sigma_i\|_d$ where σ_i are Rademachers, for a given sequence of weights (x_i); it took a while until the original bounds [44] have been tightened, in a way that explicitly depend on x [33]. While prior works [30, 36] handle this difficulty in our context implicitly (in combinatorial analyses of multinomial expansions), we use *majorization theory* to essentially compare the heterogenic and homogenic (easier) setup. We prove

▶ **Lemma 8.** *Let $\|x\|_2 = 1$ and $X_i \sim^{IID} \eta_i\sigma_i$ where η_i are IID Bernoulli and σ_i are IID Rademacher r.vs. Then for $v = v_d(x)$ where $v_d(x)$ is as in Equation (3), and even $d > 0$*

$$\|\sum_i x_i X_i\|_d \leqslant O(\|K^{-1/2}\sum_{i=1}^{K} X_i\|_d), \quad K = \lceil v^{-2}\rceil.$$

The result depends on the structure of x captured by $v = v_d(x)$, note that the equality holds when $x_i = v$ for all non-zero weights x_i (note that we normalize $\|x\|_2 = 1$ w.l.o.g.); this is the core of our method, and we can see it as a sparse analogue of Khintchine's Inequality (Bernoulli variables restrict the summation to a random subset). The result should be considered strong and somewhat surprising; per analogy to the case when there are no Bernoulli variables, results from majorization theory seem to suggest that the moment should be rather minimized for x_i that are nearly uniform[3] . The answer is in the condition $v_d(x)$ which is, to a certain degree, a relaxation of the requirement that x_i is flat and in the constant under $O(1)$. What we prove is not that (x_i) with K elements gives the maximum, but that the value differs from the actual maximum by at most a constant factor. In our proof, we use the assumption in Equation (3) and majorization [17] to compare the behavior of sums $S_k = \sum_{i_1\neq\ldots i_k} x_{i_1}^2\cdots x_{i_k}^2$ when x_i is uniform over K elements versus over the whole space. Under the normalizing condition $\|x\|_2 = 1$, they can be interpreted as *birthday collision probabilities*, which makes the comparison easy to evaluate.

2.2.1.2 Moments of IID Sums

We will need a result which provides *tight bounds on moments of iid sums*. Although this problem has been solved by a characterization due to Latala [50], the result seems to be little known within the TCS community; instead classical bounds due to Hoeffding [34], Chernoff [15], Bernstein [5] or more modern bounds stated sub-gaussian or sub-gamma distributions [11] are used. Since the analysis of sparse random projections involves random variable with little exotic behavior, the classical inequalities are not sufficient.

In hope for popularizing the technique and to make the paper self-consistent, we provide an alternative and simpler proof of Latala's result [50].

▶ **Lemma 9.** *For zero-mean r.vs. $X_i \sim^{IID} X$ and even $d > 0$*

$$\|\sum_{i=1}^{n} X_i\|_d \leqslant 2\mathrm{e}\cdot\max_k\left[\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k}\|X\|_k : \max(2,d/n)\leqslant k\leqslant d\right] \tag{6}$$

which implies the following simpler bound

$$\|\sum_{i=1}^{n} X_i\|_d \leqslant \frac{2\mathrm{e}^2}{(1-\mathrm{e}^{-1})^{1/2}}\cdot\max_k\left[d/k\cdot(n/d)^{1/k}\cdot\|X\|_k : \max(2,d/n)\leqslant k\leqslant d\right]. \tag{7}$$

▶ Remark 10. In addition to simplifying the proof, we provide an explicit constant (not given in the original proof). For non-symmetric distributions, our numerical constant is better than the one implied by symmetrizing the original proof. We also note that there is the same matching, up to a constant, lower bound [49], so that in the result above we have the equality up to a constant.

[3] The map $(x_i)\to\|\sum_i x_i\sigma_i\|_d^d$ is Schur-concave in variables x_i^2 [26].

2.2.1.3 Sharp Bounds for Binomial Moments

Having reduced the problem to studying moments of $\sum_i \eta_i\sigma_i$, we face the problem of estimating binomial moments. Somewhat surprisingly, the literature does not offer good bounds for binomial moments. What we know are combinatorial formulas [45] not in a closed asymptotic form, and nearly perfect estimates (up to $o(1)$ relative error) for binomial probabilities [62] as well as the tails [20, 53, 56] (see also the survey in [2]); these tails unfortunately lead to intractable integrals expressing moments (with Kullback-Leibler terms).

Since the question is foundational with clear potential for applications beyond our problem, we give the following general and detailed answer

▶ **Lemma 11.** *Let $S \sim \mathrm{Binom}(K,p)$ where $p \leqslant \frac{1}{2}$, and $d > 0$ be even. Then*

$$\|S - \mathbf{E}S\|_d = \Theta(1)\begin{cases}(dKp)^{1/2} & \log(d/Kp) < d/K \leqslant 2\\ Kp^{K/d} & \log(d/Kp) < 2 \leqslant d/K\\ \frac{d}{\log(d/Kp)} & \max(2,d/K) \leqslant \log(d/Kp) \leqslant d\\ (Kp)^{1/d} & d < \log(d/Kp)\end{cases}. \tag{8}$$

▶ Remark 12. The bound has up to 4 regimes, in which we provide an estimate sharp up to a constant. The upper bound (sufficient for our needs) follows from Lemma 9, while the lower bound holds because the bound in Lemma 9 is sharp up to an absolute constant [49].

2.3 Proof Outline

We actually prove that

$$(1-\epsilon)\|x\|_2^2 \leqslant \|Ax\|_2^2 \leqslant (1+\epsilon)\|x\|_2^2 \quad \text{with probability } 1-\delta \tag{9}$$

from which Equation (1) follows by taking the square roots and using the elementary inequalities $\sqrt{1+\epsilon} \leqslant 1+\epsilon$, $1-\epsilon \leqslant \sqrt{1-\epsilon}$. Denoting $Z = \|Ax\|_2^2 - \|x\|_2^2$ we find [36])

$$Z = \frac{1}{s}\sum_{r=1}^{m} Z_r, \quad Z_r \triangleq \sum_{i\neq j} x_i x_j \eta_i \eta_j \sigma_i \sigma_j. \tag{10}$$

It can be shown that Z_r are *negatively dependent* and thus their sum obey moment upper-bounds for independent random variables [25, 8]. More precisely we have that

$$\|Z\|_d \leqslant \frac{1}{s}\|\sum_{r=1}^{m} Z_r\|_d, \quad Z_r \sim^{IID} \sum_{i\neq j} x_i x_j \eta_i \eta_j \sigma_i \sigma_j. \tag{11}$$

The techniques outlined above, namely Lemma 6 and Lemma 8 show that for $K = \lceil v_d(x)^{-2}\rceil$

$$\|Z_r\|_d \leqslant O(K^{-1}\|S - S'\|_d^2), \quad S, S' \sim^{IID} \mathrm{Binom}(K,p). \tag{12}$$

Since $\|S - S'\|_d \leqslant 2\|S - \mathbf{E}S\|_d$ (the triangle inequality), by Lemma 11 we obtain

▶ **Corollary 13.** *For any even $d > 0$ we have*

$$\|Z_r\|_d \leqslant O(1)\begin{cases} dp & \log(d/Kp) < d/K \leqslant 2\\ Kp^{2K/d} & \log(d/Kp) < 2 \leqslant d/K\\ \frac{K^{-1}d^2}{\log^2(d/Kp)} & \max(2,d/K) \leqslant \log(d/Kp) \leqslant d\\ K^{-1}(Kp)^{2/d} & d < \log(d/Kp)\end{cases}. \tag{13}$$

It now suffices to plug this bound in Lemma 8 (it applies for negatively dependent r.vs.) and analyze the 4 different regimes, to obtain moment bounds for Z from Equation (10); then Theorem 1 follows by Markov's inequality. The work has been mostly finalized at this point, due to our modular approach; the application of Lemma 8 is discussed in the appendix.

▶ Remark 14. At the final stage [36] also obtains analogous bounds (with K defined in terms of $v = \|x\|_\infty/\|x\|_2$). They are however not derived via a single application of a lemma, but rather a mixture of three techniques (direct bounds on quadratic forms, linear forms, and the reproved result on the sub-gaussian norm of a binary random variable [13]).

2.4 Organization

The rest of the paper is organized as follows: in Section 3 we introduce basic notation and some simple auxiliary facts that will be used throughout the discussion, in Section 4 we present proofs of the key ingredients of our proof. Details omitted in the proof outline are provided in Appendix B. In Section 5 we conclude the work.

3 Preliminaries

3.1 Basic Notation

For a random variable X, we define its d-th moment as $\mathbf{E}|X|^d$ and its d-th norm as $\|X\|_d = (\mathbf{E}|X|^d)^{1/d}$ (this is indeed a norm when $d \geqslant 1$). For the sequence (x_i) we define $\|(x_i)\|_d = (\sum_i |x_i|^d)^{1/d}$ for $0 < d < 1$, $\|x\|_\infty = \max_i |x_i|$ and $\|x_i\|_0 = \#\{i : x_i \neq 0\}$.

By $\mathrm{Bern}(p)$ we denote the Bernoulli distribution, that is 1 with probability p and zero otherwise. By $\mathrm{Binom}(K, p)$ we denote the binomial distribution with parameters K and p (equal in the distribution to the sum of K independent copies of $\mathrm{Bern}(p)$).

3.2 Auxiliary Functions

We need the elementary properties of the two functions that often appear in our analysis:

▶ **Proposition 15.** *The function $g(d) = 1/q \cdot a^{1/q}$ for $q > 0$ is decreasing when $a \geqslant 1$ and for $a < 1$ it achieves its local maximum at $q = \log(1/a)$ with the value $g(q) = 1/\mathrm{e}\log(1/a)$.*

▶ **Proposition 16.** *The function $g(q) = q \cdot a^{1/q}$ for $q > 0$ is increasing when $a \leqslant 1$ and for $a > 1$ achieves its local minimum at $q = \log a$ with the value $g(q) = \mathrm{e}\log a$.*

3.3 Probabilistic Techniques

The following fact will allow us to handle non-symmetric distribitions.

▶ **Proposition 17** (Symmetrization trick [64])**.** *For any norm $\|\cdot\|$ we have*

$$\frac{1}{2}\|\sum_i X_i\sigma_i\| \leqslant \|\sum_i X_i\sigma_i\| \leqslant 2\|\sum_i X_i\sigma_i\|,$$

for any zero-mean independent X_i and independent Rademacher random variables σ_i.

We will also need the decoupling inequality, useful in attacking quadratic forms

▶ **Proposition 18** (Decoupling inequality [63]). *Let X_i be zero-mean independent r.vs. and X_i' be their independent copies. Then for any weights $a_{i,j}$*

$$\mathbf{E} f(\sum_{i\neq j} a_{i,j} X_i X_j) \leqslant \mathbf{E} f(4 \sum_{i\neq j} a_{i,j} X_i X_j'),$$

for any convex function f.

▶ Remark 19. The summation is over $i \neq j$, e.g. the quadratic form must be off-diagonal!

4 Proofs

4.1 Quadratic vs Linear Chaos

Proof of Lemma 6. Let X_i' be independent copies of X_i. The decoupling inequality gives

$$\|\sum_{i\neq j} X_i X_j\|_d \leqslant 4\|\sum_{i\neq j} X_i X_j'\|_d. \tag{14}$$

We apply the symmetrization trick to the d-th norm twice: first for random variables X_i with any fixed choice of X_j' which gives $\|\sum_{i\neq j} X_i X_j'\|_d \leqslant 2\|\sum_{i\neq j} X_i \sigma_i X_j'\|_d$ (here we use the independence of X_i and X_j') and second for random variables X_j' under the fixed values of $X_i\sigma_i$) which gives $\|\sum_{i\neq j} X_i X_j'\|_d \leqslant 4\|\sum_{i\neq j} X_i \sigma_i X_j' \sigma_j'\|_d$ (σ_j' is an independent Rademacher sequence). For simplicity, we denote $X_i := X_i\sigma_i$ and $X_j := X_j\sigma_j'$, note that the introduced random variables $X_i\sigma_i$ and $X_j\sigma_j$ are also identically distributed.

Consider the sum $\sum_{i,j} X_i X_j' = \sum_i (\sum_{j\neq i} X_j') X_i$ as linear in X_i with coefficients depending on X_j', and apply the multinomial theorem which gives

$$\mathbf{E}[(\sum_{i\neq j} X_i X_j')^d | (X_j')] = \sum_{(d_i)} \binom{d}{2d_1 \dots 2d_n} \prod_i (\sum_{j\neq i} X_j')^{2d_i} \mathbf{E} X_i^{2d_i}.$$

where we use the symmetry of X_i, so that all odd moments vanish. Again by the multinomial theorem we see that

$$\mathbf{E}(\sum_{j\neq i} X_j')^d \leqslant \mathbf{E}(\sum_j X_j')^d.$$

Combining the last two bounds gives

$$\begin{aligned}
\mathbf{E}(\sum_{i\neq j} X_i X_j')^d &\leqslant \mathbf{E}_{(X_j')}[\mathbf{E}[(\sum_{i\neq j} X_i X_j')^d | (X_j')]] \\
&\leqslant \sum_{(d_i)} \binom{d}{2d_1 \dots 2d_n} \mathbf{E}[\prod_i (\sum_j X_j')^{2d_i} X_i^{2d_i}] \\
&\leqslant \mathbf{E}(\sum_i (\sum_j X_j') X_i)^d \\
&= \mathbf{E}(\sum_i X_i)^d (\sum_j X_j')^d = \mathbf{E}(\sum_i X_i)^{2d},
\end{aligned}$$

which can be stated as

$$\|\sum_{i\neq j} X_i X_j'\|_d \leqslant \|\sum_i X_i\|_d^2. \tag{15}$$

By combining Equation (14) and Equation (15), and keeping in mind that X_i above are symmetrized versions, we obtain for original (only centered) random variables X_i

$$\mathbf{E}\|\sum_{j\neq i} X_i X_j\|_d \leqslant 16\mathbf{E}\|\sum_{j\neq i} X_i\sigma_i\|_d,$$

and the result follows by one more application of the symmetrization trick. ◀

4.2 Heterogenic vs Homogenic Chaos

Proof of Lemma 8. By the multinomial expansion and the symmetry of Z_i (which implies that the odd moments vanish) we obtain

$$\mathbf{E}(\sum_i x_i X_i)^d = \sum_{(d_i)} \binom{d}{2d_1\ldots 2d_n} p^{\|(d_i)\|_0}\prod_i x_i^{2d_i},$$

where the summation is over non-negative sequences (d_i) for $i=1,\ldots,n$ such that $\sum_i d_i = d/2$, and we denote $\|(d_i)\|_0 = \#\{i : d_i > 0\}$. Considering possible values of $k = \|(d_i)\|_0$, we find that the above expression is a non-negative combination of

$$S_k^{[d]}(x) = \sum_{i_1\neq\ldots\neq i_k} x_{i_1}^{2d_1}\ldots x_{i_k}^{2d_k}$$

where possible values of k are $1\leqslant k\leqslant \min(d/2, n_0)$ where $n_0 = \|(x_i)\|_0$. We now apply our assumption on x iteratively to $x_{i_k}, x_{i_{k-1}}\ldots$, obtaining

$$S_k^{[d]}(x) \leqslant v^{2\sum_{i:d_i>1}(d_i-1)} \sum_{i_1\neq\ldots\neq i_k} x_{i_1}^2\ldots x_{i_k}^2.$$

Here we have used the fact that $v_d(x)$ is increasing in d, so $v_k(x)\leqslant v$ when $k\leqslant d$; this follows from seeing $v_d(x)$ as the power mean of order $d-2$ and weights $x_i^2/\sum_{i\notin I} x_i^2$ [32, 66].

We make the following important observation: the equality holds whenever x_i is flat with the value v, e.g. all non-zero entries are equal to v. Observe that the sums $S_k(x) = \sum_{i_1\neq\ldots\neq i_k} x_{i_1}^2\ldots x_{i_k}^2$ are elementary symmetric polynomials in variables $y_i = x_i^2$ where $\sum_i y_i = \sum_i x_i^2 = 1$, hence over the probability simplex. The elementary symmetric functions are Schur-concave [17], and thus they are maximized at the uniform distribution, in our case when $x_i = n^{-1/2}$. In fact, $S_k(x)$ is the probability that k independent samples from the distribution $p_i = x_i^2$ do not collide. For any sequence (x_i^2) which has N non-zero equal entries and $\sum_i x_i^2 = 1$ we have that:

$$S_k(x) = N\cdot(N-1)\cdots(N-k+1)/N^k.$$

Since $N\geqslant k$ and since $k\leqslant d$, using Stirling's approximation [59] we obtain

$$S_k(x) = \prod_{i=0}^{k-1}(1-i/N) \geqslant k!/k^k = \Theta(1)^k \geqslant \Theta(1)^d.$$

Clearly $S_k(x)\leqslant 1$ for any x. If we replace (x_i) by a sequence such that $x_i = v$ for $K = v^{-2}$ values of i (e.g., flat), we lose at most a factor of $\Theta(1)^k\leqslant\Theta(1)^d$ in every term $S_k^{[d]}(x)$. ◀

4.3 Moments of IID Sums

Proof of Lemma 9. We have the following chain of estimates

$$\begin{aligned}\mathbf{E}(\sum_i X_i)^d &= \sum_{d_i:d_1+\ldots+d_n=d,d_i\geqslant 2} \binom{d}{d_1\ldots d_n}\prod_i \mathbf{E}X_i^{d_i}\\ &\leqslant \sum_{d_i:d_1+\ldots+d_n=d,d_i\geqslant 2}\prod_i \binom{d}{d_i}\mathbf{E}|X_i|^{d_i}\\ &\leqslant \sum_{d_i\geqslant 2}\prod_i \binom{d}{d_i}\mathbf{E}|X_i|^{d_i}\\ &\leqslant \left(\sum_{k=2}^{d}\binom{d}{k}\|X\|_k^k\right)^n.\end{aligned}$$

Applying this for $X_i := X_i/t$ we have for any $t > 0$

$$\mathbf{E}(t^{-1}\sum_i X_i)^d \leqslant \left(\sum_{k=2}^{d}\binom{d}{k}\|X\|_k^k/t^k\right)^n.$$

Thus $\|\sum_i X_i\|_d \leqslant \mathrm{e}t$ for any t such that the right-hand side is at most e, equivalently

$$\sum_{k=2}^{d}\binom{d}{k}\|X\|_k^k/t^k \leqslant \exp(d/n) - 1,$$

which is satisfied for

$$t = 2\max_{k=2\ldots d}\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k}\|X\|_k.$$

This proves the first part. Observe that for $k \geqslant 2$ we have

$$\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k} \leqslant \frac{\mathrm{e}d}{k\exp(d/kn)}\cdot\frac{1}{(1-\exp(-1))^{1/2}},$$

where we use the elementary inequalities $\binom{d}{k} \leqslant (d\mathrm{e}/k)^k$ and $\exp(u) - 1 \geqslant \exp(u)\cdot(1-\mathrm{e}^{-1})$ for $u \geqslant 1$. The function $u \to u/\exp(u)$ decreases for $u \geqslant 1$; applying this to $u = d/kn$ gives

$$\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k} \leqslant \frac{\mathrm{e}n}{(1-\mathrm{e}^{-1})^{1/2}}, \quad k \leqslant d/n.$$

Since $\|X\|_k$ increases in k we have

$$\max_{k=2\ldots d,k\leqslant d/n}\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k}\|X\|_k \leqslant \frac{\mathrm{e}n\|X\|_{d/n}}{(1-\mathrm{e}^{-1})^{1/2}}.$$

We have $(\exp(d/n)-1)^{-1/k} \leqslant (d/n)^{-1/k}$ due to the elementary inequality $\exp(u) - 1 \geqslant u$, and $\binom{d}{k} \leqslant (d\mathrm{e}/k)^k$ for any k. This gives

$$\max_{k=2\ldots d}\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k}\|X\|_k \leqslant \mathrm{e}\max_{k=2\ldots d} d/k\cdot(n/d)^{1/k}\cdot\|X\|_k$$

When $d/n \geqslant 2$ we have that $d/k\cdot(n/d)^{1/k}\cdot\|X\|_k = n\|X\|_{d/n}\cdot 2^{-1/2}$ for $k = d/n$. Comparing the last two equations, we obtain

$$\max_{k=2\ldots d,k\leqslant d/n}\binom{d}{k}^{1/k}(\exp(d/n)-1)^{-1/k}\|X\|_k \leqslant C\max_{k=2\ldots d,k>d/n} d/k\cdot(n/d)^{1/k}\cdot\|X\|_k,$$

with $C = \frac{\mathrm{e}}{(1-\mathrm{e}^{-1})^{1/2}}$. This completes the proof. ◀

4.4 Binomial Moments

Proof of Lemma 11. Applying Lemma 9 we obtain

$$\|S - \mathbf{E}S\|_d \leqslant O(1) \cdot \max\left\{(d/k) \cdot (Kp/d)^{1/k} : \max(2, d/K) \leqslant k \leqslant d\right\}.$$

because $S \sim \sum_i X_i$ where $X_i \sim \mathrm{Bern}(p)$ and $\|X_i - \mathbf{E}X_i\|_d = (p(1-p)^{d-1} + (1-p)p^{d-1})^{1/d}$ so that $\|X_i - \mathbf{E}X_i\|_d = \Theta(p)^{1/d}$ for $p \leqslant 1/2$.

The expression under the maximum is proportional to $k^{-1} \cdot a^{1/k}$ where $a = Kp/d$. The claim follows by applying Proposition 15, namely a) when $\max(2, d/K) \leqslant \log(1/a) \leqslant d$ (that is, inside the interval) we have necessarily $a \leqslant \mathrm{e}^{-2} < 1$ our maximum is at $k = \log(1/a)$ b) when $\log(1/a) > d$ we must have $a < 1$ and our maximum is at $k = d$ and c) when $\log(1/a) < \max(2, d/K)$ then the maximum is at $k = \max(2, d/K)$. ◀

5 Conclusion

We have proven novel bounds for sparse random projections, showing that the performance depends on the data statistic closed to *Renyi entropy*. Some intriguing problems we leave for future work are

- How do results extend to non-Rademacher matrices?
- Can we use majorization theory to fully characterize worst case for the linear chaos?

References

1 Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281, 2001.

2 Thomas D Ahle. Asymptotic tail bound and applications, 2017.

3 Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson–Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 557–563, 2006.

4 Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

5 SN Bernshtein. Probability theory (in Russian). *Gosizdat, Moscow-Leningrad*, 1927.

6 Gérard Biau, Luc Devroye, and Gábor Lugosi. On the performance of clustering in Hilbert spaces. *IEEE Transactions on Information Theory*, 54(2):781–790, 2008.

7 Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, 2001.

8 Henry W Block, Thomas H Savits, Moshe Shaked, et al. Some concepts of negative dependence. *The Annals of Probability*, 10(3):765–772, 1982.

9 Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The Johnson–Lindenstrauss transform itself preserves differential privacy. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 410–419. IEEE, 2012.

10 Stéphane Boucheron, Olivier Bousquet, Gábor Lugosi, Pascal Massart, et al. Moment inequalities for functions of independent random variables. *The Annals of Probability*, 33(2):514–560, 2005.

11 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

12 Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. Random projections for k-means clustering. In *Advances in Neural Information Processing Systems*, pages 298–306, 2010.

13 V Buldygin and K Moskvichova. The sub-gaussian norm of a binary random variable. *Theory of probability and mathematical statistics*, 86:33–49, 2013.

14 Christian Cachin. Smooth entropy and rényi entropy. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 193–208. Springer, 1997. URL: `https://link.springer.com/chapter/10.1007/3-540-69053-0_14`.

15 Herman Chernoff et al. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

16 Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):1–45, 2017.

17 M Lawrence Clevenson and William Watkins. Majorization and the birthday inequality. *Mathematics Magazine*, 64(3):183–188, 1991.

18 Michael B Cohen. Nearly tight oblivious subspace embeddings by trace inequalities. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 278–287. SIAM, 2016.

19 Michael B Cohen, TS Jayram, and Jelani Nelson. Simple analyses of the sparse Johnson–Lindenstrauss transform. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

20 Harald Cramér. On a new limit theorem of the theory of probability. *Uspekhi Matematicheskikh Nauk*, 10:166–178, 1944.

21 Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A sparse Johnson-Lindenstrauss transform. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 341–350, 2010.

22 Sanjoy Dasgupta. Learning mixtures of gaussians. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 634–644. IEEE, 1999.

23 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report*, 22(1):1–5, 1999.

24 Victor H de la Peña and Stephen J Montgomery-Smith. Decoupling inequalities for the tail probabilities of multivariate u-statistics. *The Annals of Probability*, pages 806–816, 1995.

25 Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

26 Morris L Eaton. A note on symmetric bernoulli random variables. *The annals of mathematical statistics*, 41(4):1223–1226, 1970.

27 Cees M Fortuin, Pieter W Kasteleyn, and Jean Ginibre. Correlation inequalities on some partially ordered sets. *Communications in Mathematical Physics*, 22(2):89–103, 1971.

28 Peter Frankl and Hiroshi Maehara. The Johnson–Lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory, Series B*, 44(3):355–362, 1988.

29 Peter Frankl and Hiroshi Maehara. Some geometric applications of the beta distribution. *Annals of the Institute of Statistical Mathematics*, 42(3):463–474, 1990.

30 Casper B Freksen, Lior Kamma, and Kasper Green Larsen. Fully understanding the hashing trick. In *Advances in Neural Information Processing Systems*, pages 5389–5399, 2018.

31 David Lee Hanson and Farroll Tim Wright. A bound on tail probabilities for quadratic forms in independent random variables. *The Annals of Mathematical Statistics*, 42(3):1079–1083, 1971.

32 G.H. Hardy, Karreman Mathematics Research Collection, J.E. Littlewood, G. Pólya, G. Pólya, and D.E. Littlewood. *Inequalities*. Cambridge Mathematical Library. Cambridge University Press, 1952. URL: `https://books.google.at/books?id=t1RCSP8YKt8C`.

33 Paweł Hitczenko. Domination inequality for martingale transforms of a rademacher sequence. *Israel Journal of Mathematics*, 84(1-2):161–178, 1993.

34 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

35 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

36 Meena Jagadeesan. Understanding sparse jl for feature hashing. In *Advances in Neural Information Processing Systems*, pages 15203–15213, 2019. `arXiv:1903.03605`.

37 Thathachar S Jayram and David P Woodruff. Optimal bounds for Johnson–Lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)*, 9(3):1–17, 2013.

38 William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

39 William B Johnson and Assaf Naor. The Johnson–Lindenstrauss lemma almost characterizes Hilbert space, but not quite. *Discrete & Computational Geometry*, 43(3):542–553, 2010.

40 Daniel Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit Johnson–Lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 628–639. Springer, 2011.

41 Daniel M Kane and Jelani Nelson. Sparser Johnson–Lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

42 Krishnaram Kenthapadi, Aleksandra Korolova, Ilya Mironov, and Nina Mishra. Privacy via the Johnson–Lindenstrauss transform. *Journal of Privacy and Confidentiality*, 5(1):39–71, 2013.

43 Michael Kerber and Sharath Raghvendra. Approximation and streaming algorithms for projective clustering via random projections. *arXiv preprint*, 2014. `arXiv:1407.2063`.

44 Aleksandr Khintchine. Über dyadische brüche. *Mathematische Zeitschrift*, 18(1):109–116, 1923.

45 Andreas Knoblauch. Closed-form expressions for the moments of the binomial probability distribution. *SIAM Journal on Applied Mathematics*, 69(1):197–204, 2008.

46 Konrad Kolesko and Rafał Latała. Moment estimates for chaoses generated by symmetric random variables with logarithmically convex tails. *Statistics & Probability Letters*, 107:210–214, 2015.

47 Samory Kpotufe and Bharath Sriperumbudur. Gaussian sketching yields a jl lemma in rkhs. In *International Conference on Artificial Intelligence and Statistics*, pages 3928–3937, 2020.

48 Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

49 Rafał Latała. Tail and moment estimates for some types of chaos. *Studia mathematica*, 135(1):39–53, 1999.

50 Rafał Latała et al. Estimation of moments of sums of independent real random variables. *The Annals of Probability*, 25(3):1502–1513, 1997.

51 Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296, 2006.

52 Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

53 John E Littlewood. On the probability in the tail of a binomial distribution. *Advances in Applied Probability*, 1(1):43–72, 1969.

54 Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of Johnson–Lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1027–1038, 2019.

55 Jiří Matoušek. On variants of the Johnson–Lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

56 Brendan D McKay. On littlewood's estimate for the binomial distribution. *Advances in Applied Probability*, 21(2):475–478, 1989.

57 Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 ieee 54th annual symposium on foundations of computer science*, pages 117–126. IEEE, 2013.

58 Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

59 Herbert Robbins. A remark on Stirling's formula. *The American mathematical monthly*, 62(1):26–29, 1955.

60 Mark Rudelson, Roman Vershynin, et al. Hanson-wright inequality and sub-gaussian concentration. *Electronic Communications in Probability*, 18, 2013.

61 Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 143–152. IEEE, 2006.

62 Pantelimon Stanica. Good lower and upper bounds on binomial coefficients. *Journal of Inequalities in Pure and Applied Mathematics*, 2(3):30, 2001.

63 Roman Vershynin. A simple decoupling inequality in probability theory. *preprint*, 2011.

64 Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

65 Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

66 Alfred Witkowski. A new proof of the monotonicity of power means. *J. Ineq. Pure and Appl. Math*, 5(1), 2004.

67 Shuheng Zhou. Sparse Hanson–Wright inequalities for subgaussian quadratic forms. *Bernoulli*, 25(3):1603–1639, 2019. appears in 2015 at `arXiv:1510.05517`.

A Some remarks on prior works

A.1 Some issues with numeric constants

Lemma 2.1 in [36] gives the following bound (expressed in our notation)

$$\|Z_r\|_d \lesssim \begin{cases} dp & d = 2 \text{ or } d \leqslant pe/v^2 \\ \min\left(\frac{d^2v^2}{\log(dv^2/p)}, \frac{d}{\log(1/p)}\right) & 1 \leqslant \log(dv^2/p) \leqslant d \\ v^2(p/dv^2)^{2/d} & d < \log(dv^2/p) \end{cases}$$

There is a minor mistake in splitting the branches: they emerge from taking the derivative test of the function $d^2v^2u^{-2}(p/dv^2)^{1/u}$ where $1 \leqslant u \leqslant d/2$ (Lemma D.1). Here the local maxima occurs at $u = \log(dv^2/p)/2$ and when comparing this with edges $u = 1$ and $u = d/2$ we obtain the conditions $2 \leqslant \log(dv^2/p)$ and $\log(dv^2/p) \leqslant d$. Thus, the splitting conditions should be a bit different; this particular issue doesn't affect the bounds expressed in the asymptotic notation; we report it with intent to motivate our effort in giving a simple and clear proof.

A.2 Gaps in symmetrization

Section 2.2 of [36], when explaining the proof strategy, proposes to apply the bounds on Z_r defined in Equation (10) assuming they are symmetric. But Z_r are not symmetric (it is easy to see they have positive higher-order moments), thus extra work is needed to push this argument forward.

B Concluding Main Theorem

Without loosing generality, we assume that $d = \log(1/\delta)$ is even. Recall that we denote $v = v_d(x)$, also without loosing generality we assume that v^{-2} is an integer. For $K = v^{-2}$ define the following quantities

$$I_1 \triangleq \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot qp : \log(q/Kp) \leqslant q/K \leqslant 2, 2 \leqslant q \leqslant d \right\}$$
$$I_2 \triangleq \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot K(Kp^{2K/q})^2 : \log(q/Kp) \leqslant 2 \leqslant q/K, 2 \leqslant q \leqslant d \right\}$$
$$I_3 \triangleq \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot K^{-1}q^2/\log^2(q/Kp) : \max(2, q/K) \leqslant \log(q/Kp) \leqslant q, 2 \leqslant q \leqslant d \right\}$$
$$I_4 \triangleq \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot K^{-1}(Kp)^{2/q} : q \leqslant \log(q/Kp), 2 \leqslant q \leqslant d \right\}.$$

Following the proof outline we arrive at Corollary 13. Taking into account Lemma 11 and Lemma 9, implies that:

$$\|\sum_{r=1}^{m} Z_r\|_d \leqslant O(\max(I_1, I_2, I_3, I_4)).$$

The goal is to prove that for $t = s\epsilon$ we have

$$\|\sum_{r=1}^{m} Z_r\|_d \leqslant t/\mathrm{e}, \tag{16}$$

and then the result follows from Markov's inequality. We give first bounds for I_1, I_2, I_4 as they are fairly easy to obtain. The case of I_3 is analyzed as the last one.

B.1 First Branch

We will show the following bound

▶ **Lemma 20.** *We have*

$$I_1 \leqslant O(dmp^2)^{1/2}.$$

Proof of Lemma 20. We have

$$\begin{aligned} I_1 &= \max_q \left\{ pd(m/d)^{1/q} : \log(q/Kp) \leqslant q/K \leqslant 2, 2 \leqslant q \leqslant d \right\} \\ &\leqslant (dmp^2)^{1/2} \end{aligned}$$

where the inequality follows because $m \geqslant d$ and $1/q \leqslant \frac{1}{2}$ (for q satisfying the constraints). This completes the proof. ◀

B.2 Second Branch

We will show the following bound

▶ **Lemma 21.** *For $p \leqslant 2\mathrm{e}^{-2}$ we have*

$$I_2 \leqslant (dmp^2)^{1/2}.$$

Proof of Lemma 20. For q satisfying the constraint we have $K/q \geqslant \mathrm{e}^{-2}/p$ which, due to $p \leqslant 2\mathrm{e}^{-2}$, implies $K/q \geqslant 1/2$. Then $p^{2K/q} \leqslant p$ (recall that $p < 1$!) and thus

$$I_2 \leqslant \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot Kp : \log(q/Kp) \leqslant 2 \leqslant q/K, 2 \leqslant q \leqslant d \right\}.$$

For q within the constraints we have $K/q \leqslant \frac{1}{2}$ and therefore

$$I_2 \leqslant \frac{p}{2} \max_q \left\{ d \cdot (m/d)^{1/q} : \log(q/Kp) \leqslant 2 \leqslant q/K, 2 \leqslant q \leqslant d \right\}.$$

Since $m/d \geqslant 1$ the expression under the maximum decreases with q, thus is not bigger than the value at $q = 2$. Thus, $I_2 \leqslant p(dm)^{1/2}/2$ and the result follows. ◀

B.3 Fourth Branch

We will prove the following bound

▶ **Lemma 22.** *We have*

$$I_4 \leqslant \begin{cases} (dmp^2)^{1/2} & \log(dv^4/mp^2) \leqslant 2 \\ dv^2/\log(dv^4/mp^2) & \log(dv^4/mp^2) > 2 \end{cases}.$$

Proof of Lemma 22. We have

$$I_4 = \max_q \left\{ K^{-1} \cdot d/q \cdot (K^2p^2m/d)^{1/q} : q \leqslant \log(q/Kp), 2 \leqslant q \leqslant d \right\}.$$

Let $a = K^2p^2m/d$, the expression under the maximum is proportional to $1/q \cdot a^{1/q}$. We now apply Proposition 15: for $a \geqslant 1$ the maximum is not bigger than the value at $q = 2$, so

$$I_4 \leqslant (dmp^2)^{1/2}.$$

We now can assume $a < 1$, equivalent to $K^2p^2m < d$. The global maximum is at $q = \log(1/a)$, thus our maximum is still at $q = 2$ when $\log(1/a) \leqslant 2$ and otherwise is not bigger than the value at $q = \log(1/a)$. We then obtain

$$I_4 \leqslant K^{-1}d/\log(d/mp^2K^2) \leqslant K^{-1}d = dv^2.$$

This complete the proof. ◀

B.4 Third Branch

We will show the following bound

▶ **Lemma 23.** *Suppose that $v^2 \geqslant s\epsilon/d^2$, then*

$$I_3 \leqslant O(dmp^2)^{1/2} + O(dv/\log(dv^2/p))^2$$

Proof of Lemma 23. The proof is based on splitting the maximum into three regimes: $q \in [2, 3]$,$3 \leqslant q \leqslant \log(m/d)$ and $\log(m/d) \leqslant q \leqslant d$. Define

$$I^0 = \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot v^2q^2/\log^2(qv^2/p) : 2 \leqslant \log(qv^2/p) \leqslant q \leqslant d, 2 \leqslant q \leqslant 3 \right\}$$

$$I^- = \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot v^2q^2/\log^2(qv^2/p) : 2 \leqslant \log(qv^2/p) \leqslant q \leqslant d, 3 \leqslant q \leqslant \log(m/d) \right\}$$

$$I^+ = \max_q \left\{ d/q \cdot (m/d)^{1/q} \cdot v^2q^2/\log^2(qv^2/p) : 2 \leqslant \log(qv^2/p) \leqslant q \leqslant d, \log(m/d) \leqslant q \leqslant d \right\}.$$

so that we have $I_3 \leqslant \max(I^0, I^+, I^-)$ (for convenience, we replace the constraint $\max(2, qv^2) \leqslant \log(qv^2/p)$ in I_3 by the weaker one $2 \leqslant \log(qv^2/p)$). By the assumptions we have $v^2/p \geqslant m\epsilon/d^2$. Since $m \geqslant d\epsilon^{-2}$ we have $\epsilon \geqslant (d/m)^{1/2}$, and thus

$$v^2/p \geqslant (m/d)^{1/2} \cdot d^{-1}.$$

▷ Claim 24. We have $I^- \leqslant O(d^2v^2/\log^2(dv^2/p)$ when $\log d \leqslant \frac{5\log(m/d)}{12}$.

Proof of Claim. For any q satisfying the restrictions it holds that

$$\begin{aligned} q &\geqslant \log(v^2/p) \\ &\geqslant \frac{\log(m/d)}{2} - \log d \\ &\geqslant \frac{\log(m/d)}{12}. \end{aligned}$$

We then have $(m/d)^{1/q} \leqslant O(1)$ and thus

$$I^- \leqslant \max_q \left\{ d \cdot qv^2/\log^2(qv^2/p) : 2 \leqslant \log(qv^2/p) \leqslant q \leqslant d, 3 \leqslant q \leqslant \log(m/d) \right\}.$$

Considering the auxiliary function $u \to u/\log^2 u$ with $u = qv^2/p \geqslant \mathrm{e}^2$, we see that it decreases in u and hence in q for fixed v^2 and p. The expression is thus not smaller than its value at $q = d$, which gives

$$I^- \leqslant d^2v^2/\log^2(dv^2/p),$$

and completes the proof. ◁

▷ Claim 25. We have $I^- \leqslant d^2v^2/\log^2(dv^2/p)$ when $\log d > \frac{5\log(m/d)}{12}$.

Proof of Claim. We have that $dv^2/p \geqslant m\epsilon/d \geqslant (m/d)^{1/2}$ and therefore

$$\begin{aligned} I^- &\leqslant dv^2 d(m/d)^{1/3}\log(m/d) \\ &\leqslant dv^2(m/d)^{5/12}/\log^2(m/d) \\ &\leqslant dv^2(m/d)^{5/12}/\log^2(dv^2/p) \\ &\leqslant O(d^2v^2/\log^2(dv^2/p)), \end{aligned}$$

which completes the proof. ◁

▷ Claim 26. We have $I^+ \leqslant O(d^2v^2/\log^2(dv^2/p))$

Proof of Claim. We have $(m/d)^{1/q} \leqslant \mathrm{e}$ for $q \geqslant \log(m/d)$, thus

$$I^+ \leqslant d \cdot \max_q \{qv^2/\log^2(qv^2/p) : \ 2 \leqslant \max(\log(qv^2/p), \log(m/d)) \leqslant q \leqslant d\}.$$

Considering the auxiliary function $u \to u/\log^2 u$ with $u = qv^2/p \geqslant \mathrm{e}^2$, we see that it decreases in u and hence in q for fixed v^2 and p. The expression is thus not smaller than its value at $q = d$, which gives

$$I^+ \leqslant O(d^2v^2/\log^2(dv^2/p))$$

and the claim follows. ◁

▷ Claim 27. We have $I^0 \leqslant O((dmp^2)^{1/2})$.

Proof of Claim. We have $I^0 \leqslant O(v^2(md)^{1/2})$ because $(m/d)^{1/q} \leqslant (m/d)^{1/2}$ (due to $m/d \geqslant 1$ and $q \geqslant 2$). However, for $q \in [2,3]$ the constraint $\log(qv^2/p) \leqslant q$ gives $v^2 \leqslant O(p)$. Thus

$$I^0 \leqslant O(p(md)^{1/2}),$$

which completes the proof. ◁

The result follows now by combining the above three claims. ◀

B.5 Merging Branch Bounds

To conclude the main result it suffices to satisfy

$$c \cdot \max(I_1, I_2, I_3, I_4) \leqslant s\epsilon \tag{17}$$

for some absolute constant c. The condition in Equation (17) for I_1, I_2 is equivalent to $c \cdot (dmp^2)^{1/2} \leqslant s\epsilon$, which holds when

$$m \geqslant \Omega(d\epsilon^{-2}). \tag{18}$$

To satisfy Equation (17) for I_4 we require, in addition to Equation (18), that $cdv^2 \leqslant s\epsilon$, equivalent to

$$v \leqslant O((s\epsilon)^{1/2}/d^{1/2}). \tag{19}$$

Finally, in order to satisfy Equation (17) for I_3 we observe that, under the restriction

$$v^2 \geqslant s\epsilon/d^2, \tag{20}$$

the bound in Lemma 23 gives

$$I_3 \leqslant O(dmp^2)^{1/2} + O(dv/\log(m\epsilon/d))^2,$$

which follows because $\log(dv^2/p) \geqslant \log(s\epsilon/dp) = \log(m\epsilon/d)$. Thus, in addition to Equation (18) and Equation (20) it suffices that

$$v \leqslant O((s\epsilon)^{1/2}\log(m\epsilon/d)/d). \tag{21}$$

Now observe that for

$$v = \Theta(s\epsilon)^{1/2}\min(\log(m\epsilon/d)/d, 1/d^{1/2}) \tag{22}$$

the condition in Equation (20) is automatically satisfied. Thus, the theorem holds for v as above, and clearly for any smaller v.

Evacuation from a Disk for Robots with Asymmetric Communication

Konstantinos Georgiou
Department of Mathematics, Toronto Metropolitan University, Canada

Nikos Giachoudis
Department of Mathematics, Toronto Metropolitan University, Canada

Evangelos Kranakis
School of Computer Science, Carleton University, Ottawa, Canada

Abstract

We consider evacuation of two robots from an Exit placed at an unknown location on the perimeter of a unit (radius) disk. The robots can move with max speed 1 and start at the center of the disk at the same time. We consider a new communication model, known as the SR model, in which the robots have communication faults as follows: one of the robots is a Sender and can only send wirelessly at any distance, while the other is a Receiver in that it can only receive wirelessly from any distance. The communication status of each robot is known to the other robot. In addition, both robots can exchange messages when they are co-located, which is known as Face-to-Face (F2F) model.

There have been several studies in the literature concerning the evacuation time when both robots may employ either F2F or Wireless (WiFi) communication. The SR communication model diverges from these two in that the two robots themselves have differing communication capabilities. We study the evacuation time, namely the time it takes until the last robot reaches the Exit, and show that the evacuation time in the SR model is strictly between the F2F and the WiFi models. The main part of our technical contribution is also an evacuation algorithm in which two cooperating robots accomplish the task in worst-case time at most $\pi + 2$. Interesting features of the proposed algorithm are the asymmetry inherent in the resulting trajectories, as well as that the robots do not move at full speed for the entire duration of their trajectories.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Communication, Cycle, Evacuation, Receiver, Sender, Mobile Agents

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.19

Funding *Konstantinos Georgiou*: Research supported in part by NSERC.
Nikos Giachoudis: Research supported in part by the Fields Institute and the Faculty of Science, Ryerson University.
Evangelos Kranakis: Research supported in part by NSERC.

1 Introduction

Evacuating a group of autonomous mobile agents (or robots) operating over a specified planar domain (e.g., circle, square, convex set, etc.) from an unknown (to the robots) Exit is an important paradigm in group search for understanding the tradeoffs of communication and mobility in distributed computing. The communication capabilities of the robots that have been considered in the literature so far are limited mainly to Wireless (WiFi) and Face-to-Face (F2F), see [2]. An important aspect in this type of group search is the ability of the robots to evacuate despite the presence of communication faults. Two types of faults have been considered in the literature: crash (passive and non-malicious) and Byzantine (active and malicious). Note that in our model only the robots' communication capabilities are affected and both robots are honest (i.e., they send correct messages and the communication

is always available). Despite its importance, fault tolerant group search has been considered less extensively mainly due to the complexity of the resulting optimization problem for this task (see [3]).

In this paper we explore two robot evacuation in the plane under a different type of faulty behaviour whereby the robots have different communication capabilities, namely one robot is a Sender in that it can only send messages wirelessly and the other is a Receiver in that it can only receive messages wirelessly. For this setting we obtain the following contributions. First we show that any algorithm is bound to have running time strictly more than the provably optimal evacuation time of two Wireless robots. Second, we provide a novel evacuation algorithm whose running time is better than the best lower bound known for two Face-to-Face robots, separating this way the two well-studied models.

1.1 Preliminaries and notation

Two robots are initially collocated in the plane and can travel with speed at most 1. An Exit is placed at known distance 1 from the robots at an unknown location (therefore defining a disk with center the original position of the robots). The robots can always communicate F2F (when co-located). In addition, one of the robots is a Sender (i.e., can send messages wirelessly, but cannot receive messages from distance) and the other is a Receiver (i.e., can receive wirelessly, but cannot send messages from distance). The communication capabilities of the robots are known to themselves and to each other, namely the robots know who is the Sender and who is the Receiver and moreover the Sender knows it can send and the Receiver it can receive messages wirelessly, the messages are transferred instantly. The goal is to design an evacuation algorithm which specifies trajectories which enable the robots to reach the unknown Exit. During their trajectories the robots can take shortcuts in the interior of the disk and can recognize its perimeter. The quality of the algorithm is measured by the time it takes the last robot to reach the Exit, which is defined as the evacuation time (of the algorithm).

Next we define more precisely the concept of evacuation time restricted to our search domain, namely the unit disk, which will be used in the rest of the paper.

▶ **Definition 1.** *The evacuation time $\mathcal{E}_A(p)$ of an algorithm A for an Exit placed at an unknown (to the robots) location p on the perimeter of the unit disk is the time it takes for the two robots starting from the centre of the unit disk and following algorithm A to evacuate from the Exit placed at p. The worst case evacuation time of an algorithm A is defined as the $\sup_p \mathcal{E}_A(p)$, where p may be any point on the perimeter of the unit circle.*

Consider a class $\mathcal{A}$ of all possible evacuation algorithms on the unit disk arising from the chosen communication model (e.g., F2F, WiFi, etc.).

▶ **Definition 2.** *The evacuation time for a class $\mathcal{A}$ of algorithms is defined as $\inf_{A\in\mathcal{A}} \mathcal{E}_A$.*

For the purposes of the current work, a robot may be in one of four possible communication states, namely F2F (Face-to-Face), WiFi (Wireless, both send and receive), S (only send wirelessly), and R (only receive wirelessly). All previous studies considered evacuation in which the robots have identical communication capabilities (e.g., both F2F, or both WiFi). Our focus in this paper is to consider robots with different communication capabilities. Before proceeding any further it will be useful to make an observation which clarifies the communication potential of an ensemble of two evacuating robots assuming that they both maintain their F2F (both send and receive only when co-located) communication capability.

▶ **Observation 3.** *There are only three possibilities for the communication capabilities of an ensemble of two robots which are (at least) able to communicate F2F: either both are WiFi, or both are only F2F, or one is a Sender and the other a Receiver.*

To see why this is true, recall that by assumption, at a minimum the robots are assumed to be able to communicate F2F. Moreover, one can think of a robot with WiFi communication potential as a robot which is both Sender and Receiver. If one of the robots can only communicate F2F then regardless of the communication capabilities the other robot has, the ensemble of two robots behaves as if both robots have only F2F communication. Similarly, if both robots are Senders or both are Receivers; the robots can communicate only F2F with each other. If one of the robots has WiFi communication but the other is only either a Sender or Receiver then the WiFi cannot use its full communication potential and therefore can only use its Sender status with a Receiver and its Receiver status with a Sender. Thus the validity of the observation follows by combining all three previous assertions.

Let $\mathcal{E}_{F2F}$, resp. $\mathcal{E}_{WiFi}$, be the evacuation times when both robots use F2F, resp. WiFi, communication. Further, let SR denote the mixed Sender and Receiver model described above. From previous studies we know that $\mathcal{E}_{WiFi} < \mathcal{E}_{F2F}$. The main questions of interest are the following:

1. Can we find an optimal evacuation algorithm in the Sender/Receiver model?
2. It is easy to see that the evacuation time in these models satisfies $\mathcal{E}_{WiFi} \leq \mathcal{E}_{SR} \leq \mathcal{E}_{F2F}$. Can we differentiate the three models and show that the inequalities are strict?

It turns out that this communication asymmetry between Sender and Receiver gives rise to interesting trade-offs which are unique to the SR model. Regarding the proposed questions, we answer them as follows.

1. We provide a technical algorithm with evacuation time strictly less than $\mathcal{E}_{F2F}$ and that we conjecture is optimal.
2. We show that any algorithm is bound to have evacuation time strictly more than $\mathcal{E}_{WiFi}$.

1.2 Mobility Model

We use the standard evacuation model for search on a unit disk where the Exit is placed on the perimeter and the robots may take short-cuts in its interior. The robots are autonomous and can exchange messages instantly using the SR communication model. The robots run synchronized clocks. They know the unit disk (its center and unit radius) and can recognize its perimeter. At any time they can stop and start, change direction and speed but their speed can never exceed 1. An evacuation algorithm is defined by the trajectories of the two robots. A robot trajectory is a continuous function $f : [0, T] \to \mathbb{R}^2$ over time, such that $f(t)$ is the location of the robot at time t and T is the duration of a robot's trajectory. Moreover, the robot's speed can never exceed 1, meaning that $\|f(t) - f(t')\|_2 \leq |t - t'|$, for all $0 \leq t, t' \leq T$, where $\|\cdot\|_2$ denotes the Euclidean norm in the plane $\mathbb{R}^2$.

1.3 Related work

In all the results below we consider worst-case evacuation time and limit our discussion mostly to the case of two robots. The general evacuation model on a disk discussed in our paper was first introduced in [2]. In this paper, among other results, the optimal value for the WiFi model was determined, namely $\mathcal{E}_{WiFi} = 1 + \frac{2\pi}{3} + \sqrt{3} \approx 4.826$, while for the value of the F2F model it was shown that $5.199 \approx 3 + \frac{\pi}{4} + \sqrt{2} \leq \mathcal{E}_{F2F} \leq 5.74$. The lower bound was later improved to $3 + \frac{\pi}{6} + \sqrt{3} \approx 5.255 \leq \mathcal{E}_{F2F}$ in [6]. Most recently, [12] improving on

results of [1], has improved the upper bound from 5.74 (which was first proved in [2]) to $\mathcal{E}_{F2F} \leq 5.6234$. However, it is worth noting that, in general, worst-case tight bounds for evacuation of two robots in the F2F model remain elusive to this day.

Search and evacuation with multiple faulty robots on an infinite line was initiated in the work of [5] and [11]. For three robots one of which is Byzantine, [16] shows that the proportional schedule presented in [11] can be analyzed to achieve an upper bound of 8.653055. More importantly, [10] gives a new class of algorithms for n robots when the number of Byzantine among them is near majority, which in turn implies the best known upper bound of 7.437011 on an infinite line for three robots one of which is Byzantine.

There is also a limited number of studies for the unit disk. In [3] the authors consider evacuation of three robots from a disk one of which may be a crash or Byzantine faulty robot. For the case of crash faults the lower bound achieved is ≈ 5.188 and the upper bound ≈ 6.309, while for Byzantine faults the lower bound is ≈ 5.948 and the upper bound ≈ 6.921. In [13] and [15] the authors consider search on a disk for n robots at most f of which may be faulty, i.e., crash or Byzantine (it turns out that search is simpler to analyze than evacuation since success is achieved when the first non-faulty robot finds the Exit.) In addition, [8] studies search for n robots in the plane with faulty robots; in this model the robots have arbitrary (not necessarily identical) max speeds and visibility ranges. Moreover, in [14] the authors study the problem on the disk with two robots one of which can have speed more than 1, while the robots can communicate wirelessly. Finally, [7] presents a survey of group search which could be useful to the reader.

The SR mixed communication model studied in our current paper was considered as a way to model group search in the presence of robots with faulty communication capabilities and was first introduced for an infinite line in [9], but otherwise has never been studied in any other domain, such as the unit disk.

1.4 Results of the paper

We show that evacuation in which one robot is a Sender and the other is a Receiver is more powerful than evacuation when both robots use F2F communication and less powerful when both robots use WiFi communication. We give upper and lower bounds for the evacuation time of two robots in the SR communication model. For the upper bound, see Section 2, we design a new evacuation algorithm which leads to evacuation time at most $\pi + 2$. The novelty of our upper bound pertains also to the asymmetry of the proposed algorithm, as well as that the robots follow trajectories in which their speed is not always 1. A similar feature was deployed previously only in the F2F model for searching on the line [4], but with an objective different than minimizing the evacuation time. For the lower bound, see Section 3, we show that there is no evacuation algorithm in the SR model whose evacuation time is equal to the optimal evacuation time in the WiFi model.

2 The Upper Bound of $2 + \pi$

In the present section we give an evacuation algorithm in the SR model and analyze its competitive ratio. The main result is the following.

▶ **Theorem 4.** *Evacuation of two robots from a disk in the Sender Receiver communication model can be accomplished in time* $2 + \pi$.

For the proof of Theorem 4 we design trajectories for the robots and show that for every placement of the Exit on the unit circle, the evacuation time is at most $2+\pi$. For convenience we think of the search domain, the unit circle, in a Cartesian system, i.e. $\mathcal{C} = \{\text{CYCLE}(t), t \in [0, 2\pi]\}$, where $\text{CYCLE}(t) = (\cos t, \sin t)$ is a parametric description of the unit circle. Note that both robots start from the center of circle $\mathcal{C}$.

We give trajectories for the robots in two steps, parameterized by the same function $\alpha\colon [0,\pi] \mapsto \mathbb{R}$ (to be fixed later). For notational convenience, we occasionally write α_t instead of $\alpha(t)$. The trajectories depend on when (and if) robots find the Exit at some point $\text{CYCLE}(t)$. The description below uses the assumption that should the Sender locate the Exit in $\text{CYCLE}(t)$ then the information will be transmitted to the Receiver instantaneously and the Receiver will abandon her trajectory in order to reach $\text{CYCLE}(t)$ in the fastest possible way (along a line segment). The time that the Receiver will reach $\text{CYCLE}(t)$ will be the total evacuation time for that placement of the Exit. Therefore, the Sender's trajectory needs to be described only under the assumption that no Exit has been found by her. The trajectory of the Sender will be fully determined once we fix the aforementioned function $\alpha(\cdot)$. It is worthwhile mentioning in advance that the Sender's trajectory will not be unit speed at all times during the execution of the algorithm.

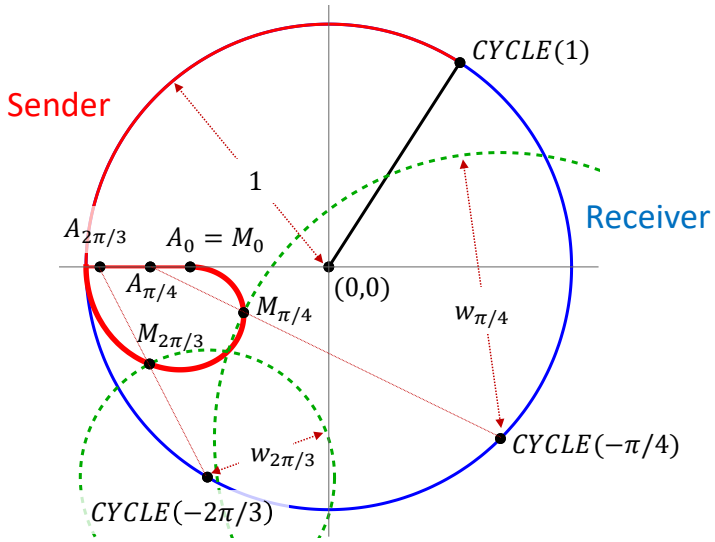


Figure 1 Robots' trajectories, for the optimal choice of parameter function $\alpha(\cdot)$. Red and blue are the Sender's and Receiver's trajectories respectively (assuming no Exit is found). The thick red curve corresponds to the Sender's partial trajectory M_t, $t = 0, \ldots, \pi$. Note that when the Sender reaches point $(-1, 0)$ she starts moving towards the center $(0, 0)$. Figure also depicts two examples of Exit placements, $\text{CYCLE}(-\pi/4)$, $\text{CYCLE}(-2\pi/3)$, along with the corresponding circles with these centers and radii $w_{\pi/4}, w_{-2\pi/3}$, along with the directive points $A_{\pi/4}, A_{-2\pi/3}$ (indicating the direction the Receiver attempts to meet the Sender), along with the intended meeting points $M_{\pi/4}, M_{-2\pi/3}$, respectively.

In contrast, the Receiver's trajectory will depend on whether she has found the Exit or not. If the Receiver finds the Exit at $\text{CYCLE}(t)$, then starting from $\text{CYCLE}(t)$ she will follow a trajectory in order to meet the Sender in her trajectory so that together they return to the Exit in time at most $2+\pi$ in total. The Receiver's trajectory will be determined uniquely once we fix function $\alpha(\cdot)$, but the Receiver will maintain a unit speed at all times (except from some placements of the Exit, in which case the Receiver will wait idle in some point in

order to meet the Sender and return together to the Exit). The description of our algorithm is coupled with the illustration of Figure 1 that we explain as we present the technicalities of the algorithm.

2.1 Robots' Trajectories

We now give more formally the robots' trajectories and the main algorithm of the paper.

Algorithm 1 Sender-Receiver Search Algorithm.

1: **Sender's Trajectory (input: speed compliant SPT pair $(\mathcal{S}, \tau)$)**
2: Phase S1: At unit speed go to $\textsc{cycle}(1)$.
3: Phase S2: At unit speed, search $\mathcal{C}$ counter-clockwise up to point $\textsc{cycle}(\pi)$, or until exit is found. If exit is found at $\textsc{cycle}(x)$ then send the Receiver the position for her to come to the exit and wait.
4: Phase S3: At unit speed go to point $(1-\pi/2, 0)$.
5: Phase S4: Reset the clock and traverse curve $\mathcal{S}$ with timing rule τ.

1: **Receiver's Trajectory (input: reserved function $\alpha(\cdot)$)**
2: Phase R1: At unit speed go to $\textsc{cycle}(1)$.
3: Phase R2: Search $\mathcal{C}$ clockwise towards point $\textsc{cycle}(\pi)$. If Exit is found at $\textsc{cycle}(t)$, $t \in [0,1]$, start R3. If Exit found at $\textsc{cycle}(-t)$, $t \in (0,\pi]$, start R4.
4: Phase R3: Go to $(1-\pi/2, 0)$ and wait for Sender, then return to $\textsc{cycle}(t)$.
5: Phase R4: Move toward $A_t := (\alpha(t), 0)$ for time $w_t := (\pi - t)/2$ (to meet the Sender at point M_t) and return to $\textsc{cycle}(-t)$.

2.1.1 Sender's Trajectory

Let $f, g : [0,\pi] \mapsto \mathbb{R}$ be continuous and differentiable real functions defining a closed curve $\mathcal{S} := \{(f(s), g(s)), s \in [0,\pi]\}$. Consider also continuous and differentiable $\tau : [0, \pi/2] \mapsto \mathcal{S}$ (that we will call the *timing-rule* for $\mathcal{S}$). The pair $(\mathcal{S}, \tau)$ will be called SPT (Sender's Partial Trajectory) if the following conditions are satisfied:

$$\lim_{t\to 0} f(t) = 1 - \pi/2, \lim_{t\to\pi} f(t) = -1, \lim_{t\to 0} g(t) = \lim_{t\to\pi} g(t) = 0 \tag{1}$$

$$\tau(0) = \lim_{t\to 0}(f(t), g(t)), \tau(\pi/2) = \lim_{t\to\pi}(f(t), g(t)) \tag{2}$$

We think of the timing rule τ as specifying the position $\tau(t) \in \mathcal{S}$ of the Sender at time t, which is used only during the traversal of curve $\mathcal{S}$. Hence, the pair $(\mathcal{S}, \tau)$ determines uniquely the speed $\|\tau'(t)\|$ of the Sender while traversing curve $\mathcal{S}$, that takes a total time of $\pi/2$. The pair $(\mathcal{S}, \tau)$ will be called *speed compliant*, if the movement along $\mathcal{S}$ with the timing rule τ induces speed which is at most 1.

The Sender's trajectory is parameterized by a speed compliant SPT pair $(\mathcal{S}, \tau)$. Both $\mathcal{S}, \tau$ will be determined later as a function of some $\alpha\colon [0,\pi] \mapsto \mathbb{R}$. It is worthwhile mentioning that, strictly speaking, curve $\mathcal{S}$ is induced by timing rule τ, so one would only need to specify τ. Nevertheless, it is a technicality of our argument that requires first to determine $\mathcal{S}$ (as a function of $\alpha(\cdot)$), and then the timing rule $\tau(\cdot)$ (implicitly, as a function of $\alpha(\cdot)$).

The formal description of the Sender's trajectory can be found in Algorithm 1, and is also depicted in Figure 1 as the red trajectory (for a specific choice of a SPT pair $(\mathcal{S}, \tau)$). Note that the SPT pair $(\mathcal{S}, \tau)$ guarantees that point $\textsc{cycle}(\pi) = (-1, 0)$ is visited twice by the Sender, and what $(\mathcal{S}, \tau)$ determines is the nature of the curved trajectory after the Sender visits point $(1-\pi/2, 0)$, denoted as A_0 in Figure 1.

Note that since $(\mathcal{S}, \tau)$ is SPT, the above trajectory is indeed feasible (it defines a continuous curve on the plane; see transition between phases S3, S4). Moreover, if $(\mathcal{S}, \tau)$ is speed compliant, then the Sender's trajectory has speed at most 1 during the execution of

the move. The Sender, during phase S1, will move from the center of the circle to CYCLE(1), which takes 1 time unit. During phase S2 the sender moves counter-clockwise on the circle from CYCLE(1) towards the point $(-1,0)$, and this phase takes time $\pi-1$. In phase S3 the sender moves from point $(-1,0)$ to the point $(1-\pi/2,0)$ which is the point A_0 as shown in Figure 1, this phase takes time $2-\pi/2$. Finally, for phase S4 the sender moves in a trajectory from point $(1-\pi/2,0)$ to reach point $(-1,0)$ for the second time. Phase S4 takes time $\pi/2$, which gives us the total time of $2+\pi$.

2.1.2 Receiver's Trajectory

A continuous and differentiable $\alpha\colon [0,\pi]\mapsto[-1,1]$ will be called *reserved* if $\alpha_0 = 1-\pi/2$, and the limit $\lim_{t\to\pi}\frac{\alpha_t-\cos t}{\sin t}$ exists and is not equal to ±1 (the latter technical condition will be used in the Proof of Claim 6 in order to show that $\alpha(\cdot)$ induces SPT pair $(\mathcal{S},\tau)$ for the Sender). Note also that the latter condition also implies that $\lim_{t\to\pi}\alpha_t=-1$. The trajectory of the Receiver is parameterized by some reserved $\alpha(\cdot)$. Intuitively, α_t will determine its movement if the Exit is found at point CYCLE$(-t)$, when $t\in[0,\pi]$.

The formal description of the Receiver's trajectory can be found in Algorithm 1, and is also depicted in Figure 1 as the blue trajectory (for a specific choice of a reserved $\alpha(\cdot)$). According to the description of the trajectory, there are two cases in which the Receiver attempts to visit the Sender, see phases R3 and R4. Correctness of phase R3 will be shown later in Claim 8, and is independent of function $\alpha(\cdot)$. The correctness of phase R4 will be shown later in Claim 9, and will follow once we fix $\alpha(\cdot)$ that will also determine the Sender's trajectory (hence pair $(\mathcal{S},\tau)$). Also this step will be correct independently of whether $(\mathcal{S},\tau)$ is speed complaint.

Note that the total duration of the trajectory of the Receiver, under the assumption that no Exit is found, is $2+\pi$, when she meets with the Sender at point $(-1,0)$, and they arrive at that point simultaneously.

2.2 Determining a Partial Trajectory and a Timing-Rule for the Sender

In this section we show how a choice of a function $\alpha(\cdot)$, as the one used to determine the Receiver's trajectory, can also determine the Sender's trajectory and in particular phase S4 of her trajectory. More specifically, given function $\alpha(\cdot)$ we determine a pair $(\mathcal{S},\tau)$ that we also call α-induced.

Intuitively, for every $0<t\le\pi$ we require that the Sender is at point M_t at the same time the Receiver would arrive there if the Exit was found at CYCLE$(-t)$. First we determine points M_t. As per the description of the Receiver's trajectory, if the Exit is found at some CYCLE$(-t)$, for $t\in(0,\pi]$, then the Receiver, starting from CYCLE$(-t)$, moves with unit speed for time[1] $w_t=(\pi-t)/2$ towards point A_t. Therefore point $M_t=(f(t),g(t))$ can be found as the intersection of the line passing through points CYCLE$(-t)=(\cos t,-\sin t)$ and $A_t=(\alpha_t,0)$, and the circle of center CYCLE$(-t)$ and radius w_t, lying within the unit circle. Point M_t can be therefore determined as one of the roots of the following quadratic system with $x=f(t)$ and $y=g(t)$.

[1] This is the maximum time that the receiver can move away from the exit in order to be able to return back to it for a total time of $2+\pi$. We just need to ensure that the Sender reaches M_t at the same time the Receiver does.

$$(y+\sin t)(\alpha_t-\cos t)=(x-\cos t)\sin t$$
$$(x-\cos t)^2+(y+\sin t)^2=w_t^2$$

Solving the system gives one of the roots to be

$$f(t)=\cos t+\frac{(\alpha_t-\cos t)\frac{\pi-t}{2}}{\sqrt{\sin^2 t+(\alpha_t-\cos t)^2}} \tag{3}$$

$$g(t)=-\sin t+\frac{\sin t\frac{\pi-t}{2}}{\sqrt{\sin^2 t+(\alpha_t-\cos t)^2}} \tag{4}$$

The above pair is the point on the line segment with endpoints $\textsc{cycle}(-t), A_t$, and from $\textsc{cycle}(-t)$ to the direction of A_t (the other root is its antipodal on the circle with center $\textsc{cycle}(-t)$ and radius w_t).

We now define the *α-induced* pair $(\mathcal{S},\tau)$. We set $\mathcal{S}:=\{M_t : t\in[0,\pi]\}$. Also, assuming that the Sender reaches point $(1-\pi/2,0)$ (i.e. no Exit is found earlier by the Sender), reset the clock for the Sender and require that for every $t\in[0,\pi]$, the Sender lies at M_t exactly at time $t/2$. Note that this defines indeed a timing-rule $\tau:[0,\pi/2]\mapsto\mathcal{S}$. Later we show that pair $(\mathcal{S},\tau)$ is indeed SPT (i.e., Conditions 1 and 2 are satisfied) for a proper choice of $\alpha(\cdot)$.

2.3 Correctness and Performance Analysis

In this section we show that starting with a reserved $\alpha(\cdot)$, the α-induced pair $(\mathcal{S},\tau)$ is indeed SPT, and that this guarantees that robots meet as per the description of their trajectories, as well as that the running time is at most $2+\pi$. The results are summarized in Lemma 5.

We need to emphasize that results of this section do not touch on the Sender's speed induced by the choice of $\alpha(\cdot)$. Finding a proper $\alpha(\cdot)$ that induces a speed-compliant pair $(\mathcal{S},\tau)$ (i.e., a pair that keeps the Sender's speed to at most 1) is the topic of the next section.

▶ **Lemma 5.** *If $\alpha(\cdot)$ is reserved, then Algorithm 1 solves the sender-receiver problem with evacuation time at most $2+\pi$. In particular, the Receiver's move is of unit speed (while it is moving), while the Sender's move has speed (possibly more than 1) that depends on the choice of $\alpha(\cdot)$.*

The proof of Lemma 5 is given by the following claims. First we show that the α-induced pair $(\mathcal{S},\tau)$ of Section 2.2 is SPT (and therefore α determines indeed continuous trajectories).

▷ Claim 6. If $\alpha(\cdot)$ is reserved, then the α-induced pair $(\mathcal{S},\tau)$ is SPT.

Proof. Since α is reserved we have that $\alpha(0)=1-\pi/2$ and $\alpha(\pi)=-1$. We use the formulas for $M_t=(f(t),g(t))$ of Section 2.2 as a function of $\alpha(\cdot)$. A direct substitution gives

$$f(0)=1+\frac{(\alpha_0-1)\frac{\pi}{2}}{\sqrt{0+(\alpha_0-1)^2}}=1-\pi/2,$$

since $\alpha_0-1<0$. The value $f(\pi)$ is indefinite, so we compute

$$\begin{aligned}\lim_{t\to\pi} f(t) &= -1+\left(\lim_{t\to\pi}\frac{\alpha_t-\cos t}{\sqrt{\sin^2 t+(\alpha_t-\cos t)^2}}\right)\left(\lim_{t\to\pi}\frac{\pi-t}{2}\right)\\ &= -1+\left(\lim_{t\to\pi}\frac{\alpha_t+1}{\sqrt{(\alpha_t+1)^2}}\right)\left(\lim_{t\to\pi}\frac{\pi-t}{2}\right)=-1.\end{aligned}$$

Again, a direct substitution gives $g(0) = 0$. The next step is the only one that uses that $\lim_{t\to\pi} \frac{\alpha_t - \cos t}{\sin t}$ exists and is not equal to ± 1 (since $\alpha(\cdot)$ is reserved). We have

$$\lim_{t\to\pi} g(t) = \lim_{t\to\pi} \frac{\sin t \frac{\pi - t}{2}}{\sqrt{\sin^2 t + (\alpha_t - \cos t)^2}} = \left(\lim_{t\to\pi} \frac{\sin t}{\sqrt{\sin^2 t + (\alpha_t - \cos t)^2}}\right)\left(\lim_{t\to\pi} \frac{\pi - t}{2}\right)$$
$$= \left(\lim_{t\to\pi} \frac{1}{\sqrt{1 + \left(\frac{\alpha_t - \cos t}{\sin t}\right)^2}}\right)\left(\lim_{t\to\pi} \frac{\pi - t}{2}\right) = 0,$$

where the second to last limit exists and is some constant, due to the fact that $\alpha(\cdot)$ is reserved.

Next we study the timing-rule $\tau(\cdot)$, and we verify that $\tau(0) = (f(0), g(0)), \tau(\pi/2) = (f(\pi), g(\pi))$. Indeed, reset the clock to 0 at the time the Sender reaches point $(1 - \pi/2, 0)$. Recall that for every $t \in [0, \pi]$, τ was defined so that at time $t/2$ the Sender is at point M_t. For $t \to 0$, our previous calculations for f, g show indeed that $\lim_{t\to 0}(f(t), g(t)) = (1 - \pi/2, 0)$, which is exactly where the Sender is at time 0. Finally, for $t \to \pi$, we know that $\lim_{t\to\pi}(f(t), g(t)) = (-1, 0)$, while M_π was indeed defined as point $(-1, 0)$. This is because M_π is the attempted meeting point if the Exit is found at point $\textsc{cycle}\,(\pi)$, in which case, by definition, M_π is $w_\pi = (\pi - \pi)/2 = 0$ away from $\textsc{cycle}\,(\pi)$. ◁

Next note that, given the described trajectories (and independently of $\alpha(\cdot)$), the Sender searches all points $\textsc{cycle}\,(t)$ with $1 \le t \le \pi$, while the Receiver searches all points $\textsc{cycle}\,(t)$ with $-\pi \le t \le 1$. Now assume that the Exit is indeed in some $\textsc{cycle}\,(t)$, where $-\pi \le t \le \pi$ (i.e., anywhere on the unit circle). Claim 7 below covers the case that $1 \le t \le \pi$, hence the Exit is found by the Sender. Claim 8 below covers the case $0 \le t \le 1$, hence the Exit is found by the Receiver. Lastly, Claim 9 below covers the case $-\pi \le t \le 0$, hence the Exit is found by the Receiver, not before she has spent time 1 searching on the circle. All these claims assume that the provided $\alpha(\cdot)$, determining robots' trajectories, is reserved.

In the first case, the evacuation cost is strictly less than $2 + \pi$.

▷ Claim 7. If the Exit is found by the Sender in $\textsc{cycle}\,(t)$, with $1 \le t \le \pi$, then the evacuation time is less than $2 + \pi$.

Proof. Consider the Exit at point $\textsc{cycle}\,(1 + x)$ for some $x \in [0, \pi - 1]$. The Exit is discovered by the Sender at time $1 + x$, after spending time x moving counter-clockwise, starting from $\textsc{cycle}\,(1)$. Until the Exit is found, the Receiver moves in the opposite direction, starting again from $\textsc{cycle}\,(1)$. Therefore, when the Receiver's receives the message from the Sender, the two are at arc-distance $2x$, or equivalently at Euclidean distance $2\sin x$. It follows that the worst case evacuation in this case equals

$$\max_{0\le x\le \pi-1} \{1 + x + 2\sin x\} \le \pi + 2\sin 1 < \pi + 2.$$ ◁

In the second case, the evacuation cost is at most $2 + \pi$, and equality holds for selected placements of the Exit.

▷ Claim 8 (Correctness of phase R3 of Receiver's Trajectory). If the Exit is found by the Receiver in $\textsc{cycle}\,(1 - t)$, with $0 \le t \le 1$, then the evacuation time is at most $2 + \pi$.

Proof. As per the Receiver's trajectory, if the Exit is found in $T := \{\textsc{cycle}\,(1 - t) : t \in [0, 1]\}$, the Receiver attempts to meet the Sender in point $P = (1 - \pi/2, 0)$. Note that for the duration that the Receiver is exploring T the corresponding part of the circle, the Receiver's distance to P is increasing. Hence, the latest time that the Receiver reaches point P, over

all placements of the exits that induce this move, is when the Exit is placed at $\textsc{cycle}(0)$. It is also important to note that the move is the same limiting move that the Receiver makes in order to meet the Sender if the Exit is at $\textsc{cycle}(t)$, and t tends to 0, either from the left or the right. In this case it is easy to see that the Receiver reaches point P at time $2+\pi/2$. Hence, for any placement of the Exit in T, the Receiver reaches P at time at most $2+\pi/2$, and waits for the Sender who arrives at P at time exactly $2+\pi/2$ (1 to reach the perimeter, $\pi-1$ to explore her part of the circle, and $2-\pi/2$ to reach P from $\textsc{cycle}(\pi)$). Noting that P is $\pi/2$ away from $\textsc{cycle}(0)$, and in fact at most $\pi/2$ away from any point in T, concludes the proof. ◁

In the third case, we show that the evacuation cost stays invariant (for infinite and uncountable many placements of the Exit).

▷ Claim 9 (Correctness of phase R4 of Receiver's Trajectory). If the Exit is found by the Receiver in $\textsc{cycle}(-t)$, with $0 \le t \le \pi$, then the evacuation time equals $2+\pi$.

Proof. Consider the Exit at point $\textsc{cycle}(-t)$. The Receiver arrives at the point in time $2+t$ (1 to reach the perimeter, extra time 1 to reach $\textsc{cycle}(0)$, and another t to reach $\textsc{cycle}(-t)$). Then, the Receiver spends time w_t toward the (intended) meeting point M_t, which is reached at time $2+t+w_t = 2+t+(\pi-t)/2 = 2+\pi/2+t/2$.

Now we show that the Sender arrives at M_t at the exact same time (and note that the Sender follows her trajectory without knowing the findings of the Receiver). Observe first that the Sender reaches point $M_0 = (1-\pi/2, 0)$ in time $2+\pi/2$ (1 to reach the perimeter, extra time $\pi-1$ to reach $\textsc{cycle}(\pi)$, and extra time $2-\pi/2$ to reach M_0). Since the Sender's trajectory uses the α-induced pair $(\mathcal{S}, \tau)$ thereafter (see definition in Section 2.2), the Sender reaches M_t in additional time $t/2$ for a total of $2+\pi/2+t/2$, which is simultaneously with the Receiver.

Since the two robots do indeed meet at point M_t in time $2+\pi/2+t/2$, they return together to $\textsc{cycle}(-t)$ in extra time w_t, for a total of $2+\pi/2+t/2+w_t = 2+\pi$. ◁

This concludes the proof of Lemma 5.

2.4 Speed Compliant Partial Trajectory (Proof of Theorem 4)

In this section we provide the missing component toward the proof of Theorem 4, a reserved function $\alpha(\cdot)$. To this end we define $\alpha\colon [0,\pi] \mapsto [-1,1]$ as

$$a(t) := 1 - \pi/2 + (-2+\pi/2)\sin(t/2)$$

which also gives rise to the α-induced pair $(\mathcal{S}, \tau)$ as per Section 2.2.

▶ **Lemma 10.** *Function $\alpha(\cdot)$ is reserved.*

Proof. Direct substitution shows that $\alpha(0) = 1-\pi/2, \alpha(\pi) = -1$. Moreover, using l'Hopital's rule

$$\begin{aligned}\lim_{t\to\pi} \frac{\alpha_t - \cos t}{\sin t} &= \lim_{t\to\pi} \frac{1-\pi/2+(-2+\pi/2)\sin(t/2)-\cos t}{\sin t} \\ &= \lim_{t\to\pi} \frac{\frac{1}{2}(-2+\pi/2)\cos(t/2)+\sin t}{\cos t} = 0,\end{aligned}$$

which concludes that $\alpha(\cdot)$ is indeed reserved. ◀

Since by Lemma 10 the function $\alpha(\cdot)$ is reserved, Lemma 5 applies, according to which the robots' trajectories of Algorithm 1 solve the sender-receiver problem with evacuation time at most $2+\pi$. The Receiver maintains speed 1 (except from when she is idle), while the Sender's speed is induced by $\alpha(\cdot)$. Therefore, Theorem 4 follows once we show that the α-induced pair $(\mathcal{S},\tau)$ is speed compliant. This is done in Lemma 11 below.

We emphasize that our proof is computer-assisted in the following sense (but still rigorous). First we rely on numerical evaluations of formulae that admit closed (symbolic) forms. Specifically for our goal to show that a certain function (the one describing the Sender's speed) takes values at most 1 over a domain, we discretize the domain and we identify a "safe" subdomain where the function is bounded away from 1. Given that we show that the function is smooth enough, the provided discretization shows that the calculations are robust against (possible) numerical inaccuracies and hence the function is at most 1 in the safe subdomain. In the complementary "critical" subdomain where the function admits values close to 1, we study the monotonicity of our function. To that end, we identify points in the critical subdomain in which the function provably evaluates to 1, and then we show that these points correspond to local strict maximizers (where the locality of the argument covers the entire "critical" subdomain).

▶ **Lemma 11.** *The α-induced pair $(\mathcal{S},\tau)$ is speed compliant.*

Proof. We show that the α-induced pair $(\mathcal{S},\tau)$ gives rise to a movement of speed at most 1. Indeed, as per the definition of the trajectory, and for $t\in[0,\pi]$, the Sender is at point $M_t=(f(t),g(t))$ in time $t/2$. Therefore, for $t\in[0,\pi/2]$, the Sender's speed is given by

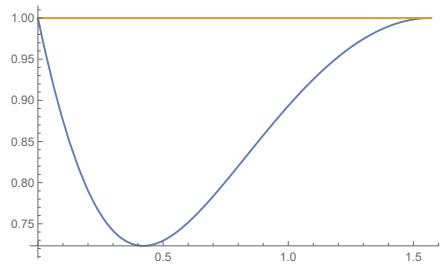
$$speed(t) := \left(\left(\frac{\mathrm{d}}{\mathrm{d}t}f(2t)\right)^2+\left(\frac{\mathrm{d}}{\mathrm{d}t}g(2t)\right)^2\right)^{1/2},$$

where functions f,g are given in (3), (4) (see Figure 2a for the plot of the function). We need to verify that $speed(t)\le 1$, for all $t\in[0,\pi/2]$. For this we rely on a computer assisted proof that relies on a refined enough discretization of domain $[0,\pi/2]$. By calculating $speed(t)$ exclusively on the discretized domain, and after we identify key monotinicity properties of the function, we show that indeed the speed is at most 1. Towards that direction we define functions $f_0(t)=speed^2(t), f_1(t)=f_0'(t), f_2(t)=f_1'(t)$; see Figures 2b, 2c and 2d for their plots, respectively. We will verify that $f_0(t)\le 1$, for all $t\in[0,\pi/2]$. The main idea behind our argument is that for values of t for which $f_0(t)$ is bounded away from 1, numerical (computer-assisted) evaluations of the function are robust enough to guarantee that the function is indeed at most 1. A special treatment is required for the neighborhood of values in which $f_0(t)$ attains provably value equal to 1.

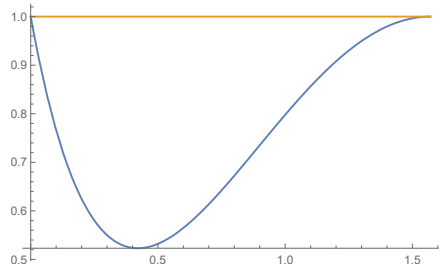
Note that all functions $f_i(t), i=0,1,2$ admit closed formulae, so all arguments below that refer to Figures 2b, 2c, 2d for the sake of intuition, are actually supported by concrete symbolic evaluations of functions $f_i(t)$ in the discretized domain.

From the corresponding evaluation of $f_0(t)$ in the discretized space of $[0,\pi/2]$, we see that $f_0(t)$ is bounded away from 1 as long as t is bounded away from 0 and $\pi/2$. Indeed, by Figure 2b we have that $f_0(t)\le 0.97$ for all $t\in[0.1,1]$. Hence, it remains to show that $f_0(t)\le 1$ also in the neighborhoods close to $t=0$ and $t=\pi/2$, covering in particular the intervals $[0,0.1]$ and $[1,\pi/2]$.

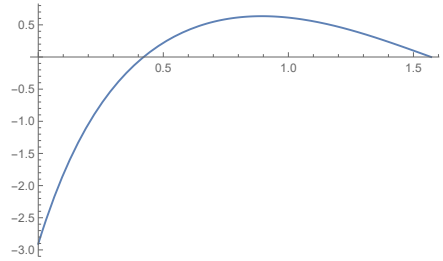
Starting with the neighborhood of $t=0$, and with direct symbolic calculations, we see that $\lim_{t\to 0^+} f_0(t)=1$. At the same time we see that $f_1(t)<-1$ for all $t\le 0.1$, see Figure 2c. It can be shown that the derivative of $f_1(t)$ is bounded and therefore a refined evaluation of $f_1(t)$ that shows it is negative for $t\le 0.1$ is enough to justify that $f_0(t)$ is at most 1. We



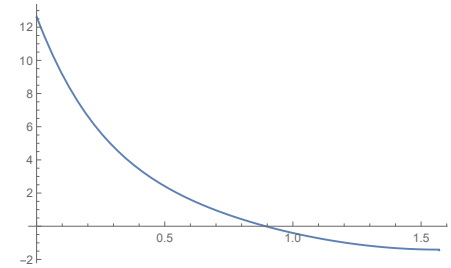
(a) The plot of function $speed(t), t \in [0, \pi/2]$.



(b) The plot of function $f_0(t), t \in [0, \pi/2]$.



(c) The plot of function $f_1(t) = f_0'(t), t \in [0, \pi/2]$.



(d) The plot of function $f_2(t) = f_0''(t), t \in [0, \pi/2]$.

Figure 2 Numerical evaluation of functions $speed(t), f_i(t), i = 0, 1, 2$ for $t \in [0, \pi/2]$.

conclude that even with the considered discretization, function $f_0(t)$ is strictly decreasing for $t \leq 0.1$. That accounts for any possible numerical inaccuracies when computing $f_0(t)$ numerically, and hence $f_0(t) \leq 1$ for all $t \leq 0.1$.

Next we study the neighborhood of $t = \pi/2$. Direct symbolic calculations show that $\lim_{t \to \pi/2} f_0(t) = 1$. However, $\lim_{t \to \pi/2} f_1(t) = 0$, so we need to follow a different argument than before. In that direction, we study $f_2(t)$, see Figure 2d. We observe that when $t \in [1, \pi/2]$, then $f_2(t) < -0.4$. Hence, we have numerical verification that $f_2(t)$ is strictly negative in $[1, \pi/2]$ (bounded away from 0), and hence $f_0(t)$ is strictly concave in the same interval. As a result, $t = \pi/2$ is a strict local maximum of $f_1(t)$, and therefore $f_0(t) \leq 1$ for all $t \in [1, \pi/2]$, concluding our argument. ◀

3 Lower Bound

We prove the following theorem which differentiates the WiFi and the SR models with respect to evacuation time.

▶ **Theorem 12.** *The evacuation time of any algorithm which accomplishes evacuation from an unknown Exit in a unit disk using the SR model is strictly bigger than* $1 + \frac{2\pi}{3} + \sqrt{3}$*, the optimal evacuation time for the WiFi model.*

Proof. The proof is based on the fact that if the evacuation time is exactly $1 + \frac{2\pi}{3} + \sqrt{3}$ then the robots must follow specific trajectories. The robots begin their search starting from the center of the unit disk. They need at least one time unit to reach the perimeter after which they begin their exploration of the perimeter in order to find the Exit. Consider the two robots at time exactly $1 + \frac{2\pi}{3}$. Together the robots can cover a length of at most $\frac{4\pi}{3}$ on the perimeter. We first prove a useful lemma which is similar to Lemma 3 in [2].

▶ **Lemma 13.** *If at time exactly* $1 + \frac{2\pi}{3}$ *for some* $x > 0$ *the total perimeter explored by the two robots is at most* $\frac{4\pi}{3} - x$ *then there is a chord on the circle of length* $> \sqrt{3}$ *none of whose endpoints has been visited by a robot.*

Proof. Assume on the contrary there is no such chord, i.e., for every chord of length $> \sqrt{3}$ at least one of its endpoints has been explored by a robot. Consider a point A on the perimeter which has not been explored by a robot. Consider two points A_1 and A_2 to the left and right of A such that we get two arcs, one starting from A_1 and moving clockwise towards A_2 without passing from A, and one starting from A_1 and moving counter-clockwise passing through A and then reaching A_2 as shown in Figure 3.

We choose points A_1 and A_2 such that every point in the arc $\widehat{A_1A_2}$ has been explored by a robot.

If there is an unexplored point B to the left of A then we can use the same reasoning as above to expand the explored arc below the arc $\widehat{A_1A_2}$. Moreover, the farther the point B from A the longer the arc. Take the point B farthest and to the left of A which is unexplored by a robot. Using the same reasoning take the point C farthest and to the right of A which is unexplored by a robot. Clearly, the arc $\widehat{BA_1A_2C}$ is fully explored by robots with the exception of the endpoints B and C. However, by the hypothesis of the lemma the arc $\widehat{BA_1A_2C}$ has length at most $\frac{4\pi}{3} - x$. It follows that $|BC| > \sqrt{3}$ but this is a contradiction since both endpoints B and C have not been explored by a robot. This proves Lemma 13. ◀

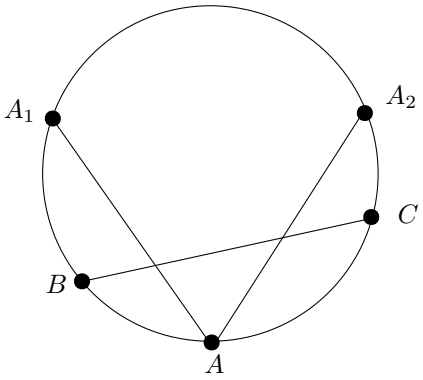


Figure 3 Robot configuration at time $1 + \frac{2\pi}{3}$. A is a point that has not been visted by any of the two robots. The arcs $\widehat{AA_1}, \widehat{AA_2}$ have length $\frac{2\pi}{3}$.

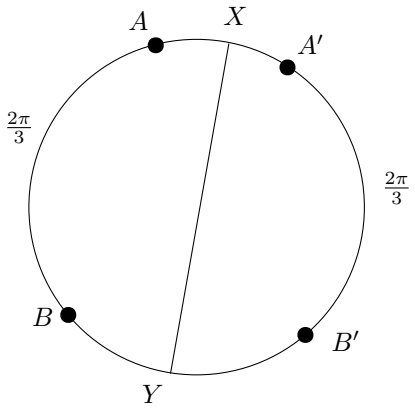


Figure 4 Two non-overlapping contiguous trajectories, forming respective arcs AB and $A'B'$, each of length $\frac{2\pi}{3}$ by the Sender and the Receiver, respectively, at time $1 + \frac{2\pi}{3}$.

Now we proceed to the main proof of Theorem 12 and to this end we consider three cases.

Case 1. The trajectories of the robots either have a nontrivial overlap or one of the trajectories is non-contiguous.
If the trajectories of the robots have an overlap of length ℓ, for some $\ell > 0$ then it is clear that in time $2\pi/3$ the robots cannot cover together more than a length of $\frac{4\pi}{3} - \ell$ of the perimeter. Therefore by Lemma 13 there is a chord of length $> \sqrt{3}$ none of whose endpoints has been explored by a robot. Similarly, if one of the robots' trajectories is not contiguous, i.e., one of the robots moves through the interior of the disk during its trajectory from one point on the perimeter to another, then it is clear that the robots cannot cover together more than a

length of $\frac{4\pi}{3} - \ell$ of the perimeter, where ℓ is the length of the arc a robot avoids by moving across the chord of this arc. Therefore again making use of Lemma 13 we see that there is a chord of length $> \sqrt{3}$ none of whose endpoints has been explored by a robot. In either case, depending on which endpoint of this chord is visited first by a robot the adversary places the Exit at the other endpoint and the additional time required will be bigger than $\sqrt{3}$. This proves the theorem in *Case 1*.

So without loss of generality from now on we may assume that both robot trajectories are contiguous (they do not skip any length on the circle) and have no non-trivial overlap. We consider two additional cases.

Case 2. If the perimeter explored by both robots by time $1 + \frac{2\pi}{3}$ is itself non-continguous. Note that each robot explores a contiguous arc of the perimeter of length $\frac{2\pi}{3}$. Moreover the two trajectories are non-overlapping as depicted in Figure 4. Hence there is a chord of length 2 none of whose endpoints has been explored by a robot. Arguing as before the resulting evacuation time will be at least $1 + \frac{2\pi}{3} + 2$. Hence the theorem is proved in *Case 2* as well.

Therefore it remains to consider the cases where the trajectories of the robots on the perimeter form a single contiguous curve of length $\frac{4\pi}{3}$. This gives rise to the following case.

Case 3. The perimeter explored by both robots by time $1 + \frac{2\pi}{3}$ is itself a contiguous arc of length $\frac{4\pi}{3}$.
There are four Subcases to consider depending on where the robots start and finish on the perimeter. If any of the following three Subcases depicted in Figure 5 occurs the evacuation time will be $1 + \frac{2\pi}{3} + 2$. Indeed, in each Subcase the trajectories are contiguous and overlapping

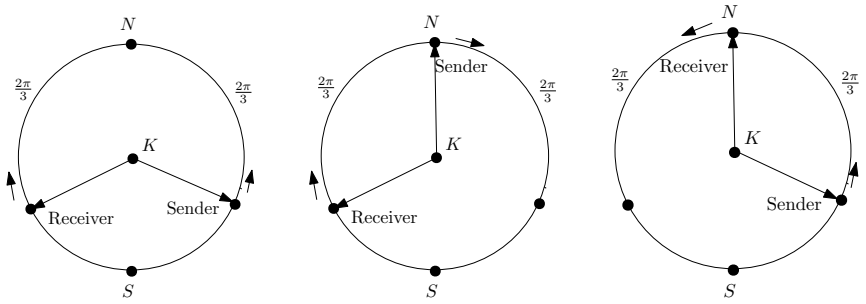


Figure 5 Three cases of possible trajectories of the two robots at time $1 + \frac{2\pi}{3}$ forming a single contiguous segment on the perimeter of total length $\frac{4\pi}{3}$.

at the single point N (at the north pole of the perimeter of the circle). Since all the points in the arc of length $\frac{2\pi}{3}$ at the bottom are unexplored we can find a point S at the bottom (namely the south pole) unexplored by any of the robots. If the adversary places the Exit at the south pole S (a point antipodal to N) either the Sender or the Receiver will have to spend an additional time 2 to evacuate. Arguing as before the resulting evacuation time will be at least $1 + \frac{2\pi}{3} + 2$. Hence the theorem is proved for these three Subcases of *Case 3* as well.

The last remaining Subcase is depicted in Figure 6 in which both robots go together to the north pole N and then traverse the perimeter for a time $\frac{2\pi}{3}$. In this final case we will make use of the fact that the Receiver cannot send messages wirelessly. Without loss of generality let the Receiver follow the left arc NX and the Sender the right arc NY. Consider the two robots at an additional time $\frac{\sqrt{3}}{2}$. On the one hand, if the Sender robot does not reach the midpoint M of the chord XY then the adversary can place the Exit at a suitable point close to X in the arc XSY so that evacuation takes additional time which is more

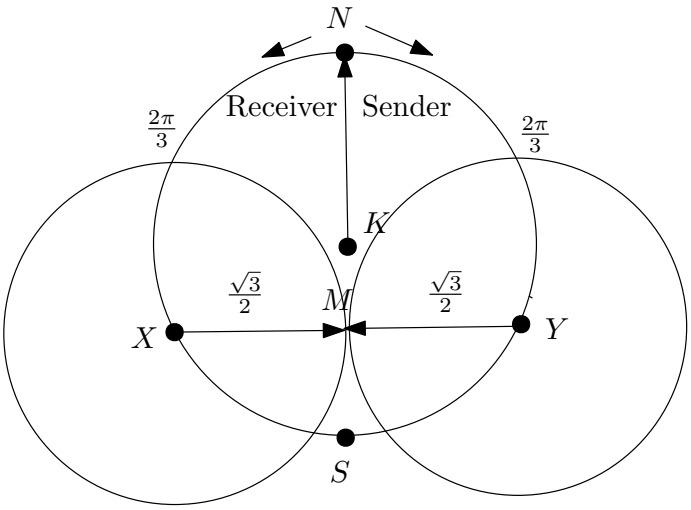


Figure 6 Robot configuration at time $1+\frac{2\pi}{3}+\frac{\sqrt{3}}{2}$. The robots meet at the midpoint M of the chord XY.

than $\frac{\sqrt{3}}{2}$ thus proving that the evacuation time exceeds $1+\frac{2\pi}{3}+\sqrt{3}$. On the other hand, if the Sender reaches the point M it cannot have explored any point in the arc XSY other than the point Y. Therefore a similar reasoning applies if at the same time the Receiver does not reach the midpoint M of the chord XY: the adversary can place the Exit at a suitable point close to Y in the arc XSY so that evacuation takes additional time which is more than $\frac{\sqrt{3}}{2}$ thus proving again that the evacuation time exceeds $1+\frac{2\pi}{3}+\sqrt{3}$.

So without loss of generality we may assume that in time exactly $1+\frac{2\pi}{3}+\frac{\sqrt{3}}{2}$ both robots meet at the point M without having explored any other part of the perimeter other than the arc XNY. Moreover, it is clear that none of the robots has explored any point on the arc XSY. Consider a point X' close to X on the arc XSY which is at distance ϵ from X for some $\epsilon>0$ sufficiently small. The adversary places the Exit at one of the endpoints of the chord $X'S$ and the evacuation will require additional time $|MS|+|X'S|-\epsilon$ which is clearly at least $\frac{\sqrt{3}}{2}+\frac{1}{2}-\epsilon$. To see this, depending on which of the endpoints of the chord $X'S$ is visited first by a robot the adversary places the Exit at the other endpoint. Therefore in this case the evacuation time exceeds $1+\frac{2\pi}{3}+\sqrt{3}+\frac{1}{2}-\epsilon$. This gives the proof in this case as well and the proof of Theorem 12 is now complete. ◀

4 Conclusion

In this paper we considered the evacuation problem for two robots in the SR model from an unknown Exit placed on the perimeter of the unit disk. We proved upper and lower bounds for evacuation and showed that the evacuation time of our algorithm is at most $\pi+2$. Closing the gap between the lower and upper bounds remains a challenging open question. The mixed communication model considered is interesting for multiple robots. For example, suppose we have n robots all of which can communicate F2F while s among the n are senders (can only send wirelessly) and the remaining r are receivers (can only receive wirelessly) such that $n=s+r$. Nothing is known about the optimal evacuation time in this more general model. (Note that if a sender and a receiver join and move together as one robot then they behave as a fully wireless robot.) In our study, the communication status of the robots (i.e., either sender or receiver) is known in advance. An interesting extension worth considering is when the communication status of the robots may be set by an adversary.

References

1 Sebastian Brandt, Felix Laufenberg, Yuezhou Lv, David Stolz, and Roger Wattenhofer. Collaboration without communication: Evacuating two robots from a disk. In *International Conference on Algorithms and Complexity*, pages 104–115. Springer, 2017.

2 J. Czyzowicz, L. Gasieniec, T. Gorry, E. Kranakis, R. Martin, and D. Pajak. Evacuating robots from an unknown exit located on the perimeter of a disc. In *DISC 2014*, pages 122–136. Springer, Austin, Texas, 2014.

3 Jurek Czyzowicz, Konstantinos Georgiou, Maxime Godon, Evangelos Kranakis, Danny Krizanc, Wojciech Rytter, and Michał Włodarczyk. Evacuation from a disc in the presence of a faulty robot. In *International Colloquium on Structural Information and Communication Complexity*, pages 158–173. Springer, 2017.

4 Jurek Czyzowicz, Konstantinos Georgiou, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Manuel Lafond, Lata Narayanan, Jaroslav Opatrny, and Sunil M. Shende. Energy consumption of group search on a line. In *46th International Colloquium on Automata, Languages, and Programming, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 137:1–137:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

5 Jurek Czyzowicz, Konstantinos Georgiou, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, and Sunil Shende. Search on a line by byzantine robots. *International Journal of Foundations of Computer Science*, pages 1–19, 2021.

6 Jurek Czyzowicz, Konstantinos Georgiou, Evangelos Kranakis, Lata Narayanan, Jaroslav Opatrny, and Birgit Vogtenhuber. Evacuating robots from a disk using face-to-face communication. *Discret. Math. Theor. Comput. Sci*, 22(4), 2020.

7 Jurek Czyzowicz, Kostantinos Georgiou, and Evangelos Kranakis. Group search and evacuation. In *Distributed Computing by Mobile Entities*, pages 335–370. Springer, 2019.

8 Jurek Czyzowicz, Maxime Godon, Evangelos Kranakis, and Arnaud Labourel. Group search of the plane with faulty robots. *Theoretical Computer Science*, 792:69–84, 2019.

9 Jurek Czyzowicz, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Denis Pankratov, and Sunil Shende. Group evacuation on a line by agents with different communication abilities. *ISAAC 2021*, pages 57:1–57:24, 2021.

10 Jurek Czyzowicz, Ryan Killick, Evangelos Kranakis, and Grzegorz Stachowiak. Search and evacuation with a near majority of faulty agents. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 217–227. SIAM, 2021.

11 Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, and Jaroslav Opatrny. Search on a line with faulty robots. *Distributed Computing*, 32(6):493–504, 2019.

12 Yann Disser and Sören Schmitt. Evacuating two robots from a disk: a second cut. In *International Colloquium on Structural Information and Communication Complexity*, pages 200–214. Springer, 2019.

13 Konstantinos Georgiou, Evangelos Kranakis, Nikos Leonardos, Aris Pagourtzis, and Ioannis Papaioannou. Optimal circle search despite the presence of faulty robots. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 192–205. Springer, 2019.

14 Ioannis Lamprou, Russell Martin, and Sven Schewe. Fast two-robot disk evacuation with wireless communication. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing*, pages 1–15, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

15 Nikos Leonardos, Aris Pagourtzis, and Ioannis Papaioannou. Byzantine fault tolerant symmetric-persistent circle evacuation. In Leszek Gąsieniec, Ralf Klasing, and Tomasz Radzik, editors, *Algorithms for Sensor Systems*, pages 111–123, Cham, 2021. Springer International Publishing.

16 X. Sun, Y. Sun, and J. Zhang. Better upper bounds for searching on a line with byzantine robots. In *Complexity and Approximation*, pages 151–171. Springer, 2020.

Extended MSO Model Checking via Small Vertex Integrity

Tatsuya Gima
Nagoya University, Japan

Yota Otachi
Nagoya University, Japan

Abstract

We study the model checking problem of an extended MSO with local and global cardinality constraints, called $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$, introduced recently by Knop, Koutecký, Masařík, and Toufar [*Log. Methods Comput. Sci.*, 15(4), 2019]. We show that the problem is fixed-parameter tractable parameterized by vertex integrity, where vertex integrity is a graph parameter standing between vertex cover number and treedepth. Our result thus narrows the gap between the fixed-parameter tractability parameterized by vertex cover number and the W[1]-hardness parameterized by treedepth.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases vertex integrity, monadic second-order logic, cardinality constraint, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.20

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2207.01024

Funding *Yota Otachi*: JSPS KAKENHI Grant Numbers JP18H04091, JP18K11168, JP18K11169, JP20H05793, JP21K11752, JP22H00513.

Acknowledgements The authors thank Michael Lampis and Valia Mitsou for fruitful discussions and sharing a preliminary version of [37].

1 Introduction

One of the most successful goals in algorithm theory is to have a meta-theorem that constructs an efficient algorithm from a *description* of a target problem in a certain format (see e.g., [29, 30, 35]). Courcelle's theorem [6, 8, 9, 11] is arguably the most successful example of such an algorithmic meta-theorem, which says (with Bodlaender's algorithm [4]) that: if a problem on graphs can be expressed in monadic second-order logic (MSO), then the problem can be solved in linear time on graphs of bounded treewidth. Many natural problems that are NP-hard on general graphs are shown to have expressions in MSO and thus have linear-time algorithms on graphs of bounded treewidth [1].

Although the expressive power of MSO captures many problems, it is known that MSO cannot represent some kinds of cardinality constraints [10]. For example, it is easy to express the problem of finding a proper vertex coloring with r colors in MSO as the existence of a partition of the vertex set into r independent sets, where the length of the corresponding MSO formula depends on r. However, the variant of the problem that additionally requires the r independent sets to be of the same size cannot be expressed in MSO even if $r = 2$ (see [10]). Indeed, this problem is known to be W[1]-hard parameterized by r and treewidth [16].[1] See [3, 28, 41] for many other examples of such problems.

[1] We assume that the readers are familiar with the concept of parameterized complexity. For standard definitions, see e.g., [12].

For those problems that do not admit MSO expressions and are hard on graphs of bounded treewidth, there is a successful line of studies on smaller graph classes with more restricted structures. For example, by techniques tailored for individual problems, several problems are shown to be tractable on graphs of bounded vertex cover number (see e.g., [15, 17, 18]). Such results are known also for more general parameters such as twin-cover [24], neighborhood diversity [36], and vertex integrity [28]. Then the natural challenge would be finding a meta-theorem covering (at least some of) such results. Recently, such meta-theorems are intensively studied for extended MSO logics with "cardinality constraints." In this paper, we follow this line of research and focus on vertex integrity as the structural parameter of input graphs. The *vertex integrity* of a graph is the smallest number $k = s + t$ such that by removing s vertices of the graph, every component can be made to have at most t vertices. The concept of vertex integrity was introduced first in the context of network vulnerability [2]. It basically measures how difficult it is to break a graph into small components by removing a small number of vertices. This can be seen as a generalization of vertex cover number, which asks to remove vertices to make the graph edge-less (corresponding to the case $t = 1$ of the definition of vertex integrity). On the other hand, the concept of treedepth can be seen as a recursive generalization of vertex integrity. This is because we can define the treedepth of a graph recursively as follows: the treedepth of a graph is 1 if it has no edges, and otherwise it is the minimum number $k = s + t$ such that s vertices can be removed from the graph to make the treedepth of every component of the remaining graph at most t. Actually, their definitions give us the inequality treedepth $\leq$ vertex integrity $\leq$ vertex cover number $- 1$ for every graph (see [28]).

There is another issue about Courcelle's theorem that the dependency of the running time on the parameters (the treewidth of the input graph and the length of formula) is quite high [20]. To cope with this issue, faster algorithms are proposed for special cases such as vertex cover number, neighborhood diversity, and max-leaf number [36], twin-cover [24], shrubdepth [25], treedepth [23], and vertex integrity [37]. The methods in these results are similar in the sense that they find a smaller part of the input graph that is equivalent to the original graph under the given MSO formula. Interestingly, these techniques are used also in studies of extended MSO logics in these special cases. Our study is no exception, and we use a result in [37] as a key lemma.

Meta-theorems on extended MSO with cardinality constraints

In this direction, there are two different lines of research, which have been merged recently. One line considers "global" cardinality constraints and the other considers "local" cardinality constraints.

Recall that the property of having a partition into r independent sets of equal size cannot be expressed in MSO. A remedy for this would be to allow a predicate like $|X| = |Y|$. The concept of global cardinality constraints basically implements this but in a more general way (see Section 2 for formal definitions). It is known that the model checking for the extended MSO logic with global cardinality constraints is fixed-parameter tractable parameterized by neighborhood diversity [27].

The concept of local cardinality constraints was originally introduced as the *fairness* of a solution [39]. The fairness of a solution (a vertex set or an edge set) upper-bounds the number of neighbors each vertex can have in the solution. It is known that finding a vertex cover with an upper bound on the fairness is W[1]-hard parameterized by treedepth and feedback vertex set number [33]. On the other hand, the problem of finding a vertex set satisfying an MSO formula and fairness constraints is fixed-parameter tractable parameterized by neighborhood

diversity [40] and by twin-cover [33]. The general concept of local cardinality constraint extends the concept of fairness by having for each vertex, an individual set of the allowed numbers of neighbors in the solution. It is known that the extension of MSO with local cardinality constraints admits an XP algorithm (i.e., a slicewise-polynomial time algorithm) parameterized by treewidth [42].

Knop, Koutecký, Masařík, and Toufar [34] recently converged two lines and studied the model checking of extended MSO with both local and global cardinality constraints. It is shown that the problem admits an XP algorithm parameterized by treewidth. Furthermore, they showed that the problem is fixed-parameter tractable parameterized by neighborhood diversity if the cardinality constraints are "linear," where each local cardinality constraint is a set of consecutive integers and each global cardinality constraint is a linear inequality.

Our results

We study the linear version of the problem in [34] mentioned above; that is, the model checking of the extended MSO logic with linear local and global cardinality constraints. We show that this problem, called $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking, is fixed-parameter tractable parameterized by vertex integrity. This result fills a missing part in the map on the complexity of $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking as vertex integrity fits between vertex cover number and treedepth: for the former the problem is fixed-parameter tractable [34], and for the latter it is W[1]-hard [33]. Note that by MSO, we mean MSO_1, which does not allow edge and edge-set variables. After proving the main result, we show that the same result holds even for the same extension of MSO_2. We apply the results to several problems and show some new examples that are fixed-parameter tractable parameterized by vertex integrity. We also show that some known results can be obtained as applications of our results.[2]

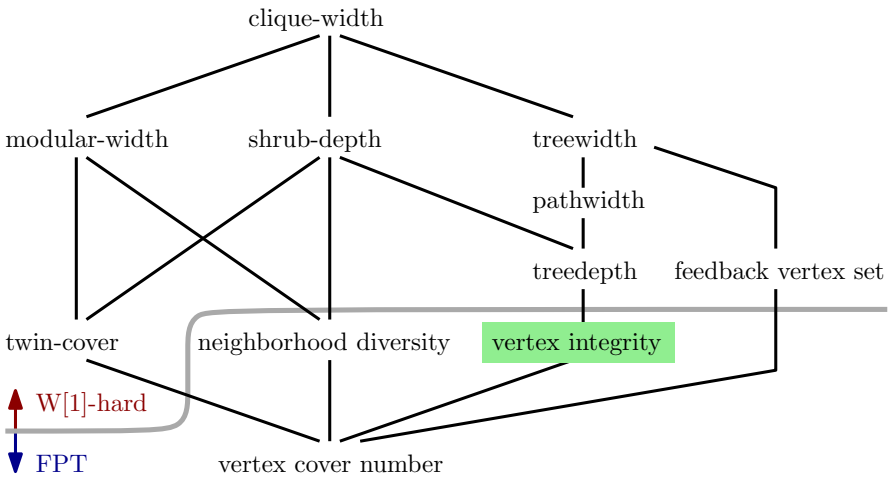


Figure 1 Some of the major graph parameters and the complexity of $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking. If one parameter is an ancestor of another, then the ancestor is upper-bounded by a function of the descendant. The fixed-parameter tractability parameterized by neighborhood diversity is shown in [34]. The W[1]-hardness parameterized by twin-cover and by treedepth and feedback vertex set number are shown in [33].

[2] Omitted from the conference version. See the full version.

2 Preliminaries

For two integers a and b, we define $[a,b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$. We write $[b]$ for the set $[1,b]$. For two tuples $\boldsymbol{A} = (A_1, \ldots, A_p)$ and $\boldsymbol{B} = (B_1, \ldots, B_q)$ of vertex sets, the concatenation $(A_1, \ldots, A_p, B_1, \ldots, B_q)$ is denoted by $\boldsymbol{A} \dot{+} \boldsymbol{B}$. For a function $f\colon X \to Y$ and a set $A \subseteq X$, the restriction of f to A is denoted by $f|_A$.

2.1 Graphs and colored graphs

We consider undirected graphs without self-loops or multiple edges. Let $G = (V, E)$ be a graph. The vertex set and the edge set of G are denoted by $V(G)$ and $E(G)$, respectively. A *component* of G is a maximal connected induced subgraph of G. For a vertex set S of a graph G, the subgraph of G induced by $V \setminus S$ is denoted by $G - S$.

A *p-color list* $\boldsymbol{C}$ of G is a tuple $\boldsymbol{C} = (C_1, \ldots, C_p)$ of p vertex sets $C_i \subseteq V$. Denote the set of colors assigned by $\boldsymbol{C}$ to $v \in V$ by $\mathrm{col}_{\boldsymbol{C}}(v)$. Note that each vertex can have multiple colors. That is, $\mathrm{col}_{\boldsymbol{C}}(v) = \{i \in [p] \mid v \in C_i\}$. Note that $\mathrm{col}_{\boldsymbol{C}}(v)$ can be computed in time polynomial in $|V|$ and p. We call a tuple $(G, \boldsymbol{C})$ a *p-colored graph*. If the context is clear, we simply call it a graph.

Let $\mathcal{G}_1 = (G_1, \boldsymbol{C}_1)$ and $\mathcal{G}_2 = (G_2, \boldsymbol{C}_2)$ be p-colored graphs. A bijection $\psi\colon V(G_1) \to V(G_2)$ is an *isomorphism* from $\mathcal{G}_1$ to $\mathcal{G}_2$ if ψ satisfies the following conditions:

- $\{u, v\} \in E(G_1)$ if and only if $\{\psi(u), \psi(v)\} \in E(G_2)$ for all $u, v \in V(G_1)$;
- $\mathrm{col}_{\boldsymbol{C}_1}(v) = \mathrm{col}_{\boldsymbol{C}_2}(\psi(v))$ for all $v \in V(G_1)$.

We say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are *isomorphic* if such ψ exists.

2.2 Vertex integrity

A *$\mathsf{vi}(k)$-set* S of a graph G is a set of vertices such that the number of vertices of every component of $G - S$ is at most $k - |S|$. The *vertex integrity* of a graph G, denoted by $\mathsf{vi}(G)$, is the minimum integer k such that there is a $\mathsf{vi}(k)$-set of G. In other words, it can be defined as follows:

$$\mathsf{vi}(G) = \min_{S \subseteq V(G)} \left\{ |S| + \max_{C \in \mathrm{cc}(G-S)} |V(C)| \right\},$$

where $\mathrm{cc}(G - S)$ is the set of connected components of $G - S$. A $\mathsf{vi}(k)$-set of G, if any exists, can be found in $O(k^{k+1}n)$ time [13], where n is the number of vertices in G.

As mentioned above, the concept of vertex integrity was originally introduced in the context of network vulnerability [2], but recently it and its close relatives are used as structural parameters in algorithmic studies. The *safe number* was introduced with a similar motivation [22] and later shown to be (non-trivially) equivalent to the vertex integrity in the sense that the safe number is bounded if and only if so is the vertex integrity for every graph [21]. The definition of *fracture number* is almost the same as the one for vertex integrity, where the only difference is that it asks the maximum (instead of the sum) of the orders of S and a maximum component of $G - S$ to be bounded by k. The *ℓ-component order connectivity* [13] measures the size of S and the maximum order of a component of $G - S$ separately, and defined to be the minimum size k of a set S such that each component of $G - S$ has order at most ℓ. For example, 1-component order connectivity is exactly the vertex cover number. Also, the 2-component order connectivity is studied as the matching-splittability [31]. A graph has vertex integrity at most k if and only if the graph has ℓ-component order connectivity at most $k - \ell$ for some ℓ.

The fracture number was used to design efficient algorithms for Integer Linear Programming [14], Bounded-Degree Vertex Deletion [26], and Locally Constrained Homomorphism [7]. The vertex integrity was used in the context of subgraph isomorphism on minor-closed graph classes [5], and then used to design algorithms for several problems that are hard on graphs of bounded treedepth such as Capacitated Dominating Set, Capacitated Vertex Cover, Equitable Coloring, Equitable Connected Partition[3], Imbalance, Maximum Common (Induced) Subgraph, and Precoloring Extension [28]. A faster algorithm for MSO Model Checking parameterized by vertex integrity is also known [37].

2.3 Monadic second-order logic

A *monadic second-order logic formula* (an MSO *formula*, for short) over p-colored graphs is a formula that matches one of the following, where x and y denote vertex variables, X denotes a vertex-set variable, C_i denotes a vertex-set constant (color): $E(x,y)$; $x=y$; $x\in X$; $x\in C_i$; $\exists x.\varphi$, $\forall x.\varphi$, $\exists X.\varphi$, $\forall X.\varphi$, $\varphi\wedge\psi$, $\varphi\vee\psi$, and $\neg\varphi$, where φ and ψ are MSO formulas. These symbols have the following semantic meaning: $E(x,y)$ means that x and y are adjacent; and the others are the usual ones. Additionally, for convenience, we introduce MSO symbols `true` and `false` that are always interpreted as true and false, respectively. Note that this version of MSO is often called MSO_1. In Section 4, we consider a variant called MSO_2, which has stronger expression power.

A variable is *bound* if it is quantified and *free* otherwise. An MSO formula is *closed* if it has no free variables and *open* otherwise. We assume that every free variable is a set variable, because a free vertex variable can be simulated by a free vertex-set variable with an MSO formula expressing that the set is of size 1. An *assignment* of an open MSO formula φ with s free set variables over G is a tuple $\boldsymbol{X}^G=(X_1^G,\dots,X_s^G)$ of s vertex sets $X_i^G\subseteq V(G)$. Let $\mathcal{G}$ be a p-colored graph, and φ be an MSO formula. If φ is closed, we write $\mathcal{G}\models\varphi$ if $\mathcal{G}$ satisfies the property expressed by φ. Otherwise, we write $(\mathcal{G},\boldsymbol{X}^G)\models\varphi$ where $\boldsymbol{X}^G$ is an assignment of φ if $\mathcal{G}$ and $\boldsymbol{X}^G$ satisfies the property expressed by φ.

From the definition of MSO, one can see that no MSO formula can distinguish isomorphic p-colored graphs. See e.g., [37] for a detailed proof.

▶ **Lemma 2.1** (Folklore). *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be isomorphic p-colored graphs. For every* MSO *formula φ, we have $\mathcal{G}_1\models\varphi$ if and only if $\mathcal{G}_2\models\varphi$.*

2.4 Extensions of MSO

We introduce an extension of MSO proposed by Knop, Koutecký, Masařík, and Toufar [34]. Let φ be an MSO formula with s free set variables $X_1,\dots,X_s$, and G be a graph with n vertices.

We introduce a linear constraint on the cardinalities of vertex sets $|X_i|$. A *global linear cardinality constraint* is an s-ary relation R expressed by a linear inequality $a_1|X_1|+\cdots+a_s|X_s|\leq b$, where a_i and b are integers and the arguments X_i are the free variables of φ. In the extension of MSO introduced later, global cardinality constraints are used as atomic formulas.

[3] In [15], Equitable Connected Partition was shown to be W[1]-hard parameterized simultaneously by pathwidth, feedback vertex set number, and the number of parts. In the full version, we strengthen the W[1]-hardness by replacing pathwidth in the parameter with treedepth.

A *local linear cardinality constraint* of G on φ is a mapping $\alpha_i^G \colon V(G) \to 2^{[n]}$, where $\alpha_i^G(v) = [l_i^v, u_i^v]$ with some integers l_i^v and u_i^v. Each α_i^G is a constraint on the number of neighbors of each vertex that are in X_i. We say that an assignment $\boldsymbol{X}^G = (X_1^G, \dots X_s^G)$ *obeys* a tuple $\boldsymbol{\alpha}^G = (\alpha_1^G, \dots, \alpha_s^G)$ of local linear cardinality constraints if $|X_i \cap N(v)| \in \alpha_i^G(v)$ for all $v \in V(G)$ and $i \in [s]$.

An $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ *formula* on a p-colored graph $\mathcal{G} = (G, \boldsymbol{C})$ is a tuple $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$ where φ, $\boldsymbol{R}$, and $\boldsymbol{\alpha}^G$ are defined as follows. The tuple $\boldsymbol{R} = (R_1, \dots, R_g)$ is a tuple of global linear cardinality constraints, and $\boldsymbol{\alpha}^G = (\alpha_1^G, \dots, \alpha_s^G)$ is a tuple of local linear cardinality constraints. The formula φ is an MSO formula with s free set variables that additionally has the g global linear cardinality constraints R_i as symbols. Now we write $(\mathcal{G}, \boldsymbol{R}, \boldsymbol{X}^G) \models \varphi$ if $(\mathcal{G}, \boldsymbol{X}^G) \models \varphi'$ where φ' is an ordinary MSO formula obtained from φ by replacing every symbol R_i with the symbol `true` or `false` representing the truth value of the formula $(|X_1^G|, \dots, |X_s^G|) \in R_i$.

Our problem, $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ Model Checking, is defined as follows.

$\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ Model Checking
Input: A p-colored graph $\mathcal{G} = (G, \boldsymbol{C})$ and an $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ formula $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$.
Question: Is there an assignment $\boldsymbol{X}^G = (X_1^G, \dots, X_s^G)$ of φ such that $(\mathcal{G}, \boldsymbol{R}, \boldsymbol{X}^G) \models \varphi$ and $\boldsymbol{X}^G$ obeys $\boldsymbol{\alpha}^G$?

It is known that $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ Model Checking is fixed-parameter tractable parameterized by neighborhood diversity [34], W[1]-hard parameterized by treedepth and feedback vertex set [34], and W[1]-hard parameterized by twin-cover [33].

3 Model checking algorithm

In this section, we present our main result, the fixed-parameter algorithm for $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ Model Checking parameterized by vertex integrity. Before going into the details, let us sketch the rough and intuitive ideas of the algorithm. Recall that our goal is to find a tuple of vertex sets in a graph of bounded vertex integrity that satisfies

- an MSO formula φ equipped with global linear cardinality constraints, and
- local linear cardinality constraints.

We first show that for the ordinary MSO Model Checking with a fixed formula on graphs of bounded vertex integrity, there is a small number of equivalence classes, called *shapes*, of tuples of vertex sets such that two tuples of the same shape are equivalent under the formula. To use the concept of shapes, we remove the global constraints from φ by replacing each of them with a guessed truth value and we find a solution that meets the guesses. Let φ' be the resultant (ordinary) MSO formula. We guess the shape of the solution and check whether a tuple with the guessed shape satisfies φ' using known efficient algorithms. If the guessed shape passed this test, then we check whether there is a tuple with the shape satisfying the global and local cardinality constraints. We can do this by expressing the rest of the problem as an integer linear programming (ILP) formula as often done for similar problems (see e.g., [34]). The ILP formula we construct has constraints for forcing a solution to be found

- to have the guessed shape,
- to satisfy the guessed global cardinality constraints, and
- to satisfy the local cardinality constraints.

The first two will be straightforward from the definitions given below. For the local cardinality constraints, we observe that after guessing the intersections of a $\mathsf{vi}(k)$-set S and each set in the solution, we know whether all vertices in $V(G) - S$ obeys the local cardinality constraints.

Thus we only need to express the local cardinality constraints in ILP for the vertices in S. Finally, we will observe that the number of variables and constraints in the constructed ILP formula depends only on k and $|\varphi|$. This will give us the desired result.

In the next subsections, we formally describe and prove the ideas explained above.

3.1 MSO model checking

Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph and S be a subset of $V(G)$. We define an equivalence relation of the components of $G - S$ as follows. Two components A_1 and A_2 of $G - S$ have the same $(\mathcal{G}, S)$-*type* if there is an isomorphism ψ from $(G[S \cup A_1], \boldsymbol{C}|_{S \cup A_1})$ to $(G[S \cup A_2], \boldsymbol{C}|_{S \cup A_2})$ such that the restriction $\psi|_S$ is the identity function. We call such an isomorphism a $(\mathcal{G}, S)$-*type isomorphism*. Clearly, having the same type is an equivalence relation. We say that a component A of $G - S$ is of $(\mathcal{G}, S)$-type t (or just type t) by using a canonical form t of the members of the $(\mathcal{G}, S)$-type equivalence class of A. Denote by $\tau_{(\mathcal{G},S)}(A)$ the type of a component A of $G - S$. We will omit the index $(\mathcal{G}, S)$ if it is clear from the context.

We define the canonical form of a $(\mathcal{G}, S)$-type as the "lexicographically" smallest one in the equivalence class in some sense (see [28] for such canonical forms of uncolored graphs). If S is a $\mathsf{vi}(k)$-set, then in time depending only on $p + k$ we can compute the canonical form of the equivalence class that a component A of $G - S$ belongs to. Thus we can compute (the canonical forms of) all $(\mathcal{G}, S)$-types in time $f(p + k)|G|^{O(1)}$ for some computable function f. Furthermore, in time $f'(p + s + k)|G|^{O(1)}$ for some computable function f', we can compute all $(\mathcal{G}', S)$-types for all $\mathcal{G}'$ obtained from $\mathcal{G}$ by adding s new colors; that is, $\mathcal{G}' = (G, \boldsymbol{C} \dot{+} \boldsymbol{X})$ for some $\boldsymbol{X} \in (V(G))^s$.

The next lemma, due to Lampis and Mitsou [37], is one of the main ingredients of our algorithm. It basically says that in the ordinary MSO Model Checking we can ignore some part of a graph if it has too many parts that have the same type.

▶ **Lemma 3.1** ([37]). *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph G, $S \subseteq V$, A be a component of $G - S$, $|A| \leq k$, and φ be a closed* MSO *formula with q quantifiers. If there are at least $2^{kq} + 1$ type $\tau(A)$ components in $G - S$, then $(G, \boldsymbol{C}) \models \varphi$ if and only if $(G - V(A), \boldsymbol{C}') \models \varphi$, where $\boldsymbol{C}'$ is the restriction of $\boldsymbol{C}$ to $V(G) \setminus V(A)$.*

Lemma 3.1 leads to the following concept "shape", which can be seen as equivalence classes of assignments.

▶ **Definition 3.2** (Shape). *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph, S be a $\mathsf{vi}(k)$-set of G, and φ be an* MSO *formula with s free set variables $(X_1, \ldots, X_s)$ and q quantifiers. Let $\mathcal{T}$ be the set of all $(\mathcal{G}, S)$-types, and $\mathcal{T}'$ be the set of all possible $(\mathcal{G}', S)$-types in $(p + s)$-colored graphs $\mathcal{G}'$ obtained from $\mathcal{G}$ by adding s new colors.*

*An S-*shape *is the pair (σ_S, σ) of a function $\sigma_S \colon S \to 2^{\{X_1, \ldots, X_s\}}$ and a function $\sigma \colon \mathcal{T}' \to [0, 2^{kq}] \cup \{\top\}$.*

*Let $\boldsymbol{X}^G = (X_1^G, \ldots, X_s^G)$ be an assignment of φ, and $\mathcal{G}' = (G, \boldsymbol{C} \dot{+} \boldsymbol{X}^G)$. The S-*shape *of $\boldsymbol{X}^G$ is (σ_S, σ) if the following conditions are satisfied:*

- *for each $i \in [s]$ and $v \in S$, $X_i \in \sigma_S(v)$ if and only if $v \in X_i^G$;*
- *for each $t' \in \mathcal{T}'$,*

$$\sigma(t') = \begin{cases} c(t') & c(t') \leq 2^{kq}, \\ \top & \text{otherwise}, \end{cases}$$

where $c(t')$ is the number of $(\mathcal{G}', S)$-type t' components of $G - S$.

Let (σ_S, σ) be an S-shape. If there is an assignment $\boldsymbol{X}^G$ of φ such that the S-shape of $\boldsymbol{X}^G$ is (σ_S, σ), we say that the S-shape (σ_S, σ) is *valid*.

The following lemma indicates that S-shapes act as a sort of equivalence classes.

▶ **Lemma 3.3.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph, S be a $\mathsf{vi}(k)$-set of G, and φ be an* MSO *formula with s free set variables $(X_1, \ldots, X_s)$. Let $\boldsymbol{X}^G$ and $\boldsymbol{Y}^G$ be assignments of φ such that their shapes are equal. Then, $(\mathcal{G}, \boldsymbol{X}^G) \models \varphi$ if and only if $(\mathcal{G}, \boldsymbol{Y}^G) \models \varphi$.*

Proof. Let $\mathcal{G}_X = (G, \boldsymbol{C} \dot{+} \boldsymbol{X}^G)$, $\mathcal{G}_Y = (G, \boldsymbol{C} \dot{+} \boldsymbol{Y}^G)$, and (σ_S, σ) be the S-shape of $\boldsymbol{X}^G$ (and of $\boldsymbol{Y}^G$). Then φ can be seen as a closed MSO formula for the $(p+s)$-colored graphs $\mathcal{G}_X$ and $\mathcal{G}_Y$. Thus we can apply Lemma 3.1 to $\mathcal{G}_X$, S, and φ, and obtain a graph $\mathcal{G}'_X = (G'_X, \boldsymbol{C}'_X)$, such that $\mathcal{G}_X \models \varphi$ if and only if $\mathcal{G}'_X \models \varphi$ and the number of each type t components of $G'_X - S$ is at most 2^{kq}, where q is the number of quantifiers in φ. We also obtain a graph $\mathcal{G}'_Y$ in the same way as for $\mathcal{G}'_X$. This reduction does not delete any vertex of S. The number of components for each type t of $G'_X - S$ or $G'_Y - S$ is $\sigma(t)$ if $\sigma(t) \neq \top$ and 2^{kq} if $\sigma(t) = \top$. Therefore, there is an isomorphism from $\mathcal{G}'_X$ to $\mathcal{G}'_Y$, and thus $\mathcal{G}'_X \models \varphi$ if and only if $\mathcal{G}'_Y \models \varphi$ by Lemma 2.1. ◀

Now, we estimate the number of candidates for S-shapes. Observe that in Definition 3.2, the number of candidates for σ_S depends only on k and s. The size of $\mathcal{T}$ depends only on k and p because it is at most the product of the number of $k \times k$ adjacency matrices and the number of p-color lists for graphs of at most k vertices. Similarly, the size of $\mathcal{T}'$ depends only on k, p and s. Since σ is a function from $\mathcal{T}'$ to $[0, 2^{kq}] \cup \{\top\}$, the number of candidates for σ depends only on k, p, s, and q. Thus the number of S-shapes depends only on k, p, s, and q.

▶ **Observation 3.4.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph with n vertices, S be a $\mathsf{vi}(k)$-set of G, and φ be an* MSO *formula with s free set variables and q quantifiers. The number of S-shapes depends only on k, p, s and q.*

3.2 Pre-evaluating the global constraints

Recall that in an $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula, the global cardinality constraints are used as atomic formulas. Namely, each of them takes the value true or false depending on the cardinalities of the free variables. To separate these constraints from the model checking process, the approach of pre-evaluation was used in the previous studies [27, 34].

▶ **Definition 3.5** (Pre-evaluation)**.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph, and $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$ be an $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula where $\boldsymbol{R} = (R_1, \ldots, R_g)$. We call a function $\gamma\colon \{R_1, \ldots, R_g\} \to \{\mathtt{true}, \mathtt{false}\}$ a* pre-evaluation*. Denote by $\gamma(\varphi)$ the* MSO *formula that obtained by mapping each global linear cardinality constraints R_i by γ.*

Since each global linear cardinality constraint R_i can be represented by a linear inequality, so is its complement $\bar{R}_i = [0, n]^s \setminus R_i$. Thus, for a pre-evaluation γ, the integers $x_1, \ldots, x_s \in [0, n]$ that satisfies the following conditions can be represented by a system of linear inequalities:

- If $\gamma(R_i) = \mathtt{true}$, then $(x_1, \ldots, x_s) \in R_i$.
- Otherwise, $(x_1, \ldots, x_s) \notin R_i$.

Denote by $\boldsymbol{R}_\gamma(x_1, \ldots, x_s)$ this system of linear inequalities. If an assignment $\boldsymbol{X}^G = (X^G_1, \ldots, X^G_s)$ of φ satisfies the system of inequalities $\boldsymbol{R}_\gamma(|X^G_1|, \ldots, |X^G_s|)$, we say that $\boldsymbol{X}^G$ *meets* the pre-evaluation γ.

3.3 Making the local constraints uniform

Observe that for a $\mathsf{vi}(k)$-set S of a graph G, a vertex v of a component of $G-S$ has at most $k-1$ neighbors. In other words, $|N(v)| \in [0, k-1]$ for each vertex $v \in V(G-S)$. Therefore, $|N(v) \cap X| \in \alpha(v)$ if and only if $|N(v) \cap X| \in \alpha(v) \cap [0, k-1]$ for every combination of $X \subseteq V(G)$, $\alpha\colon V(G) \to [0, n]$, and $v \in V(G-S)$. Thus, we can reduce the range of local constraints as follows.

▶ **Observation 3.6.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph, S be a $\mathsf{vi}(k)$-set of G, $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$ be an $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula where $\boldsymbol{\alpha}^G = (\alpha_1, \dots, \alpha_s)$, and $\boldsymbol{X}^G$ be an assignment of φ. Denote by $\boldsymbol{\beta}^G$ the local constraints obtained from $\boldsymbol{\alpha}^G$ by restricting to $\alpha_i^G(v) \cap [0, k-1]$ for each $v \in V(G) \setminus S$ and $i \in [s]$. Then, $\boldsymbol{X}^G$ obeys $\boldsymbol{\alpha}^G$ if and only if $\boldsymbol{X}^G$ obeys $\boldsymbol{\beta}^G$.*

▶ **Definition 3.7** (Uniform)**.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph with n vertices, S be a $\mathsf{vi}(k)$-set of G, and $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$ be an $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula where $\boldsymbol{\alpha}^G = (\alpha_1, \dots, \alpha_s)$. We say that the graph $\mathcal{G}$ is* uniform *on the local constraints $\boldsymbol{\alpha}^G$ if for every pair of components A_1 and A_2 of $G-S$ with the same $(\mathcal{G}, S)$-type, there is a $(\mathcal{G}, S)$-type isomorphism ψ from A_1 to A_2 such that $\alpha_i^G(v) = \alpha_i^G(\psi(v))$ for all $i \in [s]$ and $v \in V(A_1)$.*

We can obtain a uniform graph $\mathcal{G}'$ on $\boldsymbol{\alpha}^G$ from nonuniform graph $\mathcal{G} = (G, \boldsymbol{C})$ on $\boldsymbol{\alpha}^G$ as follows. For each $i \in [s]$, assign to every vertex $v \in V(G) \setminus S$ a new color $C^i_{\alpha_i(v)}$ corresponding to the local constraints $\alpha_i(v)$. Then we obtain a uniform graph $\mathcal{G}' = (G, \boldsymbol{C} \dot{+} (C^i_B)_{i \in [s], B \subseteq [0,k-1]})$. The number of new colors added to $\mathcal{G}'$ is at most sk^2.

▶ **Lemma 3.8.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph with n vertices, S be a $\mathsf{vi}(k)$-set of G, and $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$ be an $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula where $\boldsymbol{\alpha}^G = (\alpha_1, \dots, \alpha_s)$. Assume that the graph $\mathcal{G}$ is* uniform *on the local constraints $\boldsymbol{\alpha}^G$. Let $\boldsymbol{X}^G = (X_1^G, \dots, X_s^G)$ and $\boldsymbol{Y}^G = (Y_1^G, \dots, Y_s^G)$ be assignments of φ with the same S-shape (σ_S, σ). Then for each $i \in [s]$ and $v \in V(G) \setminus S$, $|X_i^G \cap N(v)| \in \alpha_i^G(v)$ if and only if $|Y_i^G \cap N(v)| \in \alpha_i^G(v)$.*

Proof. By symmetry, it suffices to prove the only-if direction. Assume that $|X_i^G \cap N(v)| \in \alpha_i^G(v)$ for each $i \in [s]$ and $v \in V(G) \setminus S$. Let $\mathcal{G}_X = (G, \boldsymbol{C} \dot{+} \boldsymbol{X}^G)$, $\mathcal{G}_Y = (G, \boldsymbol{C} \dot{+} \boldsymbol{Y}^G)$, and A_Y be a component of $G-S$. Since the S-shape of $\boldsymbol{X}^G$ and $\boldsymbol{Y}^G$ are the same, there is a component A_X of $G-S$ such that the $(\mathcal{G}_X, S)$-type of A_X is equal to the $(\mathcal{G}_Y, S)$-type of A_Y. Then, there is an isomorphism ψ from A_Y to A_X such that $|Y_i^G \cap N(v)| = |X_i^G \cap N(\psi(v))|$ and $\alpha_i^G(v) = \alpha_i^G(\psi(v))$ for each $v \in A_Y$ and $i \in [s]$, because $\mathcal{G}$ is uniform on $\boldsymbol{\alpha}$. Therefore, $|Y_i^G \cap N(v)| = |X_i^G \cap N(\psi(v))| \in \alpha_i^G(\psi(v)) = \alpha_i^G(v)$ for each $v \in V(G) \setminus S$ and $i \in [s]$. ◀

3.4 The whole algorithm

We reduce the feasibility test of global and local constraints to the feasibility test of an ILP formula with a small number of variables. The variant of ILP we consider is formalized as follows.

p-Variable Integer Linear Programming Feasibility (p-ILP)
Input: A matrix $A \in \mathbb{Z}^{m \times p}$ and a vector $\boldsymbol{b} \in \mathbb{Z}^m$.
Question: Is there a vector $\boldsymbol{x} \in \mathbb{Z}^p$ such that $A\boldsymbol{x} \leq \boldsymbol{b}$?

Lenstra [38] showed that p-ILP is fixed-parameter tractable parameterized by the number of variables p, and this algorithmic result was later improved by Kannan [32] and by Frank and Tardos [19].

▶ **Theorem 3.9** ([19,32,38])**.** *p-ILP can be solved using $O(p^{2.5p+o(p)} \cdot L)$ arithmetic operations and space polynomial in L, where L is the number of the bits in the input.*

The next technical lemma is the main tool for our algorithm.

▶ **Lemma 3.10.** *Let $\mathcal{G} = (G, \boldsymbol{C})$ be a p-colored graph, S be a $\mathsf{vi}(k)$-set of G, and $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$ be an $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula where φ has s free set variables $X_1, \dots X_s$, $\boldsymbol{R} = (R_1, \dots, R_g)$, and $\boldsymbol{\alpha}^G = (\alpha_1, \dots, \alpha_s)$. Assume that $\mathcal{G}$ is uniform on $\boldsymbol{\alpha}^G$. Then, there is an algorithm that given a valid S-shape (σ_S, σ), decides whether there exists an assignment $\boldsymbol{X}^G = (X_1^G, \dots, X_s^G)$ such that its S-shape is (σ_S, σ), $(\mathcal{G}, \boldsymbol{X}^G, \boldsymbol{R}) \models \varphi$, and $\boldsymbol{X}^G$ obeys $\boldsymbol{\alpha}^G$ in time $f(k, |\varphi|)n^{O(1)}$ for some computable function f.*

Proof. Our task is to find an assignment $\boldsymbol{X}^G$ such that
1. the S-shape of $\boldsymbol{X}^G$ is (σ_S, σ),
2. $(\mathcal{G}, \boldsymbol{X}^G, \boldsymbol{R}) \models \varphi$, and
3. $|X_i^G \cap N(v)| \in \alpha_i^G(v)$ for all $v \in V(G)$ and $i \in [s]$.

Condition 1 can be handled easily by linear inequalities in our ILP formulation. Condition 2 is equivalent to the condition that there exists a pre-evaluation γ such that $(\mathcal{G}, \boldsymbol{X}^G) \models \gamma(\varphi)$, and $\boldsymbol{X}^G$ meets γ. We check whether $(\mathcal{G}, \boldsymbol{X}^G) \models \gamma(\varphi)$ and whether $\boldsymbol{X}^G$ meets γ separately. Furthermore, Condition 3 is checked separately for vertices in S and for vertices in $V(G) \setminus S$.

Step 1. Guessing and evaluating a pre-evaluation for the global constraints. We guess a pre-evaluation γ from $2^g \leq 2^{|\varphi|}$ candidates. We check whether each assignment $\boldsymbol{X}^G$ with S-shape (σ_S, σ) satisfies the MSO formula $\gamma(\varphi)$, i.e., $(\mathcal{G}, \boldsymbol{X}^G) \models \gamma(\varphi)$. By Lemma 3.3, we only need to check whether $(\mathcal{G}, \boldsymbol{X}^G) \models \gamma(\varphi)$ for an arbitrary assignment $\boldsymbol{X}^G$ with S-shape (σ_S, σ). This can be done in $f(k, |\varphi|)n^{O(1)}$ time [8, 37]. Note that even if $(\mathcal{G}, \boldsymbol{X}^G) \models \gamma(\varphi)$ is true, this arbitrarily chosen $\boldsymbol{X}^G$ may not meet γ. In Step 3, we find an assignment with S-shape (σ_S, σ) that meets γ.

Step 2. Checking the local constraints for the vertices in $V(G) \setminus S$. By Lemma 3.8, we can check whether all shape-(σ_S, σ) assignments satisfy the local constraints for the vertices in $V(G) \setminus S$ by constructing an arbitrary assignment $\boldsymbol{Y}^G$ of S-shape (σ_S, σ) and testing whether $|Y_i^G \cap N(v)| \in \alpha_i$ for all $v \in V(G) \setminus S$ and $i \in [s]$. Since constructing an assignment $\boldsymbol{Y}^G$ can be done in $f(k, |\varphi|)n^{O(1)}$ time, this test can be done in $f(k, |\varphi|)n^{O(1)}$ time.

Step 3. Constructing a system of linear inequalities for the remaining constraints. By Steps 1 and 2, it suffices to check whether there exists an assignment $\boldsymbol{X}^G = (X_1^G, \dots, X_s^G)$ that satisfies the following conditions:
1. the S-shape of $\boldsymbol{X}^G$ is (σ_S, σ),
2. $\boldsymbol{X}^G$ meets the pre-evaluation γ, and
3. $\boldsymbol{X}^G$ obeys the local constraints $\boldsymbol{\alpha}^G$ for the vertices in S.

To this end, we construct a system of linear inequalities as follows.

In the following, we denote by $\mathcal{G}'$ the $(p+s)$-colored graph $(G, \boldsymbol{C} \dot{+} \boldsymbol{X}^G)$, where $\boldsymbol{X}^G$ is a hypothetical solution we are searching for.

Let $\mathcal{T}$ be the set of all $(\mathcal{G}, S)$-types. For every $t \in \mathcal{T}$, the number of type-t components of $G - S$ is denoted by n_t. Let $\mathcal{T}'$ be the set of all possible $(\mathcal{H}, S)$-types in $(p+s)$-colored graphs $\mathcal{H}$ obtained from $\mathcal{G}$ by adding s new colors. Observe that $\mathcal{T}'$ is a superset of the set of all $(\mathcal{G}', S)$-types, no matter how $\boldsymbol{X}^G$ is chosen. For every $t' \in \mathcal{T}'$, the $(\mathcal{G}, S)$-type of a type-t' component is uniquely determined and is denoted by $t'|_p$. (This notation comes from the fact that the $(\mathcal{G}, S)$-type of a type-t' component can be determined by considering the first p-colors.) For every $t' \in \mathcal{T}'$, we introduce the variable $x_{t'}$ that represents the number of $(\mathcal{G}', S)$-type t' components. The condition that the variables $x_{t'}$ agree with σ can be expressed as follows:

$$\sum_{t' \in \mathcal{T}',\, t'|_p = t} x_{t'} = n_t \qquad \text{for every } t \in \mathcal{T},$$
$$x_{t'} = \sigma(t') \qquad \text{for every } t' \in \mathcal{T}' \text{ such that } \sigma(t') \neq \top,$$
$$x_{t'} \geq 2^{kq} + 1 \qquad \text{for every } t' \in \mathcal{T}' \text{ such that } \sigma(t') = \top.$$

For every $i \in [s]$, we introduce an auxiliary variable y_i that represents the size of the set X_i^G, which is determined by the variables $x_{t'}$. The variables y_i can be expressed as follows:

$$y_i = |\{v \in S \mid X_i \in \sigma_S(v)\}| + \sum_{t' \in \mathcal{T}'} x_{t'} \cdot \#_i(x_{t'}) \qquad \text{for every } i \in [s],$$

where $\#_i(x_{t'})$ is the number of vertices with color $p+i$ in a type-t' component, i.e., the number of vertices assigned to the variable X_i in a type-t' component. Then, as mentioned in Section 3.2, the global constraints that match the pre-evaluation γ can be represented by the system of inequalities $R_\gamma(y_1, \ldots, y_s)$.

Finally, we formulate the local constraints for the vertices in S into a system of inequalities. For every $v \in S$, $i \in [s]$, and $t' \in \mathcal{T}'$, the number of neighbors of v with color $p+i$ (i.e., in the set variable X_i) in a type-t' component is denoted by $d_{i,t'}(v)$ (i.e., $d_{i,t'}(u) = |N(u) \cap X_i^G \cap V(A)|$ where A is a type-t' component). All constants $d_{i,t'}(v)$ can be computed in $f(k, |\varphi|)n^{O(1)}$ time. For every $i \in [s]$ and $v \in S$, we introduce an auxiliary variable $z_{v,i}$ that represents the number of neighbors of v in the set X_i, which is determined by the variables $x_{t'}$. The variables $z_{v,i}$ can be expressed as follows:

$$z_{v,i} = |\{u \in N(v) \cap S \mid X_i \in \sigma_S(u)\}| + \sum_{t' \in \mathcal{T}'} d_{i,t'}(v) x_{t'} \quad \text{for every } v \in S,\ i \in [s].$$

Since the local constraints α_i^G can be expressed by $\alpha_i^G(v) = [l_i^v, u_i^v]$ with some integers l_i^v and u_i^v for every vertex v, the local constraints for vertices in S can be expressed as follows:

$$l_i^v \leq z_{v,i} \leq u_i^v \qquad \text{for every } v \in S, i \in [s].$$

By finding a feasible solution to the ILP formula constructed above, we can find a desired assignment $\boldsymbol{X}^G$. Since the number of the variables in the ILP formula depends only on k and $|\varphi|$, the lemma follows by Theorem 3.9. ◀

▶ **Theorem 3.11.** $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ *Model Checking is fixed-parameter tractable parameterized by* $\mathsf{vi}(G)$ *and* $|\varphi|$.

Proof. Let $k = \mathsf{vi}(G)$. Let S be a $\mathsf{vi}(k)$-set. Such a set can be found in $O(k^{k+1}n)$ time [13]. We construct a uniform graph $\mathcal{H} = (G, \boldsymbol{C}')$ on $\boldsymbol{\alpha}^G$ from the input graph $\mathcal{G} = (G, \boldsymbol{C})$ as described in Section 3.3. Here, the number of colors of $\mathcal{H}$ depends only on k, p, and s. We compute the $(\mathcal{H}, S)$-types of the components of $G - S$ and count the number of $(\mathcal{H}, S)$-type t components for each t. This can be done in $f(k, |\varphi|)n$ time with some computable function f.

We guess an S-shape (σ_S, σ) of an assignment of the input formula φ. By Observation 3.4, the number of candidates for (σ_S, σ) depends only on k, p, and s. We check whether the guess shape (σ_S, σ) is valid. This can be done by checking whether (σ_S, σ) is consistent with the number of components of all $(\mathcal{H}, S)$-types. Hence, this can be done in $f(k, |\varphi|)n$ time with some computable function f.

By Lemma 3.10 with the graph $\mathcal{H}$, $\mathsf{vi}(k)$-set S, and the input $\mathsf{MSO}_{\mathsf{Lin}}^{\mathsf{GL}}$ formula $(\varphi, \boldsymbol{R}, \alpha^G)$, the theorem follows. ◀

4 Extension to MSO_2

The MSO_2 (or GSO) logic on graphs is a generalization of MSO_1 that additionally allows edge variables, edge-set variables, and an atomic formula $I(x, y)$ meaning that the edge assigned to y is incident to the vertex assigned to x. Although MSO_2 is strictly stronger than MSO_1 in general, for graphs of bounded treewidth, the model checking problem for MSO_2 can be reduced to the one for MSO_1 in polynomial time [10]. Using a similar reduction, we show that the same holds for $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ on graphs of bounded vertex integrity.

Now we define the extension of $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ with MSO_2, which we call $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$. In $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$, the local cardinality constraints for vertex-set variables and the global cardinality constraints work in the same way as in $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$. The local cardinality constraints for an edge-set variable X at a vertex v restricts the number of edges in X incident to v. We also generalize the concept of p-colored graphs such that each color can contain edges as well. A $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ *formula* on a p-colored graph $\mathcal{G} = (G, \boldsymbol{C})$ is a tuple $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$, where $\boldsymbol{R} = (R_1, \ldots, R_g)$ and $\boldsymbol{\alpha}^G = (\alpha_1^G, \ldots, \alpha_s^G)$ are the global and local cardinality constraints, and φ is an MSO_2 formula with s free set variables that additionally equipped with symbols $R_1, \ldots, R_g$. The problem $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking is formalized as follows.

$\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking
Input: A p-colored graph $\mathcal{G} = (G, \boldsymbol{C})$, and a $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ formula $(\varphi, \boldsymbol{R}, \boldsymbol{\alpha}^G)$.
Question: Is there an assignment $\boldsymbol{X}^G = (X_1^G, \ldots, X_s^G)$ of φ such that $(\mathcal{G}, \boldsymbol{R}, \boldsymbol{X}^G) \models \varphi$ and $\boldsymbol{X}^G$ obeys $\boldsymbol{\alpha}^G$?

We can show the following theorem by presenting a reduction from from $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking to $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking (see the full version for the proof).

▶ **Theorem 4.1.** $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ *Model Checking is fixed-parameter tractable parameterized by* $\mathsf{vi}(G)$ *and* $|\varphi|$.

5 Concluding remarks

In this paper, we obtained an algorithmic meta-theorem for graphs of bounded vertex integrity in a framework introduced as an extension of MSO by Knop, Koutecký, Masařík, and Toufar [34]. Namely, we showed that $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking (or more generally, $\mathsf{GSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ Model Checking) is fixed-parameter tractable parameterized by vertex integrity. This result partially covers the results of the previous study [28]: some problems admit direct translations from their definitions to expressions in $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ (e.g., Equitable r-Coloring) and some need non-trivial modifications to make them expressible in $\mathsf{MSO}^{\mathsf{GL}}_{\mathsf{Lin}}$ (e.g., Capacitated Vertex Cover). For some other problems (e.g., Imbalance and Max Common Subgraph), we were not able to determine that they can be captured by our framework or not. Also, the result newly gives algorithms for Fair Evaluation Problems [33]. It would be interesting to ask whether there is a meta-theorem that can be applied to a larger class of problems parameterized by vertex integrity. (See the full version.)

We may also consider the fine-grained complexity of our problem. We did not explicitly state the time complexity of our fixed-parameter algorithms. If we carefully analyze the running time using the algorithm by Lampis and Mitsou [37], then we can show that the algorithms run in time triple exponential in a polynomial function of the parameter. For the ordinary MSO Model Checking, it is known that under ETH, there is no $2^{2^{o(k^2)}} n^{O(1)}$-time algorithm, where k is the vertex integrity of the input graph G and n is the number of vertices of G [37]. This double-exponential lower bound applies also to our generalized problem. Filling this gap would be an interesting challenge.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

2 Curtis A. Barefoot, Roger C. Entringer, and Henda C. Swart. Vulnerability in graphs – A comparative survey. *J. Combin. Math. Combin. Comput.*, 1:13–22, 1987.

3 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In *ESA 2020*, volume 173 of *LIPIcs*, pages 14:1–14:19, 2020. doi:10.4230/LIPIcs.ESA.2020.14.

4 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

5 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph isomorphism on graph classes that exclude a substructure. *Algorithmica*, 82(12):3566–3587, 2020. doi:10.1007/s00453-020-00737-z.

6 Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6):555–581, 1992. doi:10.1007/BF01758777.

7 Laurent Bulteau, Konrad K. Dabrowski, Noleen Köhler, Sebastian Ordyniak, and Daniël Paulusma. An algorithmic framework for locally constrained homomorphisms. *CoRR*, abs/2201.11731, 2022. arXiv:2201.11731.

8 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *RAIRO Theor. Informatics Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571.

10 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*. Cambridge University Press, 2012. URL: https://www.cambridge.org/knowledge/isbn/item5758776/.

11 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

12 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.

14 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In *IJCAI 2017*, pages 607–613, 2017. doi:10.24963/ijcai.2017/85.

15 Rosa Enciso, Michael R. Fellows, Jiong Guo, Iyad A. Kanj, Frances A. Rosamond, and Ondřej Suchý. What makes equitable connected partition easy. In *IWPEC 2009*, volume 5917 of *Lecture Notes in Computer Science*, pages 122–133, 2009. doi:10.1007/978-3-642-11269-0_10.

16 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.

17 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *ISAAC 2008*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305, 2008. doi:10.1007/978-3-540-92182-0_28.

18 Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.

19 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987. doi:10.1007/BF02579200.

20 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Log.*, 130(1-3):3–31, 2004. doi:10.1016/j.apal.2004.01.007.

21 Shinya Fujita and Michitaka Furuya. Safe number and integrity of graphs. *Discret. Appl. Math.*, 247:398–406, 2018. doi:10.1016/j.dam.2018.03.074.

22 Shinya Fujita, Gary MacGillivray, and Tadashi Sakuma. Safe set problem on graphs. *Discret. Appl. Math.*, 215:106–111, 2016. doi:10.1016/j.dam.2016.07.020.

23 Jakub Gajarský and Petr Hliněný. Kernelizing MSO properties of trees of fixed height, and some consequences. *Log. Methods Comput. Sci.*, 11(1), 2015. doi:10.2168/LMCS-11(1:19)2015.

24 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *IPEC 2011*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271, 2011. doi:10.1007/978-3-642-28050-4_21.

25 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In *MFCS 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430, 2012. doi:10.1007/978-3-642-32589-2_38.

26 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/s00453-020-00758-8.

27 Robert Ganian and Jan Obdržálek. Expanding the expressive power of monadic second-order logic on restricted graph classes. In *IWOCA 2013*, volume 8288 of *Lecture Notes in Computer Science*, pages 164–177, 2013. doi:10.1007/978-3-642-45278-9_15.

28 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. In *CIAC 2021*, volume 12701 of *Lecture Notes in Computer Science*, pages 271–285, 2021. doi:10.1007/978-3-030-75242-2_19.

29 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206, 2009.

30 Petr Hliněný, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008. doi:10.1093/comjnl/bxm052.

31 Bart M. P. Jansen and Dániel Marx. Characterizing the easy-to-find subgraphs from the viewpoint of polynomial-time algorithms, kernels, and turing kernels. In *SODA 2015*, pages 616–629, 2015. doi:10.1137/1.9781611973730.42.

32 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12:415–440, 1987. doi:10.1287/moor.12.3.415.

33 Dušan Knop, Tomás Masarík, and Tomás Toufar. Parameterized complexity of fair vertex evaluation problems. In *MFCS 2019*, volume 138 of *LIPIcs*, pages 33:1–33:16, 2019. doi:10.4230/LIPIcs.MFCS.2019.33.

34 Dušan Knop, Martin Koutecký, Tomáš Masařík, and Tomáš Toufar. Simplified algorithmic metatheorems beyond MSO: Treewidth and neighborhood diversity. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:12)2019.

35 Stephan Kreutzer. Algorithmic meta-theorems. In Javier Esparza, Christian Michaux, and Charles Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Note Series*, pages 177–270. Cambridge University Press, 2011. doi:10.1017/cbo9780511974960.006.

36 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

37 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In *ISAAC 2021*, volume 212 of *LIPIcs*, pages 34:1–34:15, 2021. doi:10.4230/LIPIcs.ISAAC.2021.34.

38 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

39 Li-Shin Lin and Sartaj Sahni. Fair edge deletion problems. *IEEE Trans. Computers*, 38(5):756–761, 1989. doi:10.1109/12.24280.

40 Tomáš Masařík and Tomáš Toufar. Parameterized complexity of fair deletion problems. *Discret. Appl. Math.*, 278:51–61, 2020. doi:10.1016/j.dam.2019.06.001.

41 Stefan Szeider. Not so easy problems for tree decomposable graphs. *Ramanujan Mathematical Society, Lecture Notes Series*, No. 13:179–190, 2010. arXiv:1107.1177.

42 Stefan Szeider. Monadic second order logic on graphs with local cardinality constraints. *ACM Trans. Comput. Log.*, 12(2):12:1–12:21, 2011. doi:10.1145/1877714.1877718.

External-Memory Dictionaries with Worst-Case Update Cost

Rathish Das
University of Liverpool, UK

John Iacono
Université libre de Bruxelles, Belgium
New York University, USA

Yakov Nekrich
Michigan Technological University, Houghton, MI, USA

Abstract

The B^ϵ-tree [Brodal and Fagerberg 2003] is a simple I/O-efficient external-memory-model data structure that supports updates orders of magnitude faster than B-tree with a query performance comparable to the B-tree: for any positive constant $\epsilon < 1$ insertions and deletions take $O(\frac{1}{B^{1-\epsilon}} \log_B N)$ time (rather than $O(\log_B N)$ time for the classic B-tree), queries take $O(\log_B N)$ time and range queries returning k items take $O(\log_B N + \frac{k}{B})$ time. Although the B^ϵ-tree has an optimal update/query tradeoff, the runtimes are amortized. Another structure, the write-optimized skip list, introduced by Bender et al. [PODS 2017], has the same performance as the B^ϵ-tree but with runtimes that are randomized rather than amortized. In this paper, we present a variant of the B^ϵ-tree with deterministic worst-case running times that are identical to the original's amortized running times.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Sorting and searching; Theory of computation → Design and analysis of algorithms

Keywords and phrases Data Structures, External Memory, Buffer Tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.21

Funding *Rathish Das*: Supported by the Canada Research Chairs Programme and NSERC Discovery Grants.
John Iacono: Supported by Fonds de la Recherche Scientifique-FNRS under Grants MISU F 6001 1 and CDR/OL J.0101.22 – 40008322.

1 Introduction

The external memory model of Aggarwal and Vitter [1], sometimes called the I/O model or the Disk-Access Model (DAM), has been the most successful model of computation for problems where the data can not fit into RAM and the transfers between main memory and external memory strongly dominate the runtime on real machines. External memory data structures [2, 3, 7, 8, 10, 11, 13, 15, 21, 22, 24, 25, 27] have played a significant role in improving the performance of applications involving large datasets; the above references are but a small sample of related work in the external memory model, DBLP currently lists over 400 publications with external memory in the title.

In the external memory model, there are two levels of memory: an internal memory of size M and an external memory of unbounded size connected to the internal memory. Data is transferred between the two levels of memory in contiguous blocks of size B. The cost of an algorithm or a data structure is measured by the number of block transfers between internal and external memory; all computation in internal memory is free. When designing an algorithm in the external-memory model, the values of M and B are known, in contrast to the cache-oblivious model [18].

The B-tree, introduced by Bayer and McCreight fifty years ago [5], is the archetypal data structure in the external memory model. It stores a set of totally ordered keys, and supports insertions, deletions, predecessor queries in time $O(\log_B N)$, and range searches returning k elements in time $O(\log_B N + \frac{k}{B})$ in the external memory model. The B-tree is simple, is a standard part of the CS curriculum, and in optimized form is widely implemented as the cornerstone structure of databases due to its excellent real-world performance.

In internal memory, updates and searches share the same $O(\log N)$ runtime which is achieved by classic structures such as the AVL tree and the Red-Black tree. However, in external memory it is possible to substantially speed up updates with little increase in the search time. This trade-off was explored primary from the lower-bound view by Brodal and Fagerberg [11]. On one extreme of this trade-off curve, B-trees have an optimal query (e.g., predecessor query) bound but have a slow update bound. On the other extreme of the trade-off curve, buffer-repository trees [13] have far better update bound than B-tree but have a poor query bound.

Brodal and Fagerberg [11] introduced B^ϵ-tree at a "sweet spot" on the update-query trade-off curve. B^ϵ-tree performs insertions orders of magnitude faster than B-tree with a query performance comparable to B-tree. The improvement in insertion performance of B^ϵ-tree comes due to the use of buffers at nodes. Instead of inserting an individual key into the tree, the key is queued in the buffer of the root node, and when a significant number of keys are buffered, they are flushed recursively to the next level of the tree. This buffering technique allows B^ϵ-tree to achieve an amortized update cost $O(\frac{1}{B^{1-\epsilon}} \log_B N)$, while slowing down the queries by but a $\frac{1}{\epsilon}$ factor (ϵ is a tuning parameter that is between 0 and 1); they showed this is optimal for a random insertion workload [11]. Data structures for other fundamental abstract data types (ADTs) in the external memory and cache oblivious models with fast updates have also been created; see for example cache-oblivious dictionaries [9], priority queues [4, 12, 14, 17, 19, 23], hashing (only exact search) [21] and point location [20].

The B^ϵ-tree structure is very simple and easy to implement, but it has one major flaw: the runtimes hold in the amortized sense only and in the worst case the entire structure may need to be modified to execute a single query. Such performance is clearly unacceptable in the target application of a database. As we describe below, the B^ϵ-tree is just a B-tree with fanout B^ϵ, and where there is a buffer on each internal node which caches updates and only distributes them to the children when the buffer is full. The deterministic worst-case running time guarantee is very poor due to *flushing cascades*. A flushing cascade occurs when flushing the buffer of a node triggers flushes into multiple children nodes, which in turn trigger flushes to their children and so on. Flushing cascades become most acute when two nodes are merged into a new node, and their buffers are merged into a single buffer. The resulting buffer can overflow if the merged buffer can not hold all the items of the original two buffers. A buffer overflow can trigger new flushes, which again cause cascades and more node merges in the whole tree. In the worst-case, a single update (insert or delete) could trigger modifications to $\Omega(N^{1-o(1)})$ nodes of the tree [6].

There have been several attempts to deterministically de-amortize the performance of B^ϵ trees. In Bender et al. [8], a variant of skip lists was presented where a query takes $O(\log_B N)$ I/O with high probability and an update takes $O(\frac{1}{B^{1-\varepsilon}} \log_B N)$ I/O amortized with high probability. Recently Bender et al. [6] introduced a randomized universal buffer flushing strategy that achieves an update bound of $O(\frac{1}{B^{1-\varepsilon}} \log_B N)$ with high probability (not amortized) for a variant of the B^ϵ-tree.

So, using the best previously known results, $O(\frac{1}{B^{1-\epsilon}} \log_B N)$ updates and $O(\log_B N)$ queries are possible in either the amortized sense, or randomized with high probability.

This paper. Our contribution is that these running times can be achieved deterministically in the worst-case (i.e., deamortized) with a structure based on the B^ϵ-tree.

It seems difficult to apply standard de-amortization techniques to our problem without sacrificing performance. The standard global re-building technique [26] works by copying insertions into a new tree using a background process. When all insertions are copied into a new tree, the old tree is discarded. This approach can guarantee that the height of the tree is bounded by $O(\log_B N)$. However, we can have a large sequence of leaves with a very small number of insertions in each leaf. This can significantly increase the cost of range queries; in the worst scenario we may have to visit $\Omega(N/B)$ leaf nodes in order to answer a range query. Another approach is to maintain the sizes of buffers in internal nodes using a background process. However, the merging of two nodes can lead to a cascade of deletions. Thus each round of the background process could take almost linear time as explained above [6].

In this paper we de-amortize the B^ε-tree through a combination of several techniques. At a high level, this involves having large leaves stored in a separate structure, and periodically splitting or merging selected leaves rather than when they reach certain sizes. Splitting and merging needs to be propagated up the structure, and splitting in particular can cause overflow which requires flushing data down the structure. And, all of this needs to be done while new updates keep arriving though a very careful choice of relevant parameters.

What does it mean to deamortize a structure whose per-operation runtime may be subconstant, as it is likely to be in the case of $O(\frac{1}{B^{1-\varepsilon}} \log_B N)$? This means that if the update cost of $\frac{1}{B^{1-\varepsilon}} \log_B N$ is subconstant, then a constant number of I/Os are executed every $\frac{B^{1-\epsilon}}{\log_B N}$ operations.

We proceed by reviewing the standard B^ϵ-tree in Section 2 and presenting our variant in Section 3.

2 The B^ϵ-tree

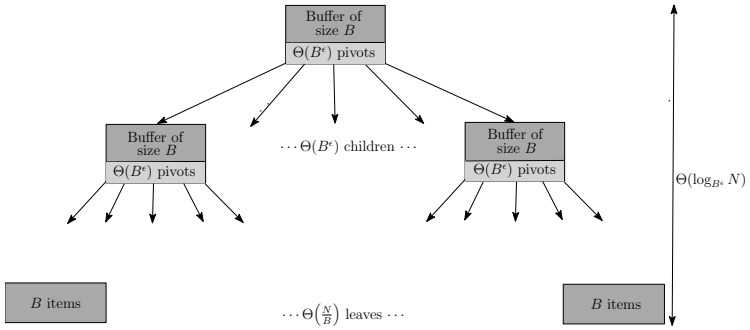


Figure 1 B^ϵ-tree.

Abstract data type

The data structure maintains a set S of keys (or key-value pairs), where the keys are constant-sized and come from a totally ordered universe, under the following operations:

- Insert(x): Adds key x to S under the precondition that $x \notin S$.
- Delete(x): Removes key x from S under the precondition that $x \in S$
- Predecessor(x): Returns the largest key $y \in S$ such that $y \leq x$
- Range-Report(x, y): Returns all keys in $[x, y] \cap S$ in sorted order.

We use N to denote $|S|$, and use the word *update* to generically refer to an insertion or deletion.

Structure description

See Figure 1. A B^ϵ-tree is just a standard B-tree where internal nodes have $\Theta(B^\epsilon)$ children and an additional buffer of size B which stores updates that are destined to the subtree of the node; leaves store $\Theta(B)$ keys. The buffers of internal nodes store updates, that is, both insertions and deletions.

Predecessor queries can be done with a single root-to-leaf traversal, pushing updates on the search path down to the leaf. During this process, paired insertions and deletions annihilate each other. In the case where such annihilations require restructuring via standard splits and merges, it is paid for in the amortized sense with the removal of the updates from the tree.

When a buffer is full, some of the updates are removed and copied to the buffer of a child, with the child being chosen so as to maximize the number of updates pushed down; as the buffer is size B, the keys are constant sized, and the number of children is $\Theta(B^\epsilon)$, there must be at least one child buffer with $\Omega(B^{1-\epsilon})$ updates to be pushed down to it. Determining this child and moving the data can be done in a constant number of block transfers. When leaves are full or become more than a constant factor empty, splits and merges are performed in the usual way, though merges may cause overflows of merged buffers which could cause more updates to be propagated, which could cause more merges, and this could continue as possibly effect every node in the structure. However, the amortized cost of an insertion/deletion is $O(\frac{1}{B^{1-\epsilon}} \log_B N)$ as each update will be involved in $O(\log_B N)$ buffer overflows at a unit cost of each, spread over $B^{1-\epsilon}$ items. We summarize the performance of B^ϵ trees:

▶ **Theorem 1** ([11]). *Given any constant ε, $0 < \varepsilon < 1$, the B^ϵ-tree supports updates in amortized time $O(\frac{1}{B^{1-\varepsilon}} \log_B N)$, predecessor searches in worst-case time $O(\log_B N)$, and range searches that return k elements in worst-case time $O(\log_B N + \frac{k}{B})$ in the external memory model.*

3 Our Data Structure

Out data structures performance is summarized by the following theorem, which we will spend the rest of this section proving:

▶ **Theorem 2.** *Given any constant ε, $0 < \varepsilon < 1$, our data structure supports updates in worst case time $O(\frac{1}{B^{1-\varepsilon}} \log_B N)$ (meaning $O(1)$ I/Os every $O(B^{1-\varepsilon} \log_B N)$ updates if $\frac{1}{B^{1-\varepsilon}} \log_B N < 1$), predecessor searches in worst-case time $O(\log_B N)$, and range searches that return k elements in worst-case time $O(\log_B N + \frac{k}{B})$ in the external memory model.*

3.1 Our Approach

We make several major changes in our structure compared to a standard buffered B^ϵ-tree in order to obtain worst-case bounds. We describe these changes at a high level first, and then give a full description of our data structure. Following the convention used in many previous papers, our data structure stores *updates*. An update is a key combined with a flag indicating whether it is an insertion or a deletion. To avoid confusion, we use update *operation* to refer to a insertion or deletion operation executed by the user, and *update* without the word "operation" to refer to a key/flag pair stored by the data structure. We define $\delta = \varepsilon/2$. We assume N is an upper bound on the number of keys logically stored by the structure, and will state several invariants of the structure in terms of N; should the actual number of keys logically stored vary polynomially from from N, the structure will need to be rebuilt, which we explain how to do in §3.6. We make the following major changes in our structure compared to the standard B^ϵ-tree:

- We increase the size of leaf nodes to $\Theta(B \log_B^2 N)$. Each leaf is stored in an auxiliary data structure implemented as a two-level tree, with fan-out $\Theta(\log_B N)$.
- Each internal node stores a buffer of size at most $B^{1-\delta}$. The only exception is we allow the root and one other node to have a buffer size double that, $2B^{1-\delta}$, because two buffers were just combined as a result of a merge.
- Instead of splitting and merging leaves when they reach some size thresholds, we alternate picking the largest leaf and splitting it (if it is large enough) and picking the smallest leaf and merging it with a sibling (if it is small enough), and then possibly splitting it. Such splits and merges are propagated up in the normal way. We show that this is sufficient to maintain the size invariants of the leaves.
- To ensure that a node's buffer size is within range, either because it is the root and updates have been added as the result of update operations or a merge caused the two sibling buffers to be combined, $B^{1-2\delta}$ items are moved from a buffer to a single child (*a flush*), and this flush is repeated down to a leaf. Multiple such flushes, up to B^{δ} may be required to reduce a merged buffer of size $2B^{1-\delta}$ down to $B^{1-\delta}$.
- We maintain auxiliary information in each internal node. Such information includes the key values needed for searching, which is standard, as well as augmented information that allows the algorithm to find the leaves storing the most/least updates. When changes are made to leaves, the auxiliary information may need to be updated in its ancestors.
- We alternate waiting for the user to execute $\Theta(\frac{B^{1-2\delta}}{\log_B N})$ update operations (as well as an unlimited number of queries), and doing structural work to maintain the invariants until there is an I/O. The various parameters are chosen with care so that the number of update operations does not overwhelm the rebalancing in progress.
- Queries do not change the structure. Instead, we guarantee by construction that all updates found in buffers of internal nodes on a root-to-leaf path fit in $\Theta(\log_B N)$ blocks and that the insertions on the path asymptotically dominate the deletions. Thus all of these updates can be copied into a temporary workspace, all insert-delete pairs are removed, the result of the predecessor query is determined and returned. Range queries work in a similar manner, with the insertions contained in the buffers that cover any range dominating the deletions. This approach avoids the need to answer queries by pushing down the changes to the root and causing a possibly uncontrolled number of cascading changes as in the classic B^{ϵ}-tree.

Summing up, our tree is leaf-heavy, among the updates stored in the buffers insertions asymptotically dominate deletions, most updates are stored in the leaves, and each leaf stores $\Theta(B \log_B^2 N)$ updates. The tree structure is maintained by regularly splitting and merging the leaves that are largest and smallest and not at some fixed threshold of size.

3.2 Invariants

As in a standard B^{ϵ} tree, internal nodes store buffers of updates that are destined for some leaf in their subtreee, and keys to guide the search. The leaves, however, are separate structures described in §3.4.

Child count. Every internal node has a number of children in the range $[\frac{1}{2}B^{\delta}, B^{\delta}]$. The root may have less children, but at least 2. This invariant can be violated by a single node at any given time, and may only violate it by ± 1.

Leaf size. Every leaf stores a number of updates in the range $[B \log_B^2 N, 5B \log_B^2 N]$. We do not count matching insertion/deletion pairs in a leaf as these will be eliminated.

Internal node buffer size. Each internal node has at most $B^{1-\delta}$ updates in its buffer. This invariant can be violated by two nodes at any given time. One of which is the root, and the other is a node that will have just had its buffer merged with a sibling. For these two nodes we ensure that there are at most $2B^{1-\delta}$ updates in their buffers. These nodes are said to be *overfull*.

Height. The height of the tree is always at most $c_h \log_B N$ for some constant c_h. This follows from the preceding invariants.

Auxiliary information. As is standard in a leaf oriented-tree, each internal node stores keys needed to guide searches. Additionally, each internal node stores which child contains the subtree with the leaf of maximum size, and what that size is, as well as the same information for the leaf of minimum size.

3.3 Updates

First we describe at a high level how update operations are executed, then present in Figure 2 the details, followed by justification of correctness and runtime.

One core subroutine is the *flush* which moves $B^{1-2\delta}$ updates from a node's overfull buffer to one of its children, and then repeats this process on that child until either a non-overfull buffer or a leaf is reached. If a buffer is overfull it has at least $B^{1-\delta}$ updates in its buffer. Thus, by pigeonhole there must be one of its at most B^δ children for which it can move $B^{1-2\delta}$ updates to.

At a high level, our algorithm can be thought of always being in the process of doing one of the following:

- While the root buffer is overfull, flush. This could require $\Theta(B^\delta)$ flushes.
- The largest leaf is split if it stores at least $4 \log_B^2 N$ updates. This split is propagated up the tree.
- The smallest leaf is merged if it stores less than $2 \log_B^2 N$ updates. This merge is propagated up the tree. However, each merge may cause a buffer to become too large, up to $2B^{1-\delta}$ instead of the normal limit of $B^{1-\delta}$; this is fixed by performing $O(B^\delta)$ flushes on the overfull buffer to remove the overfull condition. It is crucial to note that $O(B^\delta)$ flushes could be performed starting on *each* of the $O(\log_B N)$ nodes being split; as the flush is itself a loop that could visit $O(\log_B N)$ nodes, this creates a triple-nested loop whose innermost operations, one step of a flush, could run $\Theta(B^\delta \log_B^2 N)$ times.

One cycle will consist of either splitting or merging a leaf and propagating upwards, where after each time a node is split or merged the root is flushed. Figure 2 shows the full details of this.

The crucial point is that after every I/O in the above infinite process the algorithm stops and waits for the user to execute $\frac{B^{1-2\delta}}{c_i \log_B N}$ update operations, which it adds to the root's buffer, before continuing; the constant c_i is defined later.

We now argue the claimed runtime is correct, the claimed number of times each loop runs is correct, and the invariants hold. Arguing that the leaf size invariant holds is the most involved part of the result.

The leaf data structure: a preview

We include here a summary of the leaf operations and their performance so that we may analyze the main update procedure using the leaves as a black box. Section 3.4 describes how the leaves are implemented to achieve this. The leaf data structure will support the following subroutines; as in the description of the main procedure to execute update operations, these algorithms alternate execution and waiting for the user to execute update operations.

[1] Alternate between a *split phase* and a *merge phase*. Together, these are a *cycle*.
- [2] Set n_1 to the root. Move n_1 to the leaf with the most/least updates if a split/merge phase by repeatedly moving n_1 down to the appropriate child using the auxiliary information. (This takes $c_h \log_B N$ steps).
- [3] If a split phase and n_1's buffer has at least $4B \log_B^2 N$, split n_1; if a merge phase and n_1 has at most $2B \log_B^2 N$ updates, merge n_1 with one of its siblings and if this results in a leaf with more than $5B \log_B^2 N$ updates split it evenly.
- [4] **Propagate splits/merges up.** While n_1 is not null: (the parent of the root) (This loop will run at most $c_h \log_B N$ times).
  - [5] Move n_1 to its parent.
  - [6] Split/merge n_1 as in a B-tree if needed; also split or merge the buffer(s). Update the auxiliary information in n_1.
  - [7] Set n_2 to n_1.
  - [8] **Repeated flushing of split/merge node.** While n_2 is an internal node with an overfull buffer: (This loop will run at most B^δ times).
    - [9] Set n_3 to n_2.
    - [10] **Propagating a flush of a split/merge node downward.** While n_3 is an internal node with n_3 an overfull buffer: (This loop will run at most $c_h \log_B N$ times).
      - [11] Identify a child of n_3, call it n_4, for which the buffer of n_3 can move at least $B^{1-2\delta}$ elements from n_3's buffer to n_4's, this exists by pigeonhole.
      - [12] Move $B^{1-2\delta}$ items from n_3's buffer to n_4's.
      - [13] Set $n_3 = n_4$.
    - [14] **Update auxiliary information.** Move n_3 to its parent until it reaches the root while updating the auxiliary information. (This loop runs at most B^δ times).
  - [15] While the root buffer is overfull, set n_5 to the root and: (This loop runs at most B^δ times).
    - [16] **Flush from root.** While the buffer of n_5 is overfull: (This loop runs at most $c_h \log_B N$ times).
      - [17] Identify a child of n_5, call it n_6, for which the buffer of n_5 can move at least $B^{1-2\delta}$ elements from n_5's buffer to n_6's; this exists by pigeonhole.
      - [18] Move $B^{1-2\delta}$ items from n_5's buffer to n_6's.
      - [19] Set $n_5 = n_6$.
    - [20] **Update auxiliary information.** Moving n_5 to its parent until it reaches the root while updating the auxiliary information.

Figure 2 Full description of how update operations are executed. Note that this is presented as an infinite loop. In addition to the above description, every time there is an I/O the algorithm stops and waits for the user to execute $\frac{B^{1-2\delta}}{c_i \log_B N}$ update operations and then contininues until the next I/O occurs.

Bulk-insert of updates. $B^{1-2\delta}$ updates are added to the leaf. During the execution of this $O(\log_B N)$ I/Os are performed, with the algorithm stopping after each one to wait for the user to execute $\frac{B^{1-2\delta}}{c_i \log_B N}$ update operations which are added to the root buffer of the main tree.

Split and merge. These take $O(B^\delta \log_B N)$ I/Os to complete; the algorithm stops after each I/O to wait for the user to execute $\frac{B^{1-2\delta}}{c_i \log_B N}$ update operations which are added to the root buffer of the main tree.

Runtime

We require that after each I/O we wait for $\frac{B^{1-\epsilon}}{c_i \log_B N}$ update operations to be executed by the user (recall $2\delta = \epsilon$). This is exactly what having a worst case runtime of $O(\frac{\log_B N}{B^{1-\epsilon}})$ means, when $\frac{1}{B^{1-\epsilon}} < 1$, a constant number of I/Os every $\frac{B^{1-\epsilon}}{c_i \log_B N}$ updates.

We have presented the algorithm and analysis for the (common) case where $\frac{B^{1-\epsilon}}{c_i \log_B N}$ is at least one. If, however, we are in the (uncommon) case where $\frac{B^{1-\epsilon}}{c_i \log_B N}$ is less than one, instead wait for the user to execute a single update operation and add it to the root buffer every $\frac{c_i \log_B N}{B^{1-2\delta}}$ I/Os. This trivially gives the claimed runtime.

Loops

Here we argue that each loop runs the number of times indicated. The number of times each loop runs, except those of lines 8 and 15 of Figure 2, are clearly bounded by the height of the structure as they involve going up or down. However, the repeated flushing in lines 8 and 15 requires a different argument. If one of these loops run, it is because a flush is happening at one of the two nodes that may violate size of an internal node invariant, either the root (line 15) or a node that had just had it buffer merged with another (line 10). These nodes may have up to $2B^{1-\delta}$ updates in their buffers. Each execution of these loops will remove $B^{1-2\delta}$ elements from the node being flushed. This will require looping at most B^δ times to restore the invariant that they have at most $B^{1-\delta}$ updates in their buffer.

Children

As in a standard B or B^ϵ tree, the number of children may be out of range, and only by one, while in the process of propagating a split or merge up; this node where the splits and merges are in progress is n_1 in our presentation.

Internal node buffer size

In each execution of the loop at line 4, the total number of I/Os is at most $c_i B^\delta \log_B N$, for some c_i. This is where c_i is defined. Since $\frac{B^{1-2\delta}}{c_i \log_B N}$ update operations are executed per I/O by adding the updates to the root, this means $B^{1-\delta}$ updates are added to root per execution of the loop at line 4. In execution of the loop at line 4 we explicitly flush from the root in line 15 until the internal buffer size is not longer overfull.

The only other time an internal node's buffer becomes overfull is after a merge, but in line 10 flushes are performed until the node is no longer overfull.

Leaf size

In order to show that the leaf size invariant is maintained, that is each leaf holds from $B \log_B^2 N$ to $5B \log_B^2 N$ updates, we will need the following result which comes from Theorem 5 of [16]. Our formulation of this theorem will allow its easy use later.

▶ **Theorem 3.** *Let X be a set of real valued positive variables and let b be a positive constant. All variables $x \in X$ are initially 0. We execute the following process which proceeds in rounds and in each round may change the values of the $x \in X$ as follows:*

1. *Values may be decreased without restriction*
2. *The sum of the increases in x of those $x \in X$ that increase in a round is at most b.*
3. *During each round, the maximum value of x must be set to zero at some point.*

Additionally, new elements may be added to X with a value of 0 and elements with a value of 0 may be removed from X at any time. If $|X|$ is always bounded by m, there is a constant c_t such that at all times $x_j \leq c_t b \log m$ for all j.

▶ **Lemma 4.** *The leaf size invariant is always satisfied, that is, the number of updates in each leaf node is always between $B \log_B^2 N$ and $5B \log_B^2 N$.*

Proof. To avoid repetitive clutter, let τ denote $B \log_B^2 N$.

We examine how the number of updates in each leaf can change. These changes are caused by the flushing of updates into the leaves, splitting leaves, and merging leaves. We discuss each of these separately.

Let L be the set of leaves, and let $n(\ell)$ be the number of insertions stored in a leaf ℓ without a matching delete. Define $d(\ell)$ be $\max(0, n(\ell) - 4\tau)$, and $d'(\ell)$ be $\max(0, \tau - d(\ell))$. Thus, d is nonzero when a leaf is within τ of its upper limit and d' is nonzero when a leaf is within τ of its lower limit.

In a cycle, updates can be added to leaves when a flush reaches a leaf, this could happen in lines 12 and 18 of Figure 2. Multiplying by the number of times these lines may be executed in a cycle, the number of updates that are added to all leaves is at most $c_a B^{1-\delta} \log_B N$ for some constant c_a. This can cause $\sum_{\ell \in L} d(\ell)$ to increase by that much during a cycle as insertions are added to a leaf. When an deletion is added to a leaf, this will cause $n(\ell)$ to decrease by 1. Thus $\sum_{\ell \in L} d'(\ell)$ can also increase by $c_a B^{1-\delta} \log_B N$ per cycle.

Additionally, any split will cause the created leaves during the split to have $d(\ell)$ and $d'(\ell)$ be zero as they will have a size of at least 4τ and at most 5τ before the split, which results in a size of at least 2τ and at most $\frac{5}{2}\tau$.

If no split is performed then $n(\ell) < 4\tau$ for all leaves ℓ and thus $d(\ell)$ is zero for all leaves ℓ.

If a merge is performed, a node with $n(\ell)$ in the range $[\tau..2\tau]$ merges with its sibling which has $n(\ell)$ in the range $[\tau..5\tau]$, resulting a new leaf with size in the range $[2\tau..6\tau]$. If its size is in the range $[2\tau..4\tau]$ it has a $d(\ell)$ and $d'(\ell)$ value of 0. Otherwise its size is in the range $[4\tau..6\tau]$ and it is split evenly into two nodes each of which has size $[2\tau..3\tau]$ and thus has $d(\ell)$ and $d'(\ell)$ be 0.

Thus in each cycle we have shown that $\sum_{\ell \in L} d(\ell)$ and $\sum_{\ell \in L} d'(\ell)$ each increase by at most $c_a B^{1-\delta} \log_B N$ and the leaf ℓ with maximum $d(\ell)$ and ℓ' with maximum $d'(\ell)$ will both have their values d and d' reset to zero if a split/merge was performed, and if it was not, they were zero already. Thus we can apply Theorem 3 twice: one where the elements of X represent the $d(\ell)$ values, m represents the number of leaves (which is trivially at most N) and b represents $c_a B^{1-\delta} \log_B N$, and once using $d'(\ell)$ instead.

This yields bounds on $\max_\ell d(\ell)$ and $\max_\ell d'(\ell)$ of $(c_a B^{1-\delta} \log_B N) \cdot c_t \log N$, which is at most $B \log_B^2 N$, for large enough B.

The bounds on $d(\ell)$ and $d'(\ell)$ being at most τ immediately follows by the definition of d and d' that $n(\ell)$ is in the range $[\tau, 5\tau]$ which, recalling that $\tau = B \log_B^2 N$, is the claim of the Lemma. ◀

3.4 Leaf Data Structures

We now describe how the leaf data structures are implemented. Each leaf data structure supports adding $B^{1-2\delta}$ updates into the leaf, splits, merges, and searches. The main structure ensures that splits and merges are performed to maintain the leaf's size invariant.

We store each leaf ℓ as a two-level tree, and use the terminology *micro-root* and *micro-leaf* to avoid confusion with the main structure. Each leaf $T(\ell)$ consists of a micro-root with $\Theta(\log_B N)$ child micro-leaves. Each micro-leaf of $T(\ell)$ contains $\Theta(B \log_B N)$ updates in sorted order. The micro-root of $T(\ell)$ has a buffer with at most $B^{1-\delta} \log_B N$ updates. Every time when we flush $B^{1-2\delta}$ updates into ℓ we add them to the root buffer of $T(\ell)$. If the root buffer contains over $B^{1-\delta} \log_B N$ updates, we identify the micro-leaf ν where at least $B^{1-2\delta}$ updates can be moved. We flush those updates to ν and add them to blocks of ν. We can merge the newly flushed updates with the extant updates in ν in $O(\log_B N)$ I/Os, while maintaining the updates in sorted order, and annihilating matching insertion/deletion pairs on the same element. A deletion that is added to a micro-leaf is mutually annihilated with the insertion of the same element, which must be present in the leaf. Thus micro-leaves contain insertions only. Since deletions can be present only at the buffer of the micro-root of T_ℓ, there can be at most $B^{1-\delta} \log_B N$ deletions in any leaf node. Micro-leaves of $T(\ell)$ can be split and merged in a standard way so that each micro-leaf holds $\Theta(B \log_B N)$ updates.

When a leaf ℓ is split into ℓ_1 and ℓ_2, we distribute the micro-leaves of $T(\ell)$ among $T(\ell_1)$ and $T(\ell_2)$; one micro-leaf can be possibly split into two parts. The cost is $O(\log_B N)$ I/Os. When two leaves ℓ_1 and ℓ_2 are merged, all micro-leaves of $T(\ell_1)$ and $T(\ell_2)$ become the micro-leaves of $T(\ell)$. The buffers of the micro-roots of $T(\ell_1)$ and $T(\ell_2)$ are merged too. The cost of merging is $O(\log_B N)$ I/Os. After merging, the micro-root buffer of $T(\ell)$ can contain up to $2B^{1-\delta} \log_B N$ I/Os. We "repair" the micro-root buffer by moving excessive updates to micro-leaves. We can flush $B^{1-2\delta}$ updates to a micro-leaf in $O(\log_B N)$ I/Os. Hence the total cost of flushing all excessive updates to micro-leaves is $O(B^\delta \log_B N)$.

In all of these operations, every time there is an I/O the algorithm stop and waits for $\frac{B^{1-2\delta}}{c_i \log_B N}$ update operations to be executed by the user, adds these operations to the buffer of the root of the main tree, and then resumes.

3.5 Queries

To answer a range reporting query $[a, b]$, we identify all leaves that intersect with the range $[a, b]$. If a leaf ℓ intersects with but is not contained in $[a, b]$, we identify the micro-leaves that intersect with $[a, b]$. If a micro-leaf ℓ_m intersects with $[a, b]$, we traverse ℓ_m and make a list of insertions stored in $[a, b]$. If a micro-leaf is entirely contained in $[a, b]$, we list all its insertions. Let $L(\ell)$ denote the list of insertions found in ℓ. We examine updates stored in the root buffer of $T(\ell)$ and modify $L(\ell)$ accordingly. Since the updates in the root buffer are in sorted order, $L(\ell)$ can be modified in $O(\log_B N + |L(\ell)|/B)$ I/Os, where $|L|$ denotes the number of updates in a list L. If a leaf ℓ is entirely contained in $[a, b]$, we make a list $L(\ell)$ containing all insertions stored in micro-leaves and in the root buffer of $T(\ell)$ (minus the deletions in the root buffer of $T(\ell)$). This takes $O(|L(\ell)|/B)$ I/Os. There are at most two leaves that partially intersect with the query range $[a, b]$. Hence total cost of generating lists $L(\ell)$ for all relevant leaves ℓ is $O(k/B + \log_B N)$, where k is the number of insertions that must be reported. We then visit all ancestors of relevant leaves ℓ, add insertions from $[a, b]$ to an extra list L_a, and copy all updates from $[a, b]$ to internal memory. Finally we report all elements from $L(\ell)$, excluding the deletions. Assuming that the internal memory can hold $O(\log_B N \cdot B^{1-\delta})$ deletions, this can be done in $O(k/B)$ I/Os. Thus we can answer reporting

queries in $O(k/B + \log_B N)$ I/Os under assumption that the internal memory is not too small, $M > B \log_B N$. We note that B^ϵ trees as well as the structures of Bender et al. and Bender et al. [6, 8] all similarly require small non-constant amounts of memory for queries.

To answer a predecessor query q (i.e., to find the largest element that is not larger than q), we create a list L_1 of $B \log_B N$ largest insertions stored in the leaf nodes that are $\leq q$. This can be done in $O(\log_B N)$ I/Os using the same reporting procedure as described above. Since a leaf holds $\Theta(B \log_B^2 N)$ insertions, insertions in L are from at most two consecutive leaves. We examine all insertions stored in ancestors of these two leaves and make a list L_2 of all insertions $\leq q$. Let L_3 denote the list of $B \log_B N$ largest elements in L_2 and L_3. Finally we make the list L_d that consists of all deletions $\leq q$ stored in ancestors. We find the largest element in L_3 that is not in L_d using the following lemma.

▶ **Lemma 5.** *Let X and Y be two sets such that $|X| = 2m$, $|Y| = m$, and $Y \subset X$. We can find the largest element $e \in X$, such that $e \notin Y$ in $O(m/B)$ I/Os.*

Proof. Let $e(X, r)$ and $e(Y, r)$ denote the r-th largest elements in X and Y respectively. Let $X(r_1, r_2)$ be the set of elements $e(X, r_1)$, ..., $e(X, r_2)$. The cost of finding $e(X, r)$ and $e(Y, r)$ for any r is $O(\frac{m}{B})$ using linear-time selection.

If $e(X, k) = e(Y, k)$, then the k largest elements in X and Y are identical. If $e(X, k) > e(Y, k)$, then at least one among the k largest elements of X does not occur in Y. We must find the smallest k', such that $e(X, k') > e(Y, k')$. Clearly $e(X, k')$ is the required element. If $e(X, m) = e(Y, m)$, then $k' = m + 1$. Otherwise $e(X, m) > e(Y, m)$ and we proceed as follows. We compare $e(X, m/2)$ and $e(Y, m/2)$. If $e(X, m/2) = e(Y, m/2)$, we recursively search for k' among elements of $X(m/2, m)$ and $Y(m/2, m)$. If $e(X, m/2) > e(Y, m/2)$, we recursively search for k' among the elements of $X(1, m/2 - 1)$ and $Y(1, m/2 - 1)$. The total runtime is $O(m/B)$ I/Os. ◀

Lists L_1, L_2, L_3, and L_d can be constructed in $O(\log_B N)$ I/Os. We can find the largest insertion x in L_3 that is not in L_d in $O(|L_3|/B) = O(\log_B N)$: first we extract from L_d all deletions that do not exceed q and then we apply Lemma 5. Clearly, x is the largest element that is not deleted and is not larger than q. Hence x is the predecessor of q and can be found in $O(\log_B N)$ I/Os.

Summing up, we can answer predecessor and membership queries in $O(\log_B N)$ I/Os. The cost of answering a reporting query is $O(\log_B N + k/B)$ where k is the number of reported values. For reporting queries (but not for membership and predecessor queries), we need to make an assumption that the internal memory can hold at least $\log_B N$ blocks. However the same assumption is also necessary in the case of standard B^ε-trees with amortized updates.

3.6 Rebuilding

In our description we assumed that the values of $\log_B N$ and τ are fixed. However, if the size of the data structure changes significantly (e.g., the number of updates in the data structure changes from N_0 to below $\sqrt{N_0}$), the value of τ can change by more than a constant factor. Hence the invariant of leaf size, that is the $\Theta(B \log_B^2 N)$ bound on leaf size can be violated.

In order to maintain the invariant of leaf size, we must re-build the tree in certain situations. Let N_0 denote the number of updates in the tree when it was re-built for the last time. If $N \geq (1/2)N_0^2$ or $N \leq (3/2)\sqrt{N_0}$, we set $\tau' = B \log_B^2 N$ and construct a new tree, such that all leaves hold between $(1/4)\tau'$ and $(7/4)\tau'$ updates. The new tree T^{new} can be constructed in the background as explained below.

Let T^{old} denote the currently used tree and let T^{new} denote the new constructed tree. We traverse the updates stored in the leaf nodes of T^{old} in left-to-right order. Updates stored in each leaf are examined in increasing order (this is easy to do because the updates in micro-leaves and in the root node buffer of $T(\ell)$ are sorted). Let ℓ denote the currently visited leaf node and let L denote the list of insertions stored in the ancestors of ℓ. During each round we examine the next $2B^{1-\delta}$ insertions stored in ℓ. We add these insertions to L and extract the set S of $B^{1-\delta}$ smallest insertions. Next, we visit all ancestors of ℓ and remove from S all deletions (that is, if S contains an insertion e and an ancestor of ℓ stores a deletion e, then e is removed from S). We also remove all deletions stored in the root buffer of $T(\ell)$. All insertions that remain in S are added into the new tree T^{new}. When all insertions in a leaf ℓ are examined, we move to the right neighbor of ℓ and update the list L accordingly.

When new updates are added to T^{old}, we also add them to T^{new}. The only exception are deletions that correspond to unprocessed insertions: Let $v_{\max}$ denote the key value of the largest insertion from T^{old} that was added to T^{new}. All deletions with key value $v \leq v_{\max}$ are added to T^{new}; deletions with key value $v > v_{\max}$ are not added to T^{new}. When all insertions in the leaves of T^{old} are examined and copied to T^{new}, we discard T^{old} and start using T^{new} to answer queries.

▶ Remark. To simplify the description, we set $\delta = \varepsilon/2$ in our data structure. Hence the height of our tree is two times larger than the height of the standard B^{ε}-tree. It is possible to use any constant $\delta < \varepsilon$ such that $B^{\delta} \log B < B^{\varepsilon}$. Thus it is possible to reduce the height ratio of our tree and the standard B^{ε}-tree to any constant $\rho > 1$.

References

1 Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.

2 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.

3 Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. Cache-oblivious priority queue and graph algorithm applications. *SIAM Journal on Computing*, 36(6):1672–1695, 2007.

4 Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. An optimal cache-oblivious priority queue and its application to graph algorithms. *SIAM Journal on Computing*, 36(6):1672–1695, 2007. doi:10.1137/S0097539703428324.

5 Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, February 1972. doi:10.1145/1734663.1734671.

6 Michael A Bender, Rathish Das, Martín Farach-Colton, Rob Johnson, and William Kuszmaul. Flushing without cascades. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 650–669. SIAM, 2020.

7 Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming B-trees. In *Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 81–92, 2007.

8 Michael A. Bender, Martin Farach-Colton, Rob Johnson, Simon Mauras, Tyler Mayer, Cynthia Phillips, and Helen Xu. Write-optimized skip lists. In *Proc. 36th ACM Symposium on Principles of Database Systems (PODS)*, pages 69–78, May 2017.

9 Gerth Stølting Brodal, Erik D. Demaine, Jeremy T. Fineman, John Iacono, Stefan Langerman, and J. Ian Munro. Cache-oblivious dynamic dictionaries with update/query tradeoffs. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1448–1456. SIAM, 2010. doi:10.1137/1.9781611973075.117.

10 Gerth Stølting Brodal and Rolf Fagerberg. Funnel heap – A cache oblivious priority queue. In *Proc. 13th International Symposium on Algorithms and Computation (ISAAC)*, pages 219–228, 2002.

11 Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 546–554, 2003.

12 Gerth Stølting Brodal, Rolf Fagerberg, Ulrich Meyer, and Norbert Zeh. Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory – SWAT 2004*, pages 480–492, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

13 Adam L Buchsbaum, Michael H Goldwasser, Suresh Venkatasubramanian, and Jeffery R Westbrook. On external memory graph traversal. In *SODA*, pages 859–860, 2000.

14 Rezaul A. Chowdhury and Vijaya Ramachandran. Cache-oblivious buffer heap and cache-efficient computation of shortest paths in graphs. *ACM Trans. Algorithms*, 14(1):1:1–1:33, January 2018. doi:10.1145/3147172.

15 Erik D. Demaine, John Iacono, and Stefan Langerman. Worst-case optimal tree layout in external memory. *Algorithmica*, 72(2):369–378, 2015. doi:10.1007/s00453-013-9856-2.

16 Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372, 1987. doi:10.1145/28395.28434.

17 R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. Heaps and heapsort on secondary storage. *Theor. Comput. Sci.*, 220(2):345–362, June 1999. doi:10.1016/S0304-3975(99)00006-7.

18 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

19 John Iacono, Riko Jacob, and Konstantinos Tsakalidis. External memory priority queues with decrease-key and applications to graph algorithms. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 60:1–60:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.60.

20 John Iacono, Ben Karsin, and Grigorios Koumoutsos. External memory planar point location with fast updates. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 58:1–58:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.58.

21 John Iacono and Mihai Pătraşcu. Using hashing to solve the dictionary problem (in external memory). In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete algorithms*, pages 570–582. SIAM, 2012.

22 Shunhua Jiang and Kasper Green Larsen. A faster external memory priority queue with decreasekeys. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1331–1343. SIAM, 2019.

23 Vijay Kumar and Eric J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP '96)*, SPDP '96, pages 169–, Washington, DC, USA, 1996. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=829517.830723.

24 J. Ian Munro and Yakov Nekrich. Dynamic planar point location in external memory. In *35th International Symposium on Computational Geometry (SoCG)*, pages 52:1–52:15, 2019. doi:10.4230/LIPIcs.SoCG.2019.52.

25 Patrick O'Neil, Edward Cheng, Dieter Gawlic, and Elizabeth O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996. doi:10.1007/s002360050048.

26 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1987.

27 Russell Sears, Mark Callaghan, and Eric Brewer. Rose: Compressed, log-structured replication. *Proceedings of the VLDB Endowment*, 1(1):526–537, 2008.

Finding Matching Cuts in H-Free Graphs

Felicia Lucke
Department of Informatics, University of Fribourg, Switzerland

Daniël Paulusma
Department of Computer Science, Durham University, UK

Bernard Ries
Department of Informatics, University of Fribourg, Switzerland

Abstract

The well-known NP-complete problem Matching Cut is to decide if a graph has a matching that is also an edge cut of the graph. We prove new complexity results for Matching Cut restricted to H-free graphs, that is, graphs that do not contain some fixed graph H as an induced subgraph. We also prove new complexity results for two recently studied variants of Matching Cut, on H-free graphs. The first variant requires that the matching cut must be extendable to a perfect matching of the graph. The second variant requires the matching cut to be a perfect matching. In particular, we prove that there exists a small constant $r > 0$ such that the first variant is NP-complete for P_r-free graphs. This addresses a question of Bouquet and Picouleau (arXiv, 2020). For all three problems, we give state-of-the-art summaries of their computational complexity for H-free graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases matching cut, perfect matching, H-free graph, computational complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.22

1 Introduction

Cut sets and connectivity are central topics in algorithmic graph theory. We consider edge cuts in graphs that have some additional structure. The common property of these cuts is that the edges in them must form a matching. Formally, consider a connected graph $G = (V, E)$. A set $M \subseteq E$ is a *matching* if no two edges in M have a common end-vertex. A set $M \subseteq E$ is an *edge cut*, if V can be partitioned into sets B and R such that M consists of all the edges with one end-vertex in B and the other one in R. Now, M is a *matching cut* if M is a matching that is also an edge cut; see also Figure 1. Matching cuts are well studied due to their applications in number theory [16], graph drawing [27], graph homomorphisms [15], edge labelings [1] and ILFI networks [12]. The corresponding decision problem, which asks whether a given connected graph has a matching cut, is known as Matching Cut.

We also consider two natural variants of Matching Cut. First, let G be a connected graph that has a *perfect* matching M, that is, every vertex of G is incident to an edge of M. If M contains a matching cut M' of G, then M is a *disconnected perfect matching* of G; see again Figure 1 for an example. The problem Disconnected Perfect Matching is to decide if a graph has a disconnected perfect matching. Every yes-instance of Disconnected Perfect Matching is a yes-instance of Matching Cut, but the reverse might not be true; for example, the 3-vertex path has a matching cut but no (disconnected) perfect matching.

Suppose now that we search for a matching cut with a maximum number of edges, or for a disconnected perfect matching with a matching cut that is as large as possible. In both settings, the extreme case is when the matching cut is a perfect matching itself. Such a matching cut is called *perfect*; see Figure 1. By definition, a perfect matching cut is a disconnected perfect matching, but the reverse might not hold: take the cycle on six vertices which has several disconnected perfect matchings but no perfect matching cut. The problem Perfect Matching Cut is to decide if a connected graph has a perfect matching cut.

Figure 1 The graph P_6 with a matching cut that is not contained in a disconnected perfect matching (left), a matching cut that is properly contained in a disconnected perfect matching (middle) and a perfect matching cut (right). In each figure, thick edges denote matching cut edges.

All three problems are known to be NP-complete, as we will explain in more detail below. Hence, it is natural to restrict the input to some special graph class to obtain a better understanding of the computational hardness of some problem, or some set of problems. In particular, jumps in complexity can be large and unexpected. To give an extreme example [25], there exist problems that are PSPACE-complete in general but constant-time solvable for every other *hereditary* graph class, i.e., that is closed under vertex deletion.

It is readily seen that a graph class is hereditary if and only if it can be characterized by a set of forbidden induced subgraphs. A well-known example of a family of hereditary graph classes is obtained when we forbid a single subgraph H. That is, a graph G is *H-free* if G does not contain H as induced subgraph, or equivalently, if G cannot be modified into H by a sequence of vertex deletions. Many classical graph problems and graph parameters have been studied for classes of H-free graphs, as can not only be seen from surveys for e.g. Colouring [14, 28] or clique-width [10], but also from extensive studies on H-free graphs for specific graphs H, such as bull-free graphs [7] or claw-free graphs [8, 18]. We will also focus on H-free graphs in this paper. Before presenting our results we first discuss relevant known results.

1.1 Known Results

Out of the three problems, Matching Cut has been studied most extensively. Already in the eighties, Chvátal [9] proved that Matching Cut is NP-complete. Afterwards a large number of complexity results were proven for special graph classes. Here, we only discuss those results that are relevant for our context, whereas results for non-hereditary graph classes can, for example, be found in [3, 21]. In particular, we refer to a recent paper of Chen et al. [6] for a comprehensive overview.

On the positive side, Bonsma [2] proved that Matching Cut is polynomial-time solvable for $K_{1,3}$-free graphs and P_4-free graphs. Recently, Feghali [13] proved the same for P_5-free graphs, which we extended to P_6-free graphs in [24]. In the latter paper, we also showed that if Matching Cut is polynomial-time solvable for H-free graphs, for some graph H, then it is so for $(H + P_3)$-free graphs (see Section 2 for any unexplained notation and terminology).

On the negative side, Matching Cut is NP-complete even for $K_{1,4}$-free graphs. This follows from the construction of Chvátal [9] (see also [2, 20]). Bonsma [2] proved that Matching Cut is NP-complete for planar graphs of girth 5, and thus for C_r-free graphs with $r \in \{3, 4\}$. Le and Randerath [22] proved that Matching Cut is NP-complete for $K_{1,5}$-free bipartite graphs. Hence, it is NP-complete for H-free graphs if H has an odd cycle. Via a trick of Moshi [26], NP-completeness for H-free graphs also holds if H has an even cycle (see [24]). Feghali [13] proved the existence of an unspecified constant r such that Matching Cut is NP-complete for P_r-free graphs; we will show that $r = 27$ in his construction.

We now turn to Disconnected Perfect Matching. This problem was introduced by Bouquet and Picouleau [4], under a different name, but to avoid confusion with Perfect Matching Cut, Le and Telle [23] introduced the notion of disconnected perfect matchings, which we adapted. As observed in [4], it follows from a result of Diwan [11] that every planar cubic bridgeless graph, except the K_4, has a disconnected perfect matching. Bouquet and

Picouleau [4] proved that Disconnected Perfect Matching is, among others, polynomial-time solvable for claw-free graphs and P_5-free graphs, but NP-complete for bipartite graphs (of diameter 4), for $K_{1,4}$-free planar graphs (each vertex of which has either degree 3 or 4) and for planar graphs with girth 5.

Finally, we discuss Perfect Matching Cut. Heggernes and Telle [17] proved that this problem is NP-complete. Le and Telle [23] proved that for every integer $g \geq 3$, Perfect Matching Cut is NP-complete even for $K_{1,4}$-free bipartite graphs of girth g. The same authors showed that the problem is polynomial-time solvable for the class of $S_{1,2,2}$-free graphs (which contain the classes of $K_{1,3}$-free graphs and P_5-free graphs) and for chordal graphs. As explained in [23], the latter result generalizes a known result for interval graphs, for which a branch decomposition of constant mim-width can be computed in polynomial time.

1.2 New Results

For Matching Cut on H-free graphs, the remaining cases are when H is a P_{27}-free forest, each vertex of which has degree at most 3, such that H is not an induced subgraph of P_6+sP_3 or $K_{1,3}+sP_3$ for some constant $s \geq 0$. By modifying the construction of Feghali [13], we prove in Section 3 that Matching Cut is NP-complete for $(4P_5, P_{19})$-free graphs. Using the aforementioned trick of Moshi [26], we also observe that Matching Cut is NP-complete for H^*-free graphs, where H^* is the 6-vertex graph that looks like the letter "H".

For Disconnected Perfect Matching on H-free graphs, the remaining cases are when H contains an even cycle of length at least 6, such that every vertex of H has degree at most 3 and H is not an induced subgraph of $K_{1,3}$ or P_5. Bouquet and Picouleau [4] asked about the complexity of the problem for P_r-free graphs, with $r \geq 6$. We partially answer their question by proving NP-completeness for $(4P_7, P_{23})$-free graphs in Section 3 (via modifying our construction for Matching Cut for $(4P_5, P_{19})$-free graphs).

For Perfect Matching Cut on H-free graphs, the remaining cases are when H is a forest of maximum degree 3, such that H is not an induced subgraph of $S_{1,2,2}$. In Section 4, we first prove that Perfect Matching Cut is polynomial-time solvable for graphs of radius at most 2, and we use this result to obtain a polynomial-time algorithm for P_6-free graphs. We also prove that if Perfect Matching Cut is polynomial-time solvable for H-free graphs, for some graph H, then it is so for $(H+P_4)$-free graphs. All our results are obtained by combining a number of known propagation rules [21, 23] with new rules that we will introduce. After applying these rules exhaustively, we obtain a graph, parts of which have been allocated to the sides B and R of the edge cut that we are looking for. We will prove that the connected components of the remaining subgraph will be placed completely in B or R, and that this property suffices. By doing so, we extend a known approach with our new rules and show that in this way we widen its applicability.

The following three theorems present the state-of-art for H-free graphs. They are obtained by combining the aforementioned results from [2, 4, 9, 22, 23, 24, 26] with our new results. We write $G' \subseteq_i G$ to indicate that G' is an induced subgraph of G; as mentioned, recall that all undefined notation can be found in Section 2.

▶ **Theorem 1.** *For a graph H, Matching Cut on H-free graphs is*
- *polynomial-time solvable if $H \subseteq_i sP_3 + K_{1,3}$ or $sP_3 + P_6$ for some $s \geq 0$, and*
- *NP-complete if $H \supseteq_i C_r$ for some $r \geq 3$, $K_{1,4}$, P_{19}, $4P_5$ or H^*.*

▶ **Theorem 2.** *For a graph H, Disconnected Perfect Matching on H-free graphs is*
- *polynomial-time solvable if $H \subseteq_i K_{1,3}$ or P_5, and*
- *NP-complete if $H \supseteq_i C_r$ for some odd $r \geq 3$, C_4, $K_{1,4}$, P_{23} or $4P_7$.*

▶ **Theorem 3.** *For a graph H,* Perfect Matching Cut *on H-free graphs is*
- *polynomial-time solvable if $H \subseteq_i sP_4 + S_{1,2,2}$ or $sP_4 + P_6$, for some $s \geq 0$, and*
- NP*-complete if $H \supseteq_i C_r$ for some $r \geq 3$ or $K_{1,4}$.*

We state a number of open problems that originate from our systematic study in Section 5.

2 Preliminaries

We only consider finite undirected graphs without multiple edges and self-loops. Throughout this section, we let $G = (V, E)$ be a connected graph. Let $u \in V$. The set $N(u) = \{v \in V \mid uv \in E\}$ is the *neighbourhood* of u in G, where $|N(u)|$ is the *degree* of u. A graph F is a *spanning* subgraph of G if $V(F) = V(G)$ and $E(F) \subseteq E(G)$. Let $S \subseteq V$. The *neighbourhood* of S is the set $N(S) = \bigcup_{u \in S} N(u) \setminus S$. The graph $G[S]$ is the subgraph of G *induced* by $S \subseteq V$, that is, $G[S]$ is the graph obtained from G after deleting the vertices not in S. We write $G' \subseteq_i G$ if G' is an induced subgraph of G. We say that S is a *dominating* set of G, and that $G[S]$ *dominates* G, if every vertex of $V \setminus S$ has at least one neighbour in S. The *domination number* of G is the size of a smallest dominating set of G.

Let $u, v \in V$. The *distance* between u and v in G is the *length* (number of edges) of a shortest path between u and v in G. The *eccentricity* of u is the maximum distance between u and any other vertex of G. The *diameter* of G is the maximum eccentricity over all vertices of G. The *radius* of G is the minimum eccentricity over all vertices of G. If G is not a tree, then the *girth* of G is the length of a shortest cycle in G.

Let H be a graph. Recall that G is H-free if G does not contain H as an induced subgraph. Let $\{H_1, \ldots, H_n\}$ be a set of graphs. Then G is $(H_1, \ldots, H_n)$*-free*, if G is H_i-free for every $i \in \{1, \ldots, n\}$. The graph P_r is the path on r vertices. The graph C_r is the cycle on r vertices. A bipartite graph with non-empty partition classes V_1 and V_2 is *complete* if there is an edge between every vertex of V_1 and every vertex of V_2. If $|V_1| = k$ and $|V_2| = \ell$, we write $K_{k,\ell}$. The graph $K_{1,\ell}$ is the *star* on $\ell + 1$ vertices. The graph $K_{1,3}$ is also known as the *claw*. For $1 \leq h \leq i \leq j$, the graph $S_{h,i,j}$ is the tree with one vertex of degree 3, whose (three) leaves are at distance h, i and j from the vertex of degree 3. Observe that $S_{1,1,1} = K_{1,3}$. We need the following known result (which has been strengthened in [5]).

▶ **Theorem 4** ([29])**.** *A graph G is P_6-free if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating (not necessarily induced) complete bipartite graph. Moreover, such a dominating subgraph of G can be found in polynomial time.*

Let G_1 and G_2 be two vertex disjoint graphs. The graph $G_1 + G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$ is the *disjoint union* of G_1 and G_2. For a graph G, the graph sG is the disjoint union of s copies of G. Let H^* be the "H"-graph, which is the graph on six vertices obtained from the $2P_3$ by adding an edge joining the middle vertices of the two P_3s.

A *red-blue colouring* of G colours every vertex of G either red or blue. If every vertex of a set $S \subseteq V$ has the same colour (red or blue), then S (and also $G[S]$) are called *monochromatic*. A red-blue colouring is *valid*, if every blue vertex has at most one red neighbour; every red vertex has at most one blue neighbour; and both colours red and blue are used at least once. If a red vertex u has a blue vertex neighbour v, then u and v are *matched*. See also Figure 1.

For a valid red-blue colouring of G, we let R be the *red* set consisting of all vertices coloured red and B be the *blue* set consisting of all vertices coloured blue (so $V = R \cup B$). Moreover, the *red interface* is the set $R' \subseteq R$ consisting of all vertices in R with a (unique) blue neighbour, and the *blue interface* is the set $B' \subseteq B$ consisting of all vertices in B with a (unique) red neighbour in R. A red-blue colouring of G is *perfect*, if it is valid and moreover

$R' = R$ and $B' = B$. A red-blue colouring of a graph G is *perfect-extendable*, if it is valid and $G[R \setminus R']$ and $G[B \setminus B']$ both contain a perfect matching. In other words, the matching given by the valid red-blue colouring can be extended to a perfect matching in G or, equivalently, is contained in a perfect matching in G.

We can now make the following observation, which is easy to see (the notion of red-blue colourings has been used before; see, for example, [13, 24]).

▶ **Observation 5.** *Let G be a connected graph. The following three statements hold:*

(i) *G has a matching cut if and only if G has a valid red-blue colouring;*

(ii) *G has a disconnected perfect matching if and only if G has a perfect-extendable red-blue colouring;*

(iii) *G has a perfect matching cut if and only if G has a perfect red-blue colouring.*

3 NP-Completeness Results

We prove three NP-completeness results in this section. Our first result is a straightforward observation. Recall that H^* is the six-vertex graph that looks like the letter "H". Let uv be an edge in a graph G. Replacing uv by new vertices w_1 and w_2 and edges uw_1, uw_2, vw_1, vw_2 is a *$K_{2,2}$-replacement*. Let G_{uv} be the new graph. Moshi [26] observed that G has a matching cut if and only if G_{uv} has a matching cut. Applying a $K_{2,2}$-replacement on every edge to ensure that no two degree-3 vertices are adjacent anymore leads to the following:

▶ **Theorem 6.** Matching Cut *is* NP*-complete for H^*-free graphs.*

Recall that Feghali [13] showed the existence of an integer r such that Matching Cut is NP-complete for P_r-free graphs. It can be shown that the gadget in Feghali's construction has an induced P_{26}, but no induced P_{27}, so one can take $r = 27$ but not $r \leq 26$. Our next result improves this value of r to $r = 19$. The proof of our result is based on Feghali's construction [13] after making some minor modifications to it. We omit the details. It can be verified that the gadget in the proof of Theorem 7 is not $(P_{18} + sP_4)$-free for any $s \geq 1$.

▶ **Theorem 7.** Matching Cut *is* NP*-complete for $(4P_5, P_{19})$-free graphs.*

We can modify the construction in the proof of Theorem 7 to obtain the following result for Disconnected Perfect Matching, which addresses a question of Bouquet and Picouleau [4]. Again, we omit the proof details.

▶ **Theorem 8.** Disconnected Perfect Matching *is* NP*-complete for $(4P_7, P_{23})$-free graphs.*

4 Polynomial Results

In Section 4.5 we show that Perfect Matching Cut is polynomial-time solvable for graphs of radius at most 2, for P_6-free graphs and for $(H + P_4)$-free graphs should Perfect Matching Cut be polynomial-time solvable for H-free graphs. The proofs of these results are all based on a common approach. This approach is described in Sections 4.1–4.4, but we give an outline of it below.

Outline. We prove the above results via a common approach. First, in Section 4.1, we deal with the case where the input graph G has a small dominating set. This case can be dealt with by using brute force. Now suppose that we find a dominating set D of G that is not small. In Section 4.1, we also show that we can test in polynomial time whether G has a perfect red-blue colouring in which D is monochromatic.

Due to the above it remains to check if G has a perfect red-blue colouring in which the dominating set D that we found is not monochromatic. We branch by essentially guessing an edge whose end-vertices are coloured with different colours. We then exhaustively apply, in Section 4.2, a number of rules due to which more vertices will be coloured either red or blue. So this leads to a partial red-blue colouring of G. We prove that the rules are safe, that is, each of the coloured vertices received their correct colour (assuming G has a perfect red-blue colouring that coincides with our original guess). We then show that the connected components of the subgraph of G induced by the uncoloured vertices must be monochromatic in every perfect red-blue colouring extension of the partial red-blue colouring of G. This allows us to apply a number of new rules given in Section 4.3 that exploit this property. Afterwards, more vertices will be coloured, and we show in Section 4.4 that we can check in polynomial time (by a reduction to 2-SAT) whether the partial red-blue colouring can be extended to a perfect red-blue colouring of the whole graph G.

4.1 Small or Monochromatic Dominating Sets

We start with two lemmas, whose proofs we omit; they are similar to the proofs for valid but not necessarily perfect red-blue colourings; see, for example, [13] or [24].

▶ **Lemma 9.** *For every integer g, it is possible to find in $O(2^g n^{g+2})$-time a perfect red-blue colouring (if it exists) of a graph with n vertices and with domination number g.*

▶ **Lemma 10.** *Let D be a dominating set of a connected graph G. It is possible to check in polynomial time if G has a perfect red-blue colouring in which D is monochromatic.*

4.2 Partial Red-Blue Colourings: Applying General Rules

To handle "partial" red-blue colourings that we want to extend to perfect red-blue colourings, we introduce the following terminology. Let $G = (V, E)$ be a connected graph and $S, T, X, Y \subseteq V$ be four non-empty sets with $S \subseteq X$, $T \subseteq Y$ and $X \cap Y = \emptyset$. A *red-blue* (S, T, X, Y)*-colouring* of G is a red-blue colouring where

- every vertex of X is coloured red and every vertex of Y is coloured blue;
- the blue neighbour of every vertex in S belongs to T and vice versa; and
- the blue neighbour of every vertex in $X \setminus S$ and the red neighbour of every vertex of Y belong to $V \setminus (X \cup Y)$.

For a connected graph $G = (V, E)$, let S' and T' be two disjoint subsets of V, such that (i) every vertex of S' is adjacent to at most one vertex of T', and vice versa, and (ii) at least one vertex in S' is adjacent to a vertex in T'. Let S'' consist of all vertices of S' with a (unique) neighbour in T', and let T'' consist of all vertices of T' with a (unique) neighbour in S' (so, every vertex in S'' has a unique neighbour in T'', and vice versa). We call (S'', T'') the *core* of *starting pair* (S', T'); note that $|S''| = |T''| \geq 1$.

We colour every vertex in S' red and every vertex in T' blue. Propagation rules will try to extend S' and T' by finding new vertices whose colour must always be either red or blue. We place new red vertices in a set X and new blue vertices in a set Y. If a red and blue vertex are matched to each other, then we add the red one to a set $S \subseteq X$ and the blue one to a set $T \subseteq Y$. Initially, $S := S''$, $T := T''$, $X := S'$ and $Y := T'$, and we let $Z := V \setminus (X \cup Y)$.

We now present seven propagation rules for finding perfect red-blue (S, T, X, Y)-colourings. Rules R1 and R2 hold for finding red-blue colourings in general and correspond to the five rules from [21]. Rules R3-R7 are for finding perfect red-blue colourings; some of them are in a slightly different form in [23].

R1. Return `no` (i.e., G has no red-blue (S, T, X, Y)-colouring) if a vertex $v \in Z$ is
- **(i)** adjacent to a vertex in S and to a vertex in T, or
- **(ii)** adjacent to a vertex in S and to two vertices in $Y \setminus T$, or
- **(iii)** adjacent to a vertex in T and to two vertices in $X \setminus S$, or
- **(iv)** adjacent to two vertices in $X \setminus S$ and to two vertices in $Y \setminus T$.

R2. Let $v \in Z$.
- **(i)** If v is adjacent to a vertex in S or to two vertices of $X \setminus S$, then move v from Z to X. If moreover v is also adjacent to a vertex w in Y, then add v to S and w to T.
- **(ii)** If v is adjacent to a vertex in T or to two vertices of $Y \setminus T$, then move v from Z to Y. If moreover v is also adjacent to a vertex w in X, then add v to T and w to S.

R3. Let $v \in (X \cup Y) \setminus (S \cup T)$.
- **(i)** If $v \in X \setminus S$ and v is adjacent to a vertex w in Y, then add v to S and w to T.
- **(ii)** If $v \in Y \setminus T$ and v is adjacent to a vertex w in X, then add v to T and w to S.

R4. Return `no` if
- **(i)** a vertex $x \in X$ has no neighbours outside X or is adjacent to two vertices of Y, or
- **(ii)** a vertex $y \in Y$ has no neighbours outside Y, or is adjacent to two vertices of X.

R5. Let $v \in Z$, and let $w \in Z$ be a vertex with $N_G(w) = \{v\}$.
- **(i)** If v is adjacent to a vertex in X and to a vertex in Y, then return `no`.
- **(ii)** If v is adjacent to a vertex in X but not to a vertex in Y, then put v in X and w in Y, and also add v to S and w to T.
- **(iii)** If v is adjacent to a vertex in Y but not to a vertex in X, then put v in Y and w in X, and also add v to T and w to S.

R6. Let $v \in Z$ be in a connected component F of $G[Z]$ such that F contains C_4 as a spanning subgraph.
- **(i)** If v is adjacent to a vertex in X but not to a vertex in Y, and F contains a vertex not adjacent to a vertex in X, then move v from Z to X.
- **(ii)** If v is adjacent to a vertex in Y but not to a vertex in X, and F contains a vertex not adjacent to a vertex in Y, then move v from Z to Y.

R7. Let $v \in Z$ be in a connected component F of $G[Z]$ such that $\{v\}$ dominates F. Let $F - v$ have a vertex w with only one neighbour w' in $X \cup Y$.
- **(i)** If $w' \in X$, then put v in Y.
- **(ii)** If $w' \in Y$, then put v in X.

A propagation rule is *safe* if the input graph has a perfect red-blue (S, T, X, Y)-colouring before the application of the rule if and only if it has so after the application of the rule. We omit the proof of the next lemma, in which we show that Rules R1–R7 are safe.

▶ **Lemma 11.** *Rules R1–R7 are safe.*

Assume that exhaustively applying rules R1–R7 on a starting pair (S', T') did not lead to a no-answer but to a 4-tuple (S, T, X, Y). Then we call (S, T, X, Y) an *intermediate* 4-tuple. The first part of the next lemma follows from Lemma 11. The second part is straightforward.

▶ **Lemma 12.** *Let G be a graph with a starting pair (S', T') with core (S'', T'') and a resulting intermediate 4-tuple (S, T, X, Y). Then G has a perfect red-blue (S'', T'', S', T')-colouring if and only if G has a perfect red-blue (S, T, X, Y)-colouring. Moreover, (S, T, X, Y) can be obtained in polynomial time.*

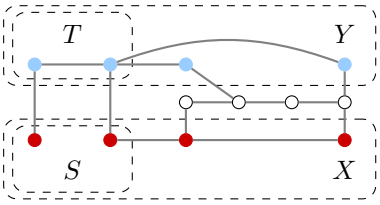


Figure 2 A red-blue (S,T,X,Y)-colouring of a graph with an intermediate 4-tuple (S,T,X,Y). In this example, $G[Z]$ consists of a single connected component isomorphic to P_4.

In the next lemma (proof omitted) we describe the structure of a graph with an intermediate 4-tuple (S,T,X,Y); see Figure 2 for an example.

▶ **Lemma 13.** *Let G be a graph with an intermediate 4-tuple (S,T,X,Y). Then:*

- **(i)** *every vertex in S has exactly one neighbour in Y and that neighbour belongs to T;*
- **(ii)** *every vertex in T has exactly one neighbour in X and that neighbour belongs to S;*
- **(iii)** *every vertex in $X \setminus S$ has no neighbour in Y;*
- **(iv)** *every vertex in $Y \setminus T$ has no neighbour in X;*
- **(v)** *every vertex in $V \setminus (X \cup Y)$ has no neighbour in $S \cup T$, at most one neighbour in $X \setminus S$ and at most one neighbour in $Y \setminus T$.*

4.3 Partial Red-Blue Colourings: Exploiting Monochromaticity

Let (S,T,X,Y) be an intermediate 4-tuple of a graph G. Let $Z = V \setminus (X \cup Y)$. A red-blue (S,T,X,Y)-colouring of G is *monochromatic* if all connected components of $G[Z]$ are monochromatic. Rules R8-R11 preserve this property; some of them were also used in [21, 23].

R8. Let $v \in Z$. If v is not adjacent to any vertex of $X \cup Y$, then return `no`.

R9. Let $v \in Z$ be a vertex in a connected component F of $G[Z]$ such that v has only one neighbour w in $X \cup Y$.
- **(i)** If $w \in X$, then put every vertex of F in Y and also add every vertex of F to T and every neighbour of every vertex of F in X to S.
- **(ii)** If $w \in Y$, then put every vertex of F in X and also add every vertex of F to S and every neighbour of every vertex of F in Y to T.

R10. Let $v \in (X \cup Y) \setminus (S \cup T)$ and F be a connected component of $G[Z]$ such that v has two neighbours in F.
- **(i)** If $v \in X \setminus S$, then put every vertex of F in X, and also add every vertex of F to S and every vertex of every neighbour of F in $Y \setminus T$ to T.
- **(ii)** If $v \in Y \setminus T$, then put every vertex of F in Y, and also add every vertex of F to T and every vertex of every neighbour of F in $X \setminus S$ to S.

R11. Let $v \in (X \cup Y) \setminus (S \cup T)$ and F be a connected component of $G[Z]$ such that v has one neighbour in F that is the only neighbour of v in Z.
- **(i)** If $v \in X \setminus S$ and v is not adjacent to Y, then put every vertex of F in Y, and also add every vertex of F to T and every vertex of every neighbour of F in $X \setminus S$ to S.
- **(ii)** If $v \in Y \setminus T$ and v is not adjacent to X, then put every vertex of F in X, and also add every vertex of F to S and every vertex of every neighbour of F in $Y \setminus T$ to T.

A propagation rule is *mono-safe* if the input graph has a (monochromatic) perfect red-blue (S,T,X,Y)-colouring before the application of the rule if and only if it has so after the application of the rule. Our next lemma is not difficult to prove and we omit its proof.

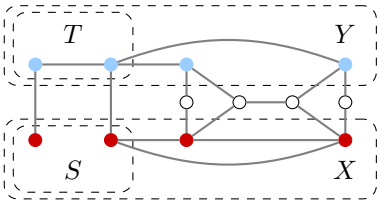


Figure 3 A red-blue (S,T,X,Y)-colouring of a graph with a final 4-tuple (S,T,X,Y). In this example, $G[Z]$ is isomorphic to $2P_1 + P_2$.

▶ **Lemma 14.** *Rules R8–R11 are mono-safe.*

Suppose exhaustively applying rules R1–R11 on an intermediate 4-tuple (S,T,X,Y) did not lead to a no-answer but to a 4-tuple (S^*,T^*,X^*,Y^*). We call (S^*,T^*,X^*,Y^*) the *final* 4-tuple. The first part of Lemma 15 follows from Lemma 14. The second part is straightforward.

▶ **Lemma 15.** *Let G be a graph with an intermediate 4-tuple (S,T,X,Y) and a resulting final 4-tuple (S^*,T^*,X^*,Y^*). Then G has a monochromatic perfect red-blue (S,T,X,Y)-colouring if and only if G has a monochromatic perfect red-blue (S^*,T^*,X^*,Y^*)-colouring. Moreover, (S^*,T^*,X^*,Y^*) can be obtained in polynomial time.*

In our next lemma (proof omitted) we describe the structure of a graph with a final 4-tuple (S,T,X,Y); see Figure 3 for an example.

▶ **Lemma 16.** *Let G be a graph with a final 4-tuple (S,T,X,Y). The following holds:*

- **(i)** *every vertex in S has exactly one neighbour in Y, which belongs to T;*
- **(ii)** *every vertex in T has exactly one neighbour in X, which belongs to S;*
- **(iii)** *every vertex in $X \setminus S$ has no neighbour in Y, at least two neighbours in $V \setminus (X \cup Y)$ but no two neighbours in the same connected component of $G[V \setminus (X \cup Y)]$;*
- **(iv)** *every vertex in $Y \setminus T$ has no neighbour in X, at least two neighbours in $V \setminus (X \cup Y)$ but no two neighbours in the same connected component of $G[V \setminus (X \cup Y)]$;*
- **(v)** *every vertex of $V \setminus (X \cup Y)$ has no neighbour in $S \cup T$, exactly one neighbour in $X \setminus S$ and exactly one neighbour in $Y \setminus T$.*

4.4 Reduction to 2-SAT

We now prove a lemma that is the cornerstone for our polynomial-time results.

▶ **Lemma 17.** *Let G be a graph with a final 4-tuple (S,T,X,Y). Then it is possible to find in polynomial time a monochromatic perfect red-blue (S,T,X,Y)-colouring of G or conclude that such a colouring does not exist.*

Proof. Let $Z = V \setminus (X \cup Y)$. Let $E^* \subseteq E$ be the set of edges consisting of all edges with one end-vertex in $(X \cup Y) \setminus (S \cup T)$ and the other end-vertex in Z. By Lemma 16-(v), we find that $|E^*| = 2|Z|$. By Lemma 16-(iii) and (iv), we find that $|E^*| \geq 2|(X \cup Y) \setminus (S \cup T)|$. Hence, $|Z| \geq |(X \cup Y) \setminus (S \cup T)|$, and $|Z| = |(X \cup Y) \setminus (S \cup T)|$ if and only if each vertex in $(X \cup Y) \setminus (S \cup T)$ has exactly two neighbours in Z.

Every vertex $u \in Z$ still needs their matching neighbour v. In order for G to have a monochromatic perfect red-blue (S,T,X,Y)-colouring, v must be outside $S \cup T$, so v belongs to $X \cup Y$. By Lemma 16-(v), we find that $v \in (X \cup Y) \setminus (S \cup T)$. As matching neighbours are "private", $|Z| \leq |(X \cup Y) \setminus (S \cup T)|$. We conclude that $|(X \cup Y) \setminus (S \cup T)| = |Z|$. Our algorithm checks this in polynomial time and returns a no-answer if $|(X \cup Y) \setminus (S \cup T)| \neq |Z|$.

From now on, assume $|(X \cup Y) \setminus (S \cup T)| = |Z|$. Hence, each vertex in $(X \cup Y) \setminus (S \cup T)$ has exactly two neighbours in Z. Just like [21], we now construct an instance ϕ of the 2-Satisfiability problem (2-SAT). Our 2-SAT formula differs from the one in [21] due to the perfectness requirement. For each connected component C of $G[Z]$, we do as follows. We define two variables x_C and y_C, and we add the clause $(x_C \vee y_C) \wedge (\neg x_C \vee \neg y_C)$ to ϕ. For each $u \in (X \cup Y) \setminus (S \cup T)$, we do as follows. From the above we known that u has exactly two neighbours v and w in Z. Let C be the connected component of $G[Z]$ that contains v and D be the connected component of $G[Z]$ that contains w. We add the clause $(x_C \vee x_D) \wedge (y_C \vee y_D)$ to ϕ. This finishes the construction of ϕ.

We claim that G has a monochromatic perfect red-blue (S, T, X, Y)-colouring if and only if ϕ has a satisfying truth assignment. It is readily seen and well known that 2-SAT is polynomial-time solvable, meaning we are done once we have proven this claim.

First suppose that G has a monochromatic perfect red-blue (S, T, X, Y)-colouring c. By definition, the vertices in each connected component C of $G[Z]$ are coloured alike. We define a truth assignment τ as follows. We let x_C be true if and only if the vertices of C are coloured red. We let y_C be true if and only if the vertices of C are coloured blue. As exactly one of these options holds, the clause $(x_C \vee y_C) \wedge (\neg x_C \vee \neg y_C)$ is satisfied.

Now consider a clause $(x_C \vee x_D) \wedge (y_C \vee y_D)$ corresponding to a vertex $u \in (X \cup Y) \setminus (S \cup T)$ that has a neighbour in each of the connected components C and D of $G[Z]$. Then, by Lemma 16-(iii) and (iv), C and D are different connected components of $G[Z]$. First assume that $u \in X \setminus S$. By Lemma 16-(iii), we find that u has no neighbour in Y and thus its blue neighbour must either be in C or in D. If it is in C, then the neighbour of u in D is coloured blue, and vice versa. As c is monochromatic, this means that either all vertices of C are coloured red and all vertices of D are coloured blue, or the other way around. Hence, the clause $(x_C \vee x_D) \wedge (y_C \vee y_D)$ is satisfied. If $u \in Y \setminus T$, we can use exactly the same arguments. We conclude that τ is a satisfying truth assignment.

Now suppose that ϕ has a satisfying truth assignment τ. For every connected component C of $G[Z]$, we colour the vertices of C red if x_C is true and we colour the vertices of C blue if y_C is true. As τ satisfies $(x_C \vee y_C) \wedge (\neg x_C \vee \neg y_C)$, exactly one of x_C or y_C is true. Hence, the colouring of the vertices of Z is well defined.

We also colour all vertices of X red and all vertices of Y blue. We let c be the resulting colouring. By construction, it is monochromatic. Hence, it remains to show that c is a perfect red-blue (S, T, X, Y)-colouring. We will do this below.

First, it follows from the definition of a core (S'', T'') that S'' and T'' are non-empty. Moreover, before applying the reduction rules, we first do an initiation, from which it follows that $S'' \subseteq S$ and $T'' \subseteq T$. Hence, at least one vertex of G is coloured red and at least one vertex of G is coloured blue.

By Lemma 16-(i), every vertex in S has exactly one neighbour in Y. By Lemma 16-(ii), every vertex in T has exactly one neighbour in X. By Lemma 16-(v), no vertex of $S \cup T$ is adjacent to a vertex of Z. Hence, the vertices in $S \cup T$ have exactly one neighbour of opposite colour.

By Lemma 16-(v), every vertex $z \in Z$ has exactly one neighbour in $X \setminus S$, which is coloured red, and exactly one neighbour in $Y \setminus T$, which is coloured blue; moreover, z is not adjacent to any vertex in $S \cup T$. Let C be the connected component of $G[Z]$ that contains z. As c is monochromatic, all vertices of C receive the same colour. Hence, the vertices in Z have each exactly one neighbour of opposite colour.

Finally, we must verify the vertices in $(X \cup Y) \setminus (S \cup T)$. Let $u \in (X \cup Y) \setminus (S \cup T)$. First assume that $u \in X \setminus S$, so u is coloured red. We recall that u has exactly two neighbours v and w in Z. Let C be the connected component of $G[Z]$ that contains u, and

let D be the connected component of $G[Z]$ that contains w. Hence, τ contains the clause $(x_C \vee x_D) \wedge (y_C \vee y_D)$. By Lemma 16-(iii), we find that C and D are two distinct connected components of $G[Z]$. As τ satisfies $(x_C \vee x_D) \wedge (y_C \vee y_D)$, the vertices of one of C, D are coloured red, while the vertices of the other one are coloured blue. By Lemma 16-(iii), we find that u has no (blue) neighbour in Y. Hence, u has exactly one blue neighbour. If $u \in Y \setminus T$, we can apply the same arguments. We conclude that also the vertices in $(X \cup Y) \setminus (S \cup T)$ have exactly one neighbour of the opposite colour.

From the above we conclude that c is monochromatic and perfect. ◀

4.5 Applications of Our Approach

We first apply the approach described in the previous subsections to graphs of radius at most 2. Our proof is similar but more involved than the one for Matching Cut on graphs of radius 2 [24].

▶ **Theorem 18.** Perfect Matching Cut *is polynomial-time solvable for graphs of radius at most* 2.

Proof. Let G be a graph of radius r at most 2. If $r = 1$, then G has a vertex that is adjacent to all other vertices. In this case G has a perfect matching cut if and only if G consists of two vertices with an edge between them. From now on, assume that $r = 2$. Then G has a dominating star H, say H has centre u and leaves $v_1, \ldots, v_s$ for some $s \geq 1$. By Observation 5 it suffices to check if G has a perfect red-blue colouring.

We first check if G has a perfect red-blue colouring in which $V(H)$ is monochromatic. By Lemma 10 this can be done in polynomial time. Suppose we find no such red-blue colouring. Then we may assume without loss of generality that a perfect red-blue colouring of G (if it exists) colours u red and exactly one of $v_1, \ldots, v_s$ blue. That is, G has a perfect red-blue colouring if and only if G has a perfect red-blue $(\{u\}, \{v_i\}, \{u\}, \{v_i\})$-colouring for some $i \in \{1, \ldots, s\}$. We consider all $O(n)$ options of choosing which v_i is coloured blue.

For each option we do as follows. Let v_i be the vertex of $v_1, \ldots, v_s$ that we coloured blue. We define the starting pair (S', T') with core (S', T'), where $S' = \{u\}$ and $T' = \{v_i\}$. We now apply rules R1–R7 exhaustively. The latter takes polynomial time by Lemma 12. If this exhaustive application leads to a no-answer, then by Lemma 12 we may discard the option. Suppose we obtain an intermediate 4-tuple (S, T, X, Y). By again applying Lemma 12, we find that G has a perfect red-blue $(\{u\}, \{v_i\}, \{u\}, \{v_i\})$-colouring if and only if G has a perfect red-blue (S, T, X, Y)-colouring. By R2-(i) and the fact that $u \in S' \subseteq S$ we find that $\{v_1, \ldots, v_s\} \setminus \{v_i\}$ belongs to X.

Suppose that G has a perfect red-blue (S, T, X, Y)-colouring c such that $G[V(G) \setminus (X \cup Y)]$ has a connected component D that is not monochromatic. Then D must contain an edge uv, where u is coloured red and v is coloured blue. Note that v cannot be adjacent to v_i, as otherwise v would have been in Y by R3 (since $v_i \in T' \subseteq T$). As H is dominating, this means that v must be adjacent to a vertex $w \in V(H) \setminus \{v_i\} = \{u, v_1, \ldots, v_s\} \setminus \{v_i\}$. As $u \in S' \subseteq S \subseteq X$ and $\{v_1, \ldots, v_s\} \setminus \{v_i\} \subseteq X$, we find that $w \in X$ by R2-(i) and thus will be coloured red. However, now v being coloured blue is adjacent to two red vertices (namely u and w), contradicting the validity of c.

From the above we conclude that every perfect red-blue (S, T, X, Y)-colouring of G is monochromatic. We now apply rules R1–R11 exhaustively. The latter takes polynomial time by Lemma 15. If this exhaustive application leads to a no-answer, then by Lemma 15 we may discard the option. Suppose we obtain a final 4-tuple (S^*, T^*, X^*, Y^*). By again applying Lemma 15, we find that G has a monochromatic perfect red-blue (S, T, X, Y)-colouring if

and only if G has a monochromatic perfect red-blue (S^*, T^*, X^*, Y^*)-colouring. We can now apply Lemma 17 to find in polynomial time whether or not G has a monochromatic perfect red-blue (S^*, T^*, X^*, Y^*)-colouring. The correctness of our algorithm follows from the above arguments. As we branch $O(n)$ times and each branch takes polynomial time to process, the total running time of our algorithm is polynomial. ◀

We now consider P_6-free graphs. As a consequence of Theorem 4, a P_6-free graph either has a small domination number, in which case we use Lemma 9, a monochromatic dominating set, in which case we use Lemma 10, or it has radius 2, in which case we use Theorem 18.

▶ **Theorem 19.** Perfect Matching Cut *is polynomial-time solvable for P_6-free graphs.*

Proof. Let G be a connected P_6-free graph. By Theorem 4, we find that G has a dominating induced C_6 or a dominating (not necessarily induced) complete bipartite graph $K_{r,s}$. By Observation 5 it suffices to check if G has a perfect red-blue colouring.

If G has a dominating induced C_6, then G has domination number at most 6. In that case we apply Lemma 9 to find in polynomial time if G has a perfect red-blue colouring. Suppose that G has a dominating complete bipartite graph D with partition classes $\{u_1, \ldots, u_r\}$ and $\{v_1, \ldots, v_s\}$. We may assume without loss of generality that $r \leq s$.

If $r \geq 2$ and $s \geq 3$, then any starting pair $(\{u_i\}, \{v_j\})$ yields a no-answer. Hence, $V(D)$ is monochromatic for any perfect red-blue colouring of G. This means that we can check in polynomial time by Lemma 10 if G has a perfect red-blue colouring.

Now assume that $r = 1$ or $s \leq 2$. In the first case, G has a (not necessarily induced) dominating star and thus G has radius 2, and we apply Theorem 18. In the second case, $r \leq s \leq 2$, and thus G has domination number at most 4, and we apply Lemma 9. Hence, in both cases, we find in polynomial time whether or not G has a perfect red-blue colouring. ◀

For our last result we again apply our approach.

▶ **Theorem 20.** *Let H be a graph. If* Perfect Matching Cut *is polynomial-time solvable for H-free graphs, then it is so for $(H + P_4)$-free graphs.*

Proof. Assume that Perfect Matching Cut can be solved in polynomial time for H-free graphs. Let G be a connected $(H + P_4)$-free graph. Say, G has an induced subgraph G' that is isomorphic to H; else we are done by our assumption. Let G^* be the graph obtained from G after removing every vertex that belongs to G' or that has a neighbour in G'. As G' is isomorphic to H and G is $(H + P_4)$-free, G^* is P_4-free. See Figure 4 for an example of this decomposition of G, where we have chosen $H = S_{1,2,2}$.

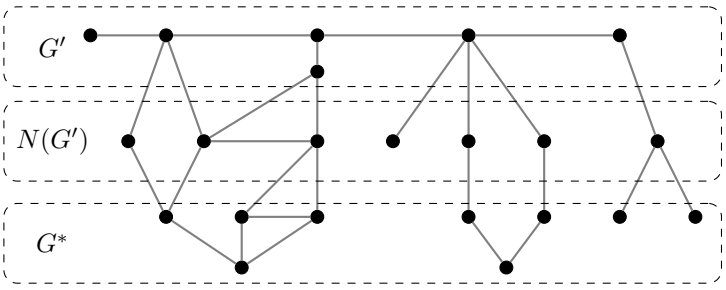


Figure 4 The decomposition of a graph G into the graphs G', which is isomorphic to H, the neighbourhood graph $N(G')$ of G', which is induced by all vertices not in G' but that have one or more neighbours in G', and the graph G^* induced by the remaining vertices of G.

We use Observation 5-(iii) and search for a perfect red-blue colouring. We define $n = |V(G)|$ and $m = |E(G)|$. Following our approach, we need a starting pair (S', T') with core (S'', T''). By definition, $|S''| = |T''| \geq 1$. Hence, we consider all $O(m)$ options of choosing an edge uv from $E(G)$, one of whose end-vertices we colour red, say u (so $u \in S''$) and the other one, v, blue (so $v \in T''$). Afterwards, for each (uncoloured) vertex in G' we consider all options of colouring it either red or blue. As G' is isomorphic to H, the number of distinct options is a constant, namely $2^{|V(H)|}$. Now, for every red (blue) vertex of G' with no blue (red) neighbour, we consider all $O(n)$ options of colouring exactly one of its neighbours blue (red). Hence, afterwards each vertex of $V(G') \cup N(V(G'))$ is either coloured red or blue. This leads to $O(m2^{|V(H)|}n^{|V(H)|})$ options (branches), which we handle one by one.

Consider an option as described above. Let S' consist of u and all red vertices of $V(G') \cup N(V(G'))$, and let T' consist of v and all blue vertices of $V(G') \cup N(V(G'))$. In this way we obtain a starting pair (S', T') with core (S'', T''). We apply rules R1-R7 exhaustively. If we find a no-answer, then we can discard the option by Lemma 12. Else we found in polynomial time an intermediate 4-tuple (S, T, X, Y), such that G has a perfect red-blue (S'', T'', S', T')-colouring if and only if G has a perfect red-blue (S, T, X, Y)-colouring.

Consider a connected component F of $G - (X \cup Y)$, for which the following holds:

1. F contains two adjacent vertices u and v, each with no neighbours in $X \cup Y$ and moreover, v is dominating F; and
2. every vertex in $F - \{u, v\}$ has a neighbour in both X and Y.

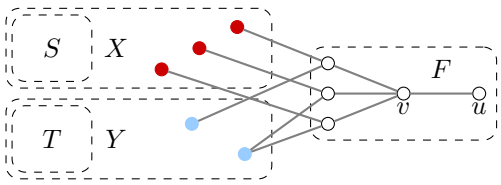


Figure 5 An example of a reducible connected component F of a graph G with an intermediate tuple (S, T, X, Y). For readability only edges with at least one end-vertex in F are drawn. Note that $F - \{u, v\}$ consists of three single-vertex components and that $F - \{u, v\}$ becomes a K_3 after our algorithm has processed F.

We say that F is a *reducible* connected component of $G - (X \cup Y)$, as after processing F in the way described below we either found that G has no perfect red-blue (S, T, X, Y)-colouring, or we have reduced the size of G. See Figure 5 for an example.

As G is connected, the fact that u and v have no neighbours in $X \cup Y$ implies that $F - \{u, v\}$ is non-empty. As every vertex in $F - \{u, v\}$ has a neighbour in both X and Y, their matching neighbour is not in F in every perfect red-blue (S, T, X, Y)-colouring of G. As v dominates F, all vertices of $F - \{u, v\}$ are adjacent to v. Hence, we find that in every perfect red-blue (S, T, X, Y)-colouring of G, all vertices of $F - \{u, v\}$ have the same colour as v, that is, all vertices of $F - \{u\}$ are coloured alike. If u is adjacent to a vertex in $F - \{u, v\}$, this means that u will receive the same colour as every other vertex in F including v and hence, u will not have a matching neighbour. So, in this case, G has no perfect red-blue (S, T, X, Y)-colouring.

Now suppose that u is not adjacent to any vertex of $F - \{u, v\}$. Then u is only adjacent to v in G. We now remove u and v and we add any missing edge between two vertices of $F - \{u, v\}$ such that in the end $F - \{u, v\}$ has become a complete graph.

The above operation is safe to do, as in any perfect red-blue (S, T, X, Y)-colouring of the new graph (if it exists) the vertices of $F - \{u, v\}$ will all be coloured alike. This is because the matching neighbour of every vertex of $F - \{u, v\}$ belongs to $X \cup Y$ and we

have modified $F - \{u, v\}$ into a complete graph. We can now give v the same colour as the vertices of $F - \{u, v\}$ and u the opposite colour. In this way we obtain a perfect red-blue (S, T, X, Y)-colouring of G. Similarly, as we argued above, a perfect red-blue (S, T, X, Y)-colouring of G gives all the vertices of $F - \{u\}$ the same colour and makes u the matching neighbour of v. Hence, such a colouring (if it exists) will correspond to a perfect red-blue (S, T, X, Y)-colouring of the new graph.

We will also process any other reducible connected components of $G-(X \cup Y)$ in the same way. Then either we found that the original graph G has no perfect red-blue (S, T, X, Y)-colouring and we discard the option, or we found a new graph that has a perfect red-blue (S, T, X, Y)-colouring if and only if G has a perfect red-blue (S, T, X, Y)-colouring. Assume that we are in the latter situation. We continue with the new graph and denote it by G again (note that the new graph is the same graph as G if G had no reducible connected components). We now prove the following claim.

$\triangleright$ Claim. Every connected component of $G - (X \cup Y)$ in every perfect red-blue (S, T, X, Y)-colouring of G is monochromatic.

Proof. In order to see this claim, let F be a connected component of $G - (X \cup Y)$. If F corresponds to some reducible connected component in the original graph then, as we argued above, F will be monochromatic in any perfect red-blue (S, T, X, Y)-colouring of G.

Now suppose that F was not obtained from some reducible connected component. By construction, F is not reducible. If $|V(F)| = 1$, then F will be monochromatic. Assume $|V(F)| \geq 2$. As $V(G') \cup N(V(G')) \subseteq S' \cup T'$ and $S' \subseteq X$ and $T' \subseteq Y$, we find that $V(F)$ belongs to G^*. Since G^* is P_4-free, F is P_4-free. It is well-known (see e.g. Lemma 2 in [19]) that every connected P_4-free graph has a spanning complete bipartite subgraph K. Say, K is isomorphic to $K_{k,\ell}$ for some integers $1 \leq k \leq \ell$.

If $k \geq 2$ and $\ell \geq 3$, then F must be monochromatic. Now suppose that $k = \ell = 2$, so F contains a C_4 as a spanning subgraph. If K contains a vertex u that has a neighbour in both X and Y, then the matching neighbour of u will be in $X \cup Y$, so not in F. Hence, the neighbours of u in F must receive the same colour as u. The latter means that the fourth vertex of F must also receive the same colour as u (if that vertex is not adjacent to u, then it will be adjacent to the two neighbours of u in F, as F contains a spanning C_4). So F is monochromatic.

We conclude that every vertex of F is adjacent to at most one vertex of $X \cup Y$. As G is connected, F has at least one vertex v with a neighbour w in $X \cup Y$, say $w \in X$. Then the other three vertices of F must also have a neighbour in X (and thus no neighbour in Y), else we would have applied R6. The only way we can extend the red-blue (S, T, X, Y)-colouring to a perfect red-blue colouring of G is by colouring each vertex of F blue, so F is monochromatic.

It remains to consider the case where $k = 1$ and $\ell \geq 1$. In this case F contains a vertex v such that $\{v\}$ dominates F. Then every vertex in $F - v$ has either no neighbours in $X \cup Y$ or a neighbour in both X and Y; else we would have applied R7. Let U be the set of vertices in $F - v$ with no neighbour in $X \cup Y$. As $\{v\}$ dominates F, every connected component of $F - v$ is monochromatic. So, v is the matching neighbour of every vertex of U. Hence, if $|U| \geq 2$, then G has no perfect red-blue (S, T, X, Y)-colouring so the claim is true. If $|U| = 0$, then the vertices in $F - v$ all have a neighbour both in X and Y. So, they do not have their matching neighbour in F and thus will receive the same colour as v. Hence, F is monochromatic. Assume that $|U| = 1$, say $U = \{u\}$ for some vertex u of G. As v is the matching neighbour of u, we find that v is adjacent to at most one vertex of $X \cup Y$.

We now have that F contains two adjacent vertices u and v, where u has no neighbours in $X \cup Y$ and moreover, v is dominating F, and every vertex in $F - \{u, v\}$ has a neighbour in both X and Y. Recall that F is not reducible. Hence, v is adjacent to exactly one vertex w

of $X \cup Y$. Then u has at least one neighbour in $F - v$; else we would have applied R5. Let u' be an arbitrary neighbour of u in $F - v$. As both u and u' are adjacent to v, it follows that u, u', v are coloured alike. Hence, u has no matching neighbour. This means that G has no perfect red-blue (S, T, X, Y)-colouring and the claim is true. ◁

We now apply rules R1–R11 exhaustively. This takes polynomial time by Lemma 15. If this leads to a no-answer, then by Lemma 15 we may discard the option. Suppose we obtain a final 4-tuple (S^*, T^*, X^*, Y^*). By Lemma 15, G has a monochromatic perfect red-blue (S, T, X, Y)-colouring if and only if G has a monochromatic perfect red-blue (S^*, T^*, X^*, Y^*)-colouring. We apply Lemma 17 to find in polynomial time if the latter holds. If so, we are done by the Claim, else we discard the option.

The correctness of our algorithm follows from its description. As the total number of branches is $O(m2^{|V(H)|}n^{|V(H)|})$ and we can process each branch in polynomial time, the total running time of our algorithm is polynomial. Hence, we have proven the theorem. ◀

5 Conclusions

We found new results on H-free graphs for three closely related edge cut problems: the classical Matching Cut problem and its variants, Disconnected Perfect Matching and Perfect Matching Cut. We summarized all known and new results for H-free graphs in Theorems 1–3. We finish our paper with two open questions.

First, as can be noticed from Theorems 1–3, our knowledge on the complexity of the three problems is different. In particular, does there exist a constant r such that Perfect Matching Cut is NP-complete for P_r-free graphs? For the other two problems such a constant exists. For Matching Cut we improved the previous value $r = 27$ [13] to $r = 19$ and for Disconnected Perfect Matching we showed that we can take $r = 23$, addressing a question in [4]. We expect that these values of r might not be tight, but it does not seem straightforward to improve our current hardness constructions.

Second, is there a graph H for which the problems behave differently on H-free graphs? The graph $H = 4P_5$ is a candidate graph should it be possible to generalize Theorem 20 from $(H + P_4)$-free graphs to $(H + P_5)$-free graphs.

References

1 Júlio Araújo, Nathann Cohen, Frédéric Giroire, and Frédéric Havet. Good edge-labelling of graphs. *Discrete Applied Mathematics*, 160:2502–2513, 2012.

2 Paul S. Bonsma. The complexity of the Matching-Cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62:109–126, 2009.

3 Mieczyslaw Borowiecki and Katarzyna Jesse-Józefczyk. Matching cutsets in graphs of diameter 2. *Theoretical Computer Science*, 407:574–582, 2008.

4 Valentin Bouquet and Christophe Picouleau. The complexity of the Perfect Matching-Cut problem. *CoRR*, abs/2011.03318, 2020. `arXiv:2011.03318`.

5 Eglantine Camby and Oliver Schaudt. A new characterization of P_k-free graphs. *Algorithmica*, 75:205–217, 2016.

6 Chi-Yeh Chen, Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. Matching Cut in graphs with large minimum degree. *Algorithmica*, 83:1238–1255, 2021.

7 Maria Chudnovsky. The structure of bull-free graphs II and III - A summary. *Journal of Combinatorial Theory, Series B*, 102:252–282, 2012.

8 Maria Chudnovsky and Paul Seymour. The structure of claw-free graphs. *Surveys in Combinatorics, London Mathematical Society Lecture Note Series*, 327:153–171, 2005.

9 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.
10 Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Clique-width for hereditary graph classes. *Proc. BCC 2019, London Mathematical Society Lecture Note Series*, 456:1–56, 2019.
11 Ajit A. Diwan. Disconnected 2-factors in planar cubic bridgeless graphs. *Journal of Combinatorial Theory, Series B*, 84:249–259, 2002.
12 Arthur M. Farley and Andrzej Proskurowski. Networks immune to isolated line failures. *Networks*, 12:393–403, 1982.
13 Carl Feghali. A note on Matching-Cut in P_t-free graphs. *Information Processing Letters*, 179:106294, 2023.
14 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of colouring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84:331–363, 2017.
15 Petr A. Golovach, Daniël Paulusma, and Jian Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theoretical Computer Science*, 457:86–100, 2012.
16 Ronald L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York Academy of Sciences*, 175:170–186, 1970.
17 Pinar Heggernes and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5:128–142, 1998.
18 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. *ACM Transactions on Algorithms*, 15:25:1–25:90, 2019.
19 Walter Kern and Daniël Paulusma. Contracting to a longest path in H-free graphs. *Proc. ISAAC 2020, LIPIcs*, 181:22:1–22:18, 2020.
20 Dieter Kratsch and Van Bang Le. Algorithms solving the Matching Cut problem. *Theoretical Computer Science*, 609:328–335, 2016.
21 Hoàng-Oanh Le and Van Bang Le. A complexity dichotomy for Matching Cut in (bipartite) graphs of fixed diameter. *Theoretical Computer Science*, 770:69–78, 2019.
22 Van Bang Le and Bert Randerath. On stable cutsets in line graphs. *Theoretical Computer Science*, 301:463–475, 2003.
23 Van Bang Le and Jan Arne Telle. The Perfect Matching Cut problem revisited. *Proc. WG 2021, LNCS*, 12911:182–194, 2021.
24 Felicia Lucke, Daniël Paulusma, and Bernard Ries. On the complexity of Matching Cut for graphs of bounded radius and H-free graphs. *Theoretical Computer Science*, 2022.
25 Barnaby Martin, Daniël Paulusma, and Siani Smith. Hard problems that quickly become very easy. *Information Processing Letters*, 174:106213, 2022.
26 Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13:527–536, 1989.
27 Maurizio Patrignani and Maurizio Pizzonia. The complexity of the Matching-Cut problem. *Proc. WG 2001, LNCS*, 2204:284–295, 2001.
28 Bert Randerath and Ingo Schiermeyer. Vertex colouring and forbidden subgraphs - A survey. *Graphs and Combinatorics*, 20:1–40, 2004.
29 Pim van 't Hof and Daniël Paulusma. A new characterization of P_6-free graphs. *Discrete Applied Mathematics*, 158:731–740, 2010.

Graph Product Structure for h-Framed Graphs

Michael A. Bekos
University of Ioannina, Greece

Giordano Da Lozzo
Roma Tre University, Rome, Italy

Petr Hliněný
Masaryk University, Brno, Czech Republic

Michael Kaufmann
Universität Tübingen, Germany

Abstract

Graph product structure theory expresses certain graphs as subgraphs of the strong product of much simpler graphs. In particular, an elegant formulation for the corresponding structural theorems involves the strong product of a path and of a bounded treewidth graph, and allows to lift combinatorial results for bounded treewidth graphs to graph classes for which the product structure holds, such as to planar graphs [Dujmović et al., J. ACM, 67(4), 22:1-38, 2020].

In this paper, we join the search for extensions of this powerful tool beyond planarity by considering the h-framed graphs, a graph class that includes 1-planar, optimal 2-planar, and k-map graphs (for appropriate values of h). We establish a graph product structure theorem for h-framed graphs stating that the graphs in this class are subgraphs of the strong product of a path, of a planar graph of treewidth at most 3, and of a clique of size $3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1$. This allows us to improve over the previous structural theorems for 1-planar and k-map graphs. Our results constitute significant progress over the previous bounds on the queue number, non-repetitive chromatic number, and p-centered chromatic number of these graph classes, e.g., we lower the currently best upper bound on the queue number of 1-planar graphs and k-map graphs from 115 to 82 and from $\lfloor \frac{33}{2}(k + 3\lfloor \frac{k}{2} \rfloor - 3) \rfloor$ to $\lfloor \frac{33}{2}(3\lfloor \frac{k}{2} \rfloor + \lfloor \frac{k}{3} \rfloor - 1) \rfloor$, respectively. We also employ the product structure machinery to improve the current upper bounds on the twin-width of 1-planar graphs from $O(1)$ to 80. All our structural results are constructive and yield efficient algorithms to obtain the corresponding decompositions.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Graphs and surfaces

Keywords and phrases Graph product structure theory, h-framed graphs, k-map graphs, queue number, twin-width

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.23

Related Version Complete proofs of statements marked with **(*)** in the main part can be found in the extended version of this document:
Extended Version: `https://arxiv.org/abs/2204.11495` [6]

Funding *Giordano Da Lozzo*: Supported in part by MIUR Project "AHeAD" under PRIN 20174LF3T8.
Petr Hliněný: Supported by the Czech Science Foundation, project no. 20-04567S.
Michael Kaufmann: Supported in part by DFG Ka 812-18/2.

Acknowledgements This research started at the Dagstuhl Seminar 21293 "Parameterized Complexity in Graph Drawing", July 18 – 23, 2021 [20].

1 Introduction

Graph product structure theory [15] was recently introduced and is receiving considerable attention, as it gives deep insights that allow a host of mathematical and algorithmic tools to be applied. Despite being a relatively new development, it is having significant impact [25]. Initially, it was introduced to settle a long-standing conjecture by Heath, Leighton and Rosenberg [21] related to the queue number of planar graphs [15]. Recently, it has been further exploited to solve several other combinatorial problems that were open for years, e.g., it was used to prove that planar graphs have bounded non-repetitive chromatic number [15], to improve the best known bounds for p-centered colorings of planar graphs and graphs excluding any fixed graph as a subdivision [11], to find shorter adjacency labelings of planar graphs [7], and to find asymptotically optimal adjacency labelings of planar graphs [13].

In its simplest form, the product structure theorem states that every planar graph is a subgraph of the strong product of a path and of a planar graph of treewidth at most 6 [15, 27]. The bound on the treewidth can be improved by allowing more than two graphs in the strong product, as it is known that every planar graph is a subgraph of the strong product of a path, of a 3-cycle and of a planar graph of treewidth at most 3 [15]. These theorems are attractive, since they describe planar graphs in terms of graphs of bounded treewidth, which are considered much simpler than the planar ones. Furthermore, they enable combinatorial results that hold for graphs of bounded treewidth to be generalised for planar graphs and, more generally, for graphs where similar structural theorems can be obtained. On the algorithmic side, it has been recently shown that the graphs involved in the product structure theorem can be computed in linear time [10], improving upon [24].

Analogous results are known for graphs of bounded Euler genus [15], apex-minor-free graphs [15], graphs with bounded degree in proper minor-closed classes [14], and graphs in non-minor closed classes [17]; see [19] for a survey. Related to our work are the structural theorems for k-planar and k-map graphs (the former ones are the graphs that can be drawn with at most k crossings per edge, whereas the latter ones are the contact-graphs of regions homeomorphic to closed disks such that at most k regions may share the same point). In particular, it is known that every k-planar graph is a subgraph of the strong product of a path, of a graph of treewidth at most $\frac{1}{6}(k+4)(k+3)(k+2)-1$, and of a clique on $18k^2+48k+30$ vertices, while every k-map graph is a subgraph of the strong product of a path, of a planar graph of treewidth at most 3, and of a clique on $k+3\lfloor\frac{k}{2}\rfloor-3$ vertices [17].

Our contribution. In this work, our focus is on the class of h-framed graphs, which were recently introduced as a notable subclass of k-planar and a superclass of k-map graphs (for appropriate values of k) [5]; a graph is *h-framed*, if it admits a drawing on the Euclidean plane whose uncrossed edges induce a biconnected spanning plane graph with faces of size at most h. Since any h-framed graph is $O(h^2)$-planar, it follows from the aforementioned product structure theorem for k-planar graphs that every h-framed graph is a subgraph of a path, of a graph with treewidth $O(h^6)$ and of a clique of size $O(h^4)$. Our main contribution is to show the following structural result (which guarantees that the graph of bounded treewidth is planar and of improved treewidth, from $O(h^6)$ to $O(1)$, and lowers the size of the involved clique to $O(h)$): Every h-framed graph is a subgraph of the strong product of a path, of a planar graph of treewidth at most 3, and of a clique on $3\lfloor\frac{h}{2}\rfloor+\lfloor\frac{h}{3}\rfloor-1$ vertices[1];

[1] Note that, almost concurrently with our result, a slightly weaker version of our product structure theorem for h-framed graphs appeared in [17], where the size of the involved clique is larger, namely, $h+3\lfloor\frac{h}{2}\rfloor-3$.

Table 1 Previous [17] and new bounds on the queue number, non-repetitive and p-centered chromatic number, and twin-width for h-framed, 1-planar, optimal 2-planar, and k-map graphs. We denote by $\chi_p(H)$ the p-centered chromatic number of planar 3-trees.

		h-framed/h-map	1-planar	opt 2-planar
queue num	old	$\lfloor \frac{33}{2}(h+3\lfloor\frac{h}{2}\rfloor-3)\rfloor$	115	132
	new	$\lfloor \frac{33}{2}(3\lfloor\frac{h}{2}\rfloor+\lfloor\frac{h}{3}\rfloor-1)\rfloor$	82	82
non-repetitive chr. num	old	$4^4\cdot(h+3\lfloor\frac{h}{2}\rfloor-3)$	1792	2048
	new	$4^4\cdot(3\lfloor\frac{h}{2}\rfloor+\lfloor\frac{h}{3}\rfloor-1)$	1536	1536
p-centered chr. num	old	$(h+3\lfloor\frac{h}{2}\rfloor-3)(p+1)\chi_p(H)$	$7(p+1)\chi_p(H)$	$8(p+1)\chi_p(H)$
	new	$(3\lfloor\frac{h}{2}\rfloor+\lfloor\frac{h}{3}\rfloor-1)(p+1)\chi_p(H)$	$6(p+1)\chi_p(H)$	$6(p+1)\chi_p(H)$
twin-width	old	–	$O(1)$	$O(1)$
	new	$17h+13$	80	80

see Theorem 3.1. Note that, since any planar graph is a subgraph of some triangulation (and thus of a 3-framed graph), we have that, for $h=3$, Theorem 3.1 coincides with the product structure theorem for planar graph proved in [15]. This is an indication that our theorem may be tight for the graphs in this family. Furthermore, we provide an alternative formulation, where the role played by a path is instead played by the $\lfloor h/2\rfloor$-th power of a path, which allows us to further reduce the size of the clique involved in the product to $\max(3, h-2)$; see Theorem 3.5. All our structural results are constructive and yield efficient algorithms to obtain the corresponding decompositions. Our techniques provide improved upper bounds on the queue number, on the non-repetitive chromatic number, and on the p-centered chromatic number of h-framed graphs that are linear in h; see Theorem 4.2, Corollary 4.7, and Lemma 4.8, respectively. Finally, by extending the product structure machinery, we are able to give an efficient construction to obtain an explicit, linear in h, upper bound on the twin-width of h-framed graphs, while the currently best explicit upper bound derives from the one for k-planar graphs and it is hence exponential in $O(h^2)$ [8, 9]; see Theorem 5.1.

Consequences on related graph classes. Since 1-planar and optimal 2-planar graphs are subgraphs of 4- and 5-framed graphs, respectively [2, 4], and since k-map graphs are subgraphs of k-framed graphs [3, 5], the product structure theorems mentioned above imply significant improvements on the currently best bounds for the following parameters (refer to Table 1). For definitions, see Section 4.

- **Queue number:** Using Theorem 3.1, we improve the best known upper bound on the queue number of k-map graphs from $\lfloor \frac{33}{2}(k+3\lfloor\frac{k}{2}\rfloor-3)\rfloor$ [17] to $\lfloor \frac{33}{2}(3\lfloor\frac{k}{2}\rfloor+\lfloor\frac{k}{3}\rfloor-1)\rfloor$ (Corollary 4.3), whereas, using Theorem 3.5, we lower the best known upper bounds on the queue number of 1-planar and optimal 2-planar graphs from 115 and 132 [17], respectively, both to 82 (Theorem 4.5).
- **Non-repetitive chromatic number:** Theorem 3.1 allows us to improve the best known upper bound on the non-repetitive chromatic number of k-map graphs from $4^4\cdot(k+3\lfloor\frac{k}{2}\rfloor-3)$ [17] to $4^4\cdot(3\lfloor\frac{k}{2}\rfloor+\lfloor\frac{k}{3}\rfloor-1)$. In particular, for the class of 1-planar graphs our improvement is from 1792 to 1536 (Corollary 4.7). The latter is a bound that notably holds also for optimal 2-planar graphs, which forms an improvement over the one of 2048 that was previously known [17].

For the sake of completeness, even though the results in [17] are still unpublished, we will compare the bounds stemming from our product structure theorem with those stemming from the one in [17].

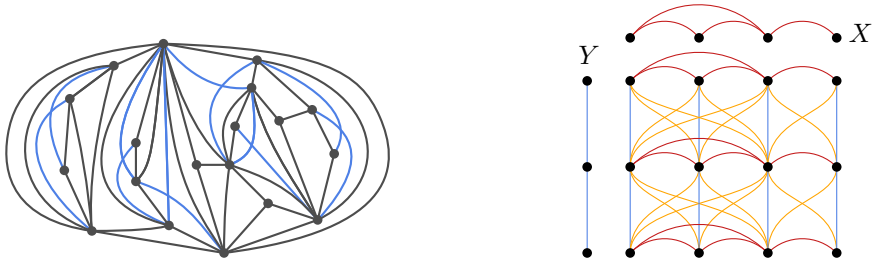


Figure 1 Illustration of: (a) a 4-framed topological graph whose skeleton edges (crossing edges) are black (blue), and (b) the strong product $X \boxtimes Y$ of a planar graph X (red) and a path Y (blue).

- ***p*-centered chromatic number:** Theorem 3.1 allows us to improve the best known upper bound on the non-repetitive chromatic number of k-map graphs from $(k+3\lfloor\frac{k}{2}\rfloor-3)(p+1)\chi_p(H)$ [17] to $(3\lfloor\frac{k}{2}\rfloor+\lfloor\frac{k}{3}\rfloor-1)(p+1)\chi_p(H)$, where $\chi_p(H) \leq (p+1)(p\lceil\log(p+1)+2p+1)$ denotes the p-centered chromatic number of planar 3-trees [11]. In particular, we lower the best known upper bounds on the p-centered chromatic number of 1-planar and optimal 2-planar graphs from $7(p+1)\chi_p(H)$ and $8(p+1)\chi_p(H)$ [17], respectively, both to $6(p+1)\chi_p(H)$ (Corollary 4.9).
- **Twin-width:** Theorem 5.1 improves the currently best upper bound on the twin-width of 1-planar and optimal 2-planar graphs, from $O(1)$ [8] to 80, whereas our improvement for k-map graphs is limited to certain value of k, as these graphs have bounded twin-width independently of k [8].

2 Preliminaries

For standard graph-theoretic terminology and notation we refer the reader, e.g., to [12].

Graphs. A graph is *simple* if it contains neither loops nor multi-edges. For a general graph G (not-necessarily simple), let $\mathrm{si}(G)$ denote the *simplification* of G, i.e., the simple graph obtained from G by removing all loops and replacing each bunch of parallel edges with a single edge. For any $i \geq 1$, the *i-th power* G^i of a graph G is the graph with the same vertex set as G, in which two vertices are adjacent if and only if they are at distance at most i in G. Clearly, $G \subseteq G^i$. A graph H is a *minor* of a graph G, if H can be obtained from a subgraph of G by contracting edges.

Topological graphs. A *topological graph* is a graph drawn on the plane such that any two edges cross in at most one point and no edge crosses itself. In this paper, we will solely consider topological graphs in which no two adjacent edges cross and no three edges cross in the same point (in the literature, such drawings are commonly referred to as "simple"). A *plane graph* is a topological graph with no crossing edges. A graph is *k-planar* if it is isomorphic to a topological graph in which each edge crosses at most k other edges. Furthermore, a k-planar graph with the maximum number of edges is called *optimal*. A *k-map graph* is one that admits a *k-map*, i.e., a representations where each vertex is a region homeomorphic to a closed disk, such that regions have pairwise disjoint interiors, no more than k regions share the same boundary point, and two regions touch if and only if the corresponding vertices are adjacent.

Given a topological graph G, the subgraph $\mathrm{sk}(G)$ of G consisting of all its vertices and uncrossed edges is the *skeleton* of G; refer to Fig. 1a. A topological graph G whose skeleton $\mathrm{sk}(G)$ is biconnected is called *h-framed* [5], if all the faces of $\mathrm{sk}(G)$ have size at most h, and *internally h-framed*, if all the faces of $\mathrm{sk}(G)$, except for possibly one, have size at most h. The importance of this class lies in the following connections with k-planar and k-map graphs [3, 5]. Optimal 1-planar and optimal 2-planar graphs are 4- and 5-framed, respectively, while general 1-planar graphs can be augmented to 8-framed graphs, if multi-edges are forbidden, or to 4-framed graphs, if multi-edges are allowed. Finally, note that any k-map graph is a subgraph of a k-framed (multi-)graph and of a $2k$-framed simple graph [3, 5].

Treewidth. Let $(\mathcal{X}, T)$ be a pair such that $\mathcal{X} = \{X_1, X_2, \ldots, X_\ell\}$ is a collection of subsets of vertices of a graph G, called *bags*, and T is a tree whose nodes are in one-to-one correspondence with the elements of $\mathcal{X}$. The pair $(\mathcal{X}, T)$ is a *tree-decomposition* of G if it satisfies the following two conditions: (i) for every edge (u, v) of G, there exists a bag $X_i \in \mathcal{X}$ that contains both u and v, and (ii) for every vertex v of G, the set of nodes of T whose bags contain v induces a non-empty subtree of T. The *width* of a tree-decomposition $(\mathcal{X}, T)$ of G is $\max_{i=1}^{\ell} |X_i| - 1$, while the *treewidth* $\mathrm{tw}(G)$ of G is the minimum width over all tree-decomposition of G.

Quotient graph. For a graph G and a partition $\mathcal{P}$ of $V(G)$, the *quotient of G by $\mathcal{P}$*, denoted by $G/\mathcal{P}$, is a graph containing a vertex v_P for each part P in $\mathcal{P}$ (we say that v_P *stems from* P) and an edge $(v_{P'}, v_{P''})$ if and only if there exists a vertex in P' adjacent to a vertex in P'' in G. Note that, $G/\mathcal{P}$ is a minor of G, if every part in $\mathcal{P}$ induces a connected subgraph of G.

Strong product. The *strong product* of two graphs X and Y, denoted by $X \boxtimes Y$, is the graph whose vertex set $V(X \boxtimes Y)$ is the Cartesian product $V(X) \times V(Y)$, such that there exists an edge in $E(X \boxtimes Y)$ between the vertices $\langle x_1, y_1\rangle, \langle x_2, y_2\rangle \in V(X \boxtimes Y)$ if and only if one of the following occurs: (a) $x_1 = x_2$ and $(y_1, y_2) \in E(Y)$, (b) $y_1 = y_2$ and $(x_1, x_2) \in E(X)$, or (c) $(x_1, x_2) \in E(X)$ and $(y_1, y_2) \in E(Y)$; see Fig. 1b. Dujmović et al. [15, 17] and Ueckerdt, Wood, and Yi [27] showed the following main graph product structure results.

▶ **Theorem 2.1** (Dujmović et al. [15, 17], Ueckerdt et al. [27]). *For a graph G, the next hold:*

(a) *If G is planar, then $G \subseteq P \boxtimes H$, for a path P and a planar graph H with $\mathrm{tw}(H) \leq 6$.*

(b) *If G is planar, then $G \subseteq P \boxtimes H \boxtimes K_3$, for a path P and a planar graph H with $\mathrm{tw}(H) \leq 3$.*

(c) *If G is 1-planar, then $G \subseteq P \boxtimes H \boxtimes K_7$, for a path P and a planar graph H with $\mathrm{tw}(H) \leq 3$.*

(d) *If G is k-planar with $k > 1$, then $G \subseteq P \boxtimes H \boxtimes K_{18k^2+48k+30}$, for a path P and a graph H with $\mathrm{tw}(H) \leq \frac{1}{6}(k+4)(k+3)(k+2) - 1$.*

(e) *If G is a k-map graph, then $G \subseteq P \boxtimes H \boxtimes K_{21k(k-3)}$, for a path P and a graph H with $\mathrm{tw}(H) \leq 9$.*

Layering. Consider a graph G. A *layering* of G is an ordered partition $(V_0, V_1, \ldots)$ of $V(G)$ such that, for every edge (v, w) of G with $v \in V_i$ and $w \in V_j$, it holds $|i - j| \leq 1$. If $i = j$, then (v, w) is an *intra-level edge*; otherwise, (v, w) is an *inter-level edge*. Each part V_i is called a *layer*. Let T be a BFS tree of G rooted at a vertex r. The *BFS layering* of G *determined by* r is the layering $(V_0, V_1, \ldots)$ of G such that V_i contains all vertices of G at distance i from r. Given a partition $\mathcal{P}$ of $V(G)$ and a layering $\mathcal{L}$ of G, the *layered width of $\mathcal{P}$ with respect to $\mathcal{L}$* is the size of the largest set obtained by intersecting a part in $\mathcal{P}$ and a layer in $\mathcal{L}$. The *layered width* of $\mathcal{P}$ is the minimum layered width of $\mathcal{P}$ over all layerings of G.

3 Computing the Product Structure

This section is devoted to the proof of a product structure theorem for h-framed graphs, summarized in the next theorem; several applications of this result are presented in Section 4.

▶ **Theorem 3.1** (Product Structure Theorem for h-Framed Graphs). *Let G be a not-necessarily simple h-framed graph with $h \geq 3$. Then, $\mathrm{si}(G)$ is a subgraph of the strong product $H \boxtimes P \boxtimes K_{3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1}$, where H is a planar graph with $\mathrm{tw}(H) \leq 3$ and P is a path.*

The algorithm supporting Theorem 3.1 is going to recursively decompose the graph G into parts with special properties, such that the resulting quotient graph will be H, and the additional properties of the constructed partition will imply the claimed product structure. We start with a technical setup followed by the core recursion in Lemma 3.2.

Layering G. Let T be a BFS tree of $\mathrm{sk}(G)$ rooted at an arbitrary vertex r incident to the unbounded face of $\mathrm{sk}(G)$. For an arbitrary H and its implicitly fixed BFS tree $T' \subseteq H$ (such as $H = \mathrm{sk}(G)$ and $T' = T$ in our case), we call a path $P \subseteq \mathrm{sk}(G)$ *vertical* if P is a subpath of some root-to-leaf path of T'. Let $\mathcal{L} = (V_0, V_1, \ldots, V_b)$ be the BFS layering of $\mathrm{sk}(G)$ determined by r. Observe that, if P is a vertical path in $\mathrm{sk}(G)$, then P intersects every part of $\mathcal{L}$ in at most one vertex. Given $\mathcal{L}$, we define a new ordered partition $\mathcal{W} = (W_0, W_1, \ldots, W_\ell)$ of the vertex set of G with $\ell = \lceil b / \lfloor \frac{h}{2} \rfloor \rceil - 1$, by merging consecutive $\lfloor \frac{h}{2} \rfloor$-tuples of layers of $\mathcal{L}$. This is done as follows. For $i = 0, 1, \ldots, \ell$, we let $W_i := \bigcup_{j=0}^{\lfloor h/2 \rfloor - 1} V_{i\lfloor h/2 \rfloor + j}$ (assuming $V_x = \emptyset$ if $x > b$). Then, $\mathcal{W} := (W_0, W_1, \ldots, W_\ell)$ is a layering of G; see [6].

Partitioning G. The core of our algorithm is a construction of a special partition $\mathcal{R}$ of $V(G)$ such that $H = G/\mathcal{R}$ is a planar graph with $\mathrm{tw}(H) \leq 3$, and the layered width of $\mathcal{R}$ with respect to $\mathcal{W}$ is not large. Our recursive decomposition of G is analogous to the one in [15] (as applied to planar graphs); however several non-trivial changes are needed to exploit the existence of the underlying (plane) skeleton of G. The algorithm starts from the unbounded face and recursively "dives" into gradually-shrinking areas of G.

Central in our approach is the following notion. For a cycle $C \subseteq \mathrm{sk}(G)$, the *subgraph of G bounded by C*, denoted by G_C, is the subgraph of G formed by the vertices and edges of C and the vertices and edges of G drawn inside C. Consider a subset $U \subseteq V(G)$. For the partition $\mathcal{L}$ (resp., the partition $\mathcal{W}$), the *width of U with respect to $\mathcal{L}$* (resp. *to $\mathcal{W}$*), denoted by $\lambda_{\mathcal{L}}(U)$ (resp. by $\lambda_{\mathcal{W}}(U)$), is the largest size of a set obtained by intersecting U and a part of $\mathcal{L}$ (resp. of $\mathcal{W}$). We are now ready to present our main technical lemma.

▶ **Lemma 3.2.** *Let G be an h-framed graph with $h \geq 3$ and let $\mathcal{L}$ be a BFS layering of G. Also, let C be a cycle in $\mathrm{sk}(G)$, and let G_C be the subgraph of G bounded by C. Further, for some $k \in \{1, 2, 3\}$, let $P_1, \ldots, P_k$ be paths belonging to C such that $\mathcal{R}^0 = \{X_i : X_i := V(P_i), 1 \leq i \leq k\}$ is a partition of $V(C)$. Then, it is possible to construct in quadratic time a* good *partition $\mathcal{R}'$ of $V(G_C)$, i.e., one that satisfies the following properties:*

1. *$\mathcal{R}' \supseteq \mathcal{R}^0$, and for every part $X \in \mathcal{R}' \setminus \mathcal{R}^0$, there exist $q \in \{1, 2, 3\}$ and $X' \subseteq X$ such that*[2]
   - *$X \setminus X'$ is a union of the vertex sets of at most q vertical paths of $\mathrm{sk}(G)$, and so, in particular, $\lambda_{\mathcal{L}}(X \setminus X') \leq q$, and*
   - *$|X'| \leq h - 3$ if $q = 1$, $|X'| \leq \lfloor (h-1)/2 \rfloor - 1$ if $q = 2$, and $|X'| \leq \lfloor h/3 \rfloor - 1$ if $q = 3$.*

[2] The somehow technical Property 1 of Lemma 3.2 will imply that $\lambda_{\mathcal{W}}(X) \leq 3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1$ in the proof of Theorem 3.1, but we will also make use of the stated more detailed treatment.

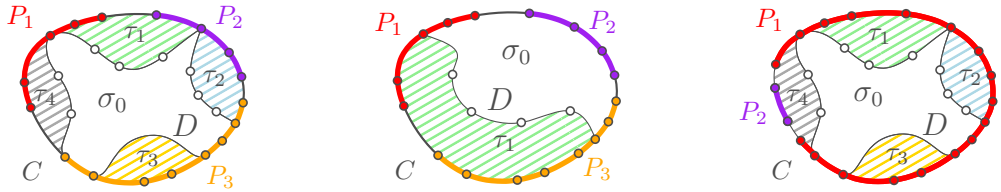


Figure 2 Illustrations of graph $C \cup D$ for the separable case of G_C (Definition 3.3). The number a of τ-faces is 4 in (a) and 1 in (b). (c) A case when Property 2 of Definition 3.3 is not met (by τ_4).

2. *the quotient graph* $H' = G_C/\mathcal{R}'$ *is a planar graph with* $\mathrm{tw}(H') \leq 3$, *and*
3. *the vertices of* H' *that stem from* X_i, *with* $1 \leq i \leq k$, *are incident to the same face of* H' *and induce a clique (i.e., either a vertex, or an edge, or a triangle).*

Proof. We prove Lemma 3.2 by providing a recursive procedure that we describe in the following. The base case of the recursion occurs when $V(G_C) = V(C)$ (i.e., there are no vertices in the interior of C and the edges in $E(G_C) \setminus E(C)$ are chords of C). In this case, the algorithm returns the partition $\mathcal{R}' = \mathcal{R}^0$, which is clearly good since the graph H' is a plane clique of size k whose vertices stem from the parts of $\mathcal{R}^0$. Note that, if $|E(G_C) \setminus E(C)| = 0$, then $V(G_C) = V(C)$, since G_C cannot have isolated vertices.

In the recursive step of the algorithm, we assume that there exist vertices and edges of G_C that lie in the interior of C. Our aim is to recurse on instances that contain fewer edges in the interior, but not on the boundary, of the cycle delimiting their unbounded face. We first need to handle a possible degenerate case[3] of G_C. Recall that, since G is h-framed, all bounded faces of $G_C \cap \mathrm{sk}(G)$ have length at most h.

▶ **Definition 3.3.** *We say that* G_C *is* separable *if the following conditions hold (see Fig. 2):*

1. *The plane graph* $G_C \cap \mathrm{sk}(G)$ *contains a bounded face* σ_0 *that intersects* C *in* $a \geq 1$ *disjoint maximal subpaths (some of the paths may consist of single vertices). Denoting by* D *the facial cycle of* σ_0, *let* $\tau_1, \ldots, \tau_a$ *be the bounded faces of the plane graph* $C \cup D$ *other than* σ_0.
2. *For each* τ_j, *with* $j = 1, \ldots, a$, *the boundary of* τ_j *is a cycle* C_j *of* $\mathrm{sk}(G)$, *at most two parts of* $\mathcal{R}^0$ *intersect* C_j, *and every part of* $\mathcal{R}^0$ *intersecting* C_j *does so in a single subpath.*[4]

Separable case. Suppose that G_C is separable and let G_j, $j = 1, \ldots, a$, denote the subgraph of G_C bounded by the facial cycle C_j of τ_j[5]. By definition, we have that $|E(G_j) \setminus E(C_j)| \subseteq |E(G_C) \setminus E(C)|$ (even when $a = 1$). Since $E(C_j) \setminus E(C) \neq \emptyset$, the latter implies $|E(G_j) \setminus E(C_j)| < |E(G_C) \setminus E(C)|$. Also, let Y denote the vertices of D that do not belong to C, i.e., $Y = V(D) \setminus V(C)$; refer to the hollow vertices of Fig. 2. By the previous, we have $|Y| \leq |D| - 2 \leq h - 2$.

For $j = 1, \ldots, a$, let $\mathcal{R}^0_j$ be the partition of $V(C_j)$ consisting of the set $Y_j = Y \cap V(C_j)$, if it is not empty, and of the sets $X^j_i = X_i \cap V(C_j)$, $i = 1, \ldots, k$, if X^j_i is not empty; by Property 2 of Definition 3.3, R^0_j consists of at most three parts. Therefore, by a recursive application of our algorithm, each graph G_j, $j = 1, \ldots, a$, admits a good partition $\mathcal{R}_j \supseteq \mathcal{R}^0_j$ of $V(G_j)$.

[3] Such a case does not explicitly occur in the planar proof of [15], but the implicit case of a so-called [15] "tripod" with degenerate legs is analogous to what we are defining here.

[4] A part of $\mathcal{R}^0$ indeed may intersect the boundary C_j of τ_j in two subpaths (see Fig. 2c). This, however, can happen only if $k \leq 2$.

[5] Note that, if $a = 1$, we have $|V(D) \cap V(C)| \geq 2$, since otherwise the face τ_1 would not be bounded by a cycle as required by Property 2 of Definition 3.3.

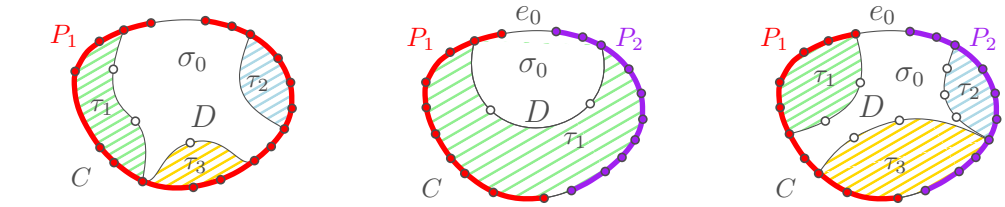


Figure 3 Illustrations of graph $C \cup D$ for the general case of G_C, for $k = 1$ (a) and $k = 2$ (b)(c).

We construct a partition $\mathcal{R}'$ of $V(G_C)$ by putting into $\mathcal{R}'$ the parts of $\mathcal{R}_0$, the set Y (if non-empty), and the recursively obtained parts of each G_j that do not touch C_j; formally, $\mathcal{R}' = \mathcal{R}_0 \cup \{Y\} \cup \bigcup_{j=1,\dots,a}(\mathcal{R}_j \setminus \mathcal{R}_j^0)$, or $\mathcal{R}' = \mathcal{R}_0 \cup \bigcup_{j=1,\dots,a}(\mathcal{R}_j \setminus \mathcal{R}_j^0)$ if $Y = \emptyset$. Note that $\mathcal{R}'$ is indeed a partition of $V(G_C)$, since each vertex of G_C that lies in the interior of C must belong either to Y or to a part $X \in \mathcal{R}_j \setminus \mathcal{R}_j^0$, for some $j \in \{1, \dots, a\}$. We show that the constructed partition $\mathcal{R}'$ is good in [6].

General case. Now we move to the general (i.e., not-necessarily separable) case of G_C in Lemma 3.2. If $\mathbf{k} = \mathbf{1}$, we pick the bounded face σ_0 of $G_C \cap \mathrm{sk}(G)$ incident to the single edge of $E(C) \setminus E(P_1)$; refer to Fig. 3a. The face σ_0 then witnesses the separable case for G_C, by Definition 3.3, which is solved as above. If $\mathbf{k} = \mathbf{2}$, then we pick $e_0 \in E(C)$ as one of the edges joining P_1 and P_2 on C, and σ_0 as the bounded face of $G_C \cap \mathrm{sk}(G)$ incident to e_0; see Figures 3b and 3c. Then we are back to the separable case for G_C with σ_0, by Definition 3.3.

In the remainder, we assume $\mathbf{k} = \mathbf{3}$. First, we color every vertex v of G_C by the color $i \in \{1, 2, 3\}$ if the (unique) path in the BFS tree T from v to the root r hits $V(P_i)$ before possibly hitting other parts of $\mathcal{R}^0$. In particular, the vertices of P_i are colored i. Our aim is to find, in the plane graph $F := G_C \cap \mathrm{sk}(G)$, a bounded face σ_1 containing vertices of all the three colors on its boundary. There our arguments divert from those used in [15] – since F is generally not a near-triangulation, and we additionally need that the face σ_1 intersects the boundary cycle C at most once (which requires additional care). We exploit the following.

▷ Claim 3.4. **(*)** In the setting above, there exists a cycle R bounding a bounded face σ_1 of F, such that $V(R)$ contains all three of our colors, and R intersects C in at most one connected piece. Furthermore, the colors on R appear in three consecutive sections.

Next, consider the set $V(R) \cap V(C)$. If this set contains all three colors, then all three colors occur on the path $R_0 := C \cap R$, and one of them, say color 1, occurs only on internal vertices of R_0 (and nowhere else on C). In this case, the face σ_1 again witnesses the case of separable G_C with $a = 1$ which is solved as above. If, instead, the set $V(R) \cap V(C)$ does not contain all three colors, then we choose on R representatives – vertices $t_i \in V(R)$ of color i where $i = 1, 2, 3$, as in one of the following three possible cases of $V(R) \cap V(C)$ (refer to Fig. 4):

C.1 If $V(R) \cap V(C)$ contains two colors, say 1 and 2, we choose $t_1, t_2 \in V(R) \cap V(C)$ as neighbors on C and $t_3 \in V(R) \setminus V(C)$ arbitrarily; refer to Fig. 4a.

C.2 If $V(R) \cap V(C)$ contains one color, say 1, we choose $t_1 \in V(R) \cap V(C)$ arbitrarily and pick $t_2, t_3 \in V(R) \setminus V(C)$ such that $t_2t_3 \in E(R)$ (this is unique). Furthermore, up to symmetry between the colors 2 and 3, we may assume that the distance on R between t_2 and $V(C)$ is not smaller than the distance on R between t_3 and $V(C)$; refer to Fig. 4b.

C.3 If $V(R) \cap V(C) = \emptyset$, then, up to symmetry between the colors, we may assume that the color 3 occurs in $V(R)$ no more times than each of the colors 1 and 2. We then choose $t_3 \in V(R)$ arbitrarily (of color 3), and set t_1 and t_2 to the two (unique) vertices colored 1 and 2 on R that are neighbors of vertices of color 3 on R; refer to Fig. 4c.

For $i = 1, 2, 3$, let R_i denote the unique vertical path in T from t_i to $V(P_i)$; see Fig. 4. Note that some vertices t_i may lie on C, and then $R_i = t_i$ is a single-vertex path. Let Q be the subpath of R with the ends t_1 and t_2 and avoiding t_3. We define $R' \subseteq R$ as the subpath or cycle (in the case $R' = R$) obtained from R by deleting all internal vertices of Q. Finally, we set $R^+ := R' \cup R_1 \cup R_2 \cup R_3$ which is a connected subgraph of F (R^+ will play the same role here as the so-called tripods in [15]).

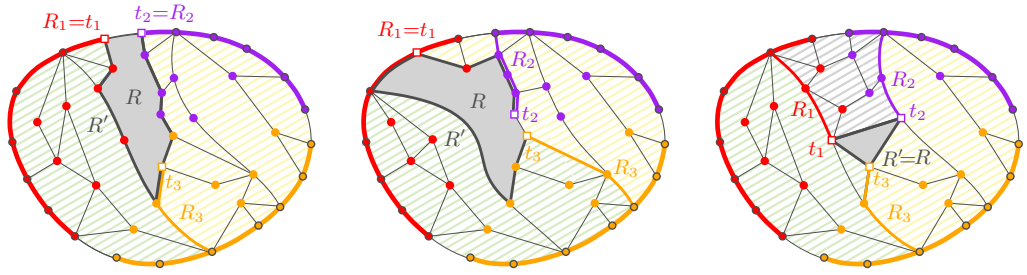
Observe that $C \cup R^+$ is a 2-connected plane graph (in each of the three cases above). Moreover, it contains $a \in \{2, 3\}$ bounded faces $\tau_1, \ldots, \tau_a$, plus the bounded face σ_1 in the case of $R' = R$; for the latter see Fig. 4c. We denote by C_j, $j \in \{1, \ldots, a\}$, the facial cycle of τ_j. It is now important to notice that each cycle C_j intersects at most two parts of $\mathcal{R}^0$, which follows from our "multi-colored" choice of t_1, t_2, t_3 and R_1, R_2, R_3 in all three cases. Furthermore, every two parts of $\mathcal{R}^0$ are together intersected by at most one of C_j.

We next proceed similarly as in the separable case above. Let $G_j \subsetneq G_C$, $j \in \{1, \ldots, a\}$, be the strict subgraph of G_C bounded by C_j, and let $\mathcal{R}^0_j$ be the partition of $V(C_j)$ consisting of $V(C_j) \setminus V(C)$ and of the non-empty parts $X \cap V(C_j)$ over $X \in \mathcal{R}^0$. So, $|\mathcal{R}^0_j| \leq 3$. Therefore, by a recursive application of our algorithm, we may assume that each graph G_j admits a good partition $\mathcal{R}_j \supseteq \mathcal{R}^0_j$ of $V(G_j)$, with $j = 1, \ldots, a$.

We construct a partition $\mathcal{R}' \supseteq \mathcal{R}^0$ of $V(G_C)$ similarly as before; besides $\mathcal{R}^0$ we add the set $Z := V(R^+) \setminus V(C) \neq \emptyset$ as whole, and the recursively obtained parts of each G_j that do not touch C_j. Formally, $\mathcal{R}' = \mathcal{R}_0 \cup \{Z\} \cup \bigcup_{j=1,\ldots,a}(\mathcal{R}_j \setminus \mathcal{R}^0_j)$. Note that $\mathcal{R}'$ is a partition of $V(G_C)$ – in particular, each vertex of G_C which is not on C must belong either to Z or to a part $X \in \mathcal{R}_j \setminus \mathcal{R}^0_j$, for some $j \in \{1, \ldots, a\}$, by induction. We show that the constructed partition $\mathcal{R}'$ is good in [6].

We conclude the proof of Lemma 3.2 by discussing the time complexity of our algorithm, which follows the same ideas as the ones by Dujmovic et al. [15] to compute the decomposition deriving from their product structure theorem for n-vertex planar graphs[6]. To show that a

[6] Note that subsequent improvements have brought the running time of this procedure first to $O(n \log n)$ [24] and finally to $O(n)$ [10].



(a) Case 1. **(b)** Case 2. **(c)** Case 3.

Figure 4 Illustrations for the representatives t_1, t_2, and t_3, and the vertical paths R_1, R_2, and R_3. The vertices of R bound the gray shaded region. R' is depicted with black thick edges.

good partition of G_C can be obtained in $O(|V(G_C)|^2)$ time, it suffices to observe that the non recursive work needed to compute the graphs on which the recursive calls are applied can be easily implemented to run in $O(|V(G_C)|)$ time, by performing a visit of the planar skeleton of the input h-framed graph and of its BFS-tree (provided that G_C is a topological h-framed graph). Since the total number of recursive calls is at most linear in $|V(G_C)|$, the total running time is thus quadratic in $|V(G_C)|$. ◀

Proof of Theorem 3.1. Let C denote the cycle bounding the unbounded face of $\mathrm{sk}(G)$, which, by a possible homeomorphism of the sphere, may be assumed to satisfy $|V(C)| \geq 3$. Based on the BFS tree T of $\mathrm{sk}(G)$ rooted in a vertex $r \in V(C)$, we define the following partition $\mathcal{R}^0$ of C: We split C into a path P_1 only consisting of the vertex r, and two paths P_2 and P_3 of lengths at most $\lfloor \frac{h-1}{2} \rfloor$ and $\lfloor \frac{h}{2} \rfloor$, respectively. Then, we set $\mathcal{R}^0 = \{V(P_1), V(P_2), V(P_3)\}$ and apply the algorithm given in the proof of Lemma 3.2. This way we obtain a good partition $\mathcal{R}'$ of $V(G_C) = V(G)$ and graph $H' := G_C/\mathcal{R}'$ in $O(|V(G)|^2)$ time.

Note that, in general, $G_C \neq G$ as G may have edges drawn in the unbounded face (bounded by C) of $\mathrm{sk}(G)$. However, by setting $H = H'$, we guarantee all edges of G in the unbounded face of $\mathrm{sk}(G)$ are "captured", since the quotient graph H' anyway contains a triangle on the vertices that stem from $\mathcal{R}_0$. In fact, we have just obtained the graph H with the desired properties, i.e., H is planar and of $\mathrm{tw}(H) \leq 3$.

What remains to prove is that G indeed is a subgraph of the strong product $H \boxtimes P \boxtimes K_{3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1}$ for some path P. Recall that the number of layers of the layering $\mathcal{W}$ is $\ell + 1$, and that $\mathcal{W}$ was obtained by merging consecutive $\lfloor \frac{h}{2} \rfloor$-tuples of layers of $\mathcal{L}$. We set P to be the path on $\ell + 1$ vertices denoted in order by $p_0, p_1, \ldots, p_\ell$. To a vertex $v \in V(G)$, we assign the pair (t, p_i) where $t \in V(H)$ if t stems from the part of $\mathcal{R}'$ that v belongs to, and $v \in W_i \in \mathcal{W}$. This assignment is sound and unique.

If $vv' \in E(G)$ is any edge of G, and v and v' are assigned the pairs (t, p_i) and (t', p_j) as above, then $tt' \in E(H)$ or $t = t'$ since $H = G/\mathcal{R}'$ is the quotient graph, and $p_i p_j \in E(P)$ or $i = j$ since $\mathcal{W}$ is a layering of G. Using Property 1 of Lemma 3.2, we furthermore estimate, for every part $X \in \mathcal{R}'$ and its $X' \subseteq X$ (cf. Property 1),

$$\begin{aligned}\lambda_{\mathcal{W}}(X) &\leq |X'| + \lambda_{\mathcal{L}}(X \setminus X') \cdot \lfloor h/2 \rfloor \\ &\leq \max\big(h - 3 + \lfloor h/2 \rfloor,\ \lfloor h/2 \rfloor - 1 + 2\lfloor h/2 \rfloor,\ \lfloor h/3 \rfloor - 1 + 3\lfloor h/2 \rfloor\big) \\ &\leq 3\lfloor h/2 \rfloor + \lfloor h/3 \rfloor - 1\,,\end{aligned}$$

and hence at most $3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1$ vertices of G are assigned to the same pair (t, p_i). This concludes the proof that $\mathrm{si}(G) \subseteq H \boxtimes P \boxtimes K_{3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1}$. ◀

We next present a variant of Theorem 3.1, which reduces the size of the clique in the product by replacing the path with a power of it. The variant does not immediately follow from the statement of Theorem 3.1. However, it can easily be derived by adopting in the proof of Theorem 3.1 the layering $\mathcal{L}$ instead of the layering $\mathcal{W}$.

▶ **Theorem 3.5.** *Let G be an h-framed graph (where G is not necessarily simple). Then* $\mathrm{si}(G)$ *is a subgraph of the strong product of three graphs $H \boxtimes P^{\lfloor h/2 \rfloor} \boxtimes K_{\max(3,h-2)}$, where H is a planar graph with* $\mathrm{tw}(H) \leq 3$ *and P is a path.*

Proof. Recall that $\mathcal{L} = (V_0, V_1, \ldots, V_b)$ is a BFS layering of the skeleton $\mathrm{sk}(G)$, and thus every edge of G has ends in parts $V_i, V_j \in \mathcal{L}$ such that $|i - j| \leq \lfloor h/2 \rfloor$. Hence, we may choose P as the path on $b+1$ vertices $(p_0, p_1, \ldots, p_b)$, use $P^{\lfloor h/2 \rfloor}$, and assign each vertex $v \in V(G)$ to the pair (t, p_i) where $t \in V(H)$ if t stems from the part of $\mathcal{R}'$ that v belongs to, and $v \in V_i \in \mathcal{L}$. Now,

the number of vertices of G assigned to the same pair (t, p_i) (where t stems from a part X) is at most $\lambda_{\mathcal{L}}(X) \leq |X'| + \lambda_{\mathcal{L}}(X \setminus X') \leq \max(h-3+1, \lfloor h/2 \rfloor - 1 + 2, \lfloor h/3 \rfloor - 1 + 3) = \max(3, h-2)$. This concludes that $\mathrm{si}(G) \subseteq H \boxtimes P^{\lfloor h/2 \rfloor} \boxtimes K_{\max(3,h-2)}$. ◀

4 Consequences of the Product Structure

As mentioned in the introduction, Dujmović et al. [17] have derived upper bounds on the queue number, on the non-repetitive chromatic number, and on the p-centered chromatic number of k-planar and k-map graphs exploiting Theorem 2.1. In the following, we present our improvements to each of these problems.

Queue number. A *queue layout* of a graph G is a linear order σ of the vertices of G together with an assignment of its edges to sets, called *queues*, such that no two edges in the same set nest. The *queue number* $\mathrm{qn}(G)$ of a graph G is the minimum number of queues over all queue layouts of G. In [15], Dujmović et al. have proved the following useful lemma concerning the queue number of graphs that can be expressed as subgraphs of the strong product of a path P, a graph H with queue number $\mathrm{qn}(H)$, and a clique K_ℓ on ℓ vertices.

▶ **Lemma 4.1** (Dujmović et al. [15]). *If $G \subseteq P \boxtimes H \boxtimes K_\ell$ then $\mathrm{qn}(G) \leq 3\ell\,\mathrm{qn}(H) + \lfloor \frac{3}{2}\ell \rfloor$.*

Combining Lemma 4.1 and Theorem 2.1(d), together with the fact that the queue number of planar 3-trees is at most 5 [1], Dujmović, Morin, and Wood showed the first constant upper bound on the queue number of k-planar graphs [17], thus resolving a long-standing open question. Analogously, by combining Lemma 4.1 and Theorem 3.1, we obtain the following.

▶ **Theorem 4.2.** *The queue number of h-framed graphs is at most*

$$\left\lfloor \frac{33}{2} \left(3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1 \right) \right\rfloor .$$

Dujmović et al. [15] first showed the queue number of k-map graphs is at most $2(98(k+1))^3$. Later, by combining Theorem 2.1(e) and Lemma 4.1, Dujmović et al. [16] improved this bound to $32225k(k-3)$. More recently, Dujmović et al. [17] have also observed that k-map graphs are k-framed and have exploited this observation to further improve this bound to $\lfloor \frac{33}{2}(k + 3\lfloor \frac{k}{2} \rfloor - 3) \rfloor$. By Theorem 4.2, we can further improve these bounds by also leveraging the fact that these graphs are subgraphs of k-framed graphs [3, 5].

▶ **Corollary 4.3.** *The queue number of k-map graphs is at most*

$$\left\lfloor \frac{33}{2} \left(3\lfloor \frac{k}{2} \rfloor + \lfloor \frac{k}{3} \rfloor - 1 \right) \right\rfloor .$$

For $h \in 4, 5$, Theorem 4.2 gives us an upper bound of 95. Since any 1-planar graph can be augmented to a (not-necessarily simple) 4-framed graph [2], Theorem 4.2 improves the currently best upper bound of 1-planar graphs from 115 [17] to 95. Since any optimal 2-planar graph is 5-framed, Theorem 4.2 improves the currently best upper bound on their queue number from 132 [17] to 95. Next, we show a generalization of Lemma 4.1 that allows further improvements.

▶ **Lemma 4.4.** **(*)** *If $G \subseteq H \boxtimes P^i \boxtimes K_\ell$ then $\mathrm{qn}(G) \leq i\ell + (2i+1)\ell\mathrm{qn}(H) + \lfloor \frac{\ell}{2} \rfloor$.*

Lemma 4.4 in conjunction with Theorem 3.5 yields a quadratic (in h) upper bound on the queue number of h-framed graphs. However, for $h \leq 5$, it implies an improved bound on the queue number of 1-planar and optimal 2-planar graph, which we summarize in the following.

▶ **Theorem 4.5.** *The queue number of 1-planar and optimal 2-planar graphs is at most 82.*

Non-repetitive chromatic number. An *r-coloring* of a graph G is a function $\phi : V(G) \to [r]$. A path $p = (v_1, v_2, \ldots, v_{2\tau})$ is *repetitively colored* by ϕ if $\phi(v_i) = \phi(v_{i+\tau})$ for $i = 1, 2, \ldots, \tau$. A coloring ϕ of G is *non-repetitive* if no path of G is repetitively colored by ϕ. Clearly, non-repetitive colorings are *proper*, i.e., $\phi(u) \neq \phi(v)$ if u and v are adjacent in G. The *non-repetitive chromatic number* $\pi(G)$ of G is the minimum integer r such that G admits a non-repetitive r-coloring. In [18], Dujmović et al. developed the following lemma to upper-bound the non-repetitive chromatic number of graphs that can be expressed as subgraphs of the strong product of a path P, a graph H with $\mathrm{tw}(H)$, and a clique K_ℓ on ℓ vertices.

▶ **Lemma 4.6** (Dujmović et al. [18]). *If $G \subseteq P \boxtimes H \boxtimes K_\ell$, then $\pi(G) \leq 4^{tw(H)+1} \cdot \ell$.*

Using Lemma 4.6 and Theorem 2.1(c), Dujmović et al. [17] provide an upper bound of 1792 and of 2048 on the non-repetitive chromatic number of 1-planar and optimal 2-planar graphs, respectively. Since 1-planar and optimal 2-planar graphs are 4-framed and 5-framed, respectively, we improve both bounds to 1536. By Lemma 4.6 and the product structure theorem for h-framed graph in [17], one can obtain an upper bound of $4^4 \cdot (h + 3\lfloor \frac{h}{2} \rfloor - 3)$ for the non-repetitive chromatic number of the graphs in this family. By Lemma 4.6 and Theorem 3.1, we can further improve this upper bound to $4^4 \cdot (3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1)$. Also, since k-map graphs are subgraphs of k-framed graphs [3, 5], their non-repetitive chromatic number is also improved from to $4^4 \cdot (k + 3\lfloor \frac{k}{2} \rfloor - 3)$ to $4^4 \cdot (3\lfloor \frac{k}{2} \rfloor + \lfloor \frac{k}{3} \rfloor - 1)$. Specifically, we get the following.

▶ **Corollary 4.7.** *For a graph G, it holds:*

- $\pi(G) \leq 4^4 \cdot 6$, *if G is 1-planar,*
- $\pi(G) \leq 4^4 \cdot (3\lfloor \frac{k}{2} \rfloor + \lfloor \frac{k}{3} \rfloor - 1)$, *if G is k-map, and*
- $\pi(G) \leq 4^4 \cdot (3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1)$, *if G is h-framed.*

p-centered chromatic number. For any $c, p \in \mathbb{N}$ with $c \geq p$, a c-coloring of a graph G is *p-centered* if, for every connected component X of G, at least one of the following holds: (i) the vertices of X are colored with more than p colors, or (ii) there exists a vertex v of X that is assigned a color different from the ones of the remaining vertices of X. For any graph G, the *p-centered chromatic number* $\chi_p(G)$ of G is the minimum integer c such that G admits a p-centered c-coloring. The following lemma is implied by combining results by Pilipczuk and Siebertz [26], Debski et al. [11] and Dujmović et al. [17].

▶ **Lemma 4.8** ([11, 17, 26]). *If $G \subseteq P \boxtimes H \boxtimes K_\ell$, where H is a planar graph with $\mathrm{tw}(H) \leq 3$, it holds that $\chi_p(G) \leq \ell(p+1)^2(p\lceil \log(p+1) + 2p + 1)$.*

By Lemma 4.8 and Theorem 2.1, Dujmović et al. [17] showed the following upper bounds: $\chi_p(G) \leq 7(p+1)^2(p\lceil \log(p+1) + 2p + 1)$ if G is a 1-planar graph, $\chi_p(G) \leq 8(p+1)^2(p\lceil \log(p+1) + 2p + 1)$ if G is an optimal 2-planar graph, $\chi_p(G) \leq (k + 3\lfloor \frac{k}{2} \rfloor - 3)(p+1)^2(p\lceil \log(p+1) + 2p + 1)$ if G is a k-map graph, and $\chi_p(G) \leq (h + 3\lfloor \frac{h}{2} \rfloor - 3)(p+1)^2(p\lceil \log(p+1) + 2p + 1)$ if G is an h-framed graph. By exploiting Theorem 3.1 and Lemma 4.8, we get the next.

▶ **Corollary 4.9.** *For a graph G, it holds:*

- $\chi_p(G) \leq 6(p+1)^2(p\lceil \log(p+1) + 2p + 1)$, *if G is 1-planar or optimal 2-planar,*
- $\chi_p(G) \leq (3\lfloor \frac{k}{2} \rfloor + \lfloor \frac{k}{3} \rfloor - 1)(p+1)^2(p\lceil \log(p+1) + 2p + 1)$, *if G is k-map, and*
- $\chi_p(G) \leq (3\lfloor \frac{h}{2} \rfloor + \lfloor \frac{h}{3} \rfloor - 1)(p+1)^2(p\lceil \log(p+1) + 2p + 1)$, *if G is h-framed.*

5 Bounding Twin-width

Besides the direct consequences of the product structure theorem(s) surveyed in Section 4, the construction presented in Section 3 has another strong implication described next.

Consider only simple graphs for the coming definition.[7] A *trigraph* is a simple graph G in which some edges are marked as *red*, and we then naturally speak about *red neighbors* and *red degree* in G. We denote the set of red neighbors of a vertex v by $N_r(v)$. For a pair of (possibly not adjacent) vertices $x_1, x_2 \in V(G)$, we define a *contraction* of the pair x_1, x_2 as the operation creating a trigraph G' which is the same as G except that x_1, x_2 are replaced with a new vertex x_0 whose full neighborhood is the union of neighborhoods of x_1 and x_2 in G, that is, $N(x_0) = (N(x_1) \cup N(x_2)) \setminus \{x_1, x_2\}$, and the red neighbors $N_r(x_0)$ of x_0 inherit all red neighbors of x_1 and of x_2 and those in $N(x_1) \oplus N(x_2)$, that is, $N_r(x_0) = \big((N_r(x_1) \cup N_r(x_2)) \setminus \{x_1, x_2\}\big) \cup \big(N(x_1) \oplus N(x_2)\big)$, where $\oplus$ denotes the symmetric difference.

A *contraction sequence* of a trigraph G is a sequence of successive contractions turning G into a single vertex, and its *width* is the maximum red degree of any vertex in any trigraph of the sequence. The *twin-width* is the minimum width over all possible contraction sequences (where for an ordinary graph, we start with the same trigraph having no red edges). As noted already in the pioneering paper on this concept [8], the twin-width of k-planar graphs is bounded for any fixed k by means of FO transductions (which, therefore, gives a not-even-asymptotically expressible bound). Explicit asymptotic bounds for the twin-width of k-planar graphs (albeit with $O(k)$ in the exponent, and so not giving an explicit number for e.g. $k = 1$) are in [9] (with a generalization to higher surfaces), and specially for planar graphs, the current upper bounds on twin-width are 583 in [9], improved to 183 in [23], to 37 in [6] and to 9 in [22], which is currently the best known one. It is worth to mention that both [9, 23], more or less explicitly, use the product structure machinery of planar graphs. Here, we give a new explicit (non-asymptotic) bounds on the twin-width of h-framed and 1-planar graphs.

▶ **Theorem 5.1. (*)** *Let G be a simple spanning subgraph of an h-framed graph with $h \geq 4$. Then the twin-width of G is at most $33\lfloor h/2 \rfloor + \lfloor h/3 \rfloor + 13 \leq 17h + 13$.*

Proof sketch. The rough idea (inspired in part by [23, Section 4]) is to recursively decompose G into subgraphs bounded by cycles of the plane skeleton of an h-framed supergraph of G – the same decomposition that has been used to obtain the product structure in Lemma 3.2. Then, on the "way back" from the recursion, we contract vertices inside these cycles which are at the same time in the same BFS layer in a controlled way, that is, not creating too-high red degrees in general, and completely avoiding red edges to suitably selected vertices on the bounding cycles (however, unlike [23], we do not exploit planarity for the latter task). ◀

▶ **Corollary 5.2.** *The twin-width of simple 1-planar and optimal 2-planar graphs is at most 80.*

We point out that Theorem 5.1 implies an improvement on the twin-width of k-map graphs only up to a certain k, as these graphs have bounded twin-width independently of k [8].

[7] In general, the concept of twin-width is defined for binary relational structures of a finite signature, and so one may either define the twin-width of a multigraph as the twin-width of its simplification, or allow only bounded multiplicities of edges and use the more general matrix definition of twin-width.

6 Conclusions

In this paper we have provided a product structure theorem for h-framed graphs. Our approach is constructive and can easily be implemented to run in quadratic time to obtain the corresponding decomposition, provided that the input graph is a topological h-framed graph.

A major open question is to obtain a speed up in the construction; the recent algorithmic advances in [10, 24] have the potential to lead to improvements in the running time. Another important open problem is whether each k-planar graph is a subgraph of the strong product of a path, a (planar) graph of constant treewidth, and a clique whose size is a function of k; our results suggest that such a structure might be possible.

References

1 Jawaherul Md. Alam, Michael A. Bekos, Martin Gronemann, Michael Kaufmann, and Sergey Pupyrev. Queue layouts of planar 3-trees. *Algorithmica*, 82(9):2564–2585, 2020. doi:10.1007/s00453-020-00697-4.

2 Md. Jawaherul Alam, Franz J. Brandenburg, and Stephen G. Kobourov. Straight-line grid drawings of 3-connected 1-planar graphs. In Stephen K. Wismath and Alexander Wolff, editors, *Graph Drawing*, volume 8242 of *LNCS*, pages 83–94. Springer, 2013. doi:10.1007/978-3-319-03841-4_8.

3 Michael A. Bekos, Giordano Da Lozzo, Svenja Griesbach, Martin Gronemann, Fabrizio Montecchiani, and Chrysanthi N. Raftopoulou. Book embeddings of nonplanar graphs with small faces in few pages. *CoRR*, abs/2003.07655, 2020. arXiv:2003.07655.

4 Michael A. Bekos, Michael Kaufmann, and Chrysanthi N. Raftopoulou. On optimal 2- and 3-planar graphs. In Boris Aronov and Matthew J. Katz, editors, *SoCG*, volume 77 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.SoCG.2017.16.

5 Michael A. Bekos, Giordano Da Lozzo, Svenja Griesbach, Martin Gronemann, Fabrizio Montecchiani, and Chrysanthi N. Raftopoulou. Book embeddings of nonplanar graphs with small faces in few pages. In *SoCG*, volume 164 of *LIPIcs*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

6 Michael A. Bekos, Giordano Da Lozzo, Petr Hlinený, and Michael Kaufmann. Graph product structure for h-framed graphs. *CoRR*, abs/2204.11495, 2022. arXiv:2204.11495.

7 Marthe Bonamy, Cyril Gavoille, and Michal Pilipczuk. Shorter labeling schemes for planar graphs. In Shuchi Chawla, editor, *SODA*, pages 446–462. SIAM, 2020. doi:10.1137/1.9781611975994.27.

8 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022.

9 Édouard Bonnet, O-joung Kwon, and David R. Wood. Reduced bandwidth: a qualitative strengthening of twin-width in minor-closed classes (and beyond). *CoRR*, abs/2202.11858, 2022. arXiv:2202.11858.

10 Prosenjit Bose, Pat Morin, and Saeed Odak. An optimal algorithm for product structure in planar graphs. *CoRR*, abs/2202.08870, 2022. arXiv:2202.08870.

11 Michal Debski, Stefan Felsner, Piotr Micek, and Felix Schröder. Improved bounds for centered colorings. In Shuchi Chawla, editor, *SODA*, pages 2212–2226. SIAM, 2020. doi:10.1137/1.9781611975994.136.

12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Vida Dujmovic, Louis Esperet, Cyril Gavoille, Gwenaël Joret, Piotr Micek, and Pat Morin. Adjacency labelling for planar graphs (and beyond). In *FOCS*. IEEE, 2020. doi:10.1109/FOCS46700.2020.00060.

14 Vida Dujmovic, Louis Esperet, Pat Morin, Bartosz Walczak, and David R. Wood. Clustered 3-colouring graphs of bounded degree. *Comb. Probab. Comput.*, 31(1):123–135, 2022. doi:10.1017/S0963548321000213.

15 Vida Dujmovic, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. doi:10.1145/3385731.

16 Vida Dujmovic, Pat Morin, and David R. Wood. Graph product structure for non-minor-closed classes. *CoRR*, abs/1907.05168v3, 2020. arXiv:1907.05168v3.

17 Vida Dujmovic, Pat Morin, and David R. Wood. Graph product structure for non-minor-closed classes. *CoRR*, abs/1907.05168v4, April 2022. arXiv:1907.05168v4.

18 Vida Dujmović, Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David Wood. Planar graphs have bounded nonrepetitive chromatic number. *Advances in Combinatorics*, March 2020. doi:10.19086/aic.12100.

19 Zdenek Dvořák, Tony Huynh, Gwenaël Joret, Chun-Hung Liu, and David R. Wood. Notes on graph product structure theory. *CoRR*, abs/2001.08860, 2020. arXiv:2001.08860.

20 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Parameterized complexity in graph drawing (dagstuhl seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021. doi:10.4230/DagRep.11.6.82.

21 Lenwood S. Heath, Frank Thomson Leighton, and Arnold L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discrete Math.*, 5(3):398–412, 1992. doi:10.1137/0405031.

22 Petr Hliněný. Twin-width of planar graphs is at most 9. *CoRR*, abs/2205.05378, 2022. arXiv:2205.05378.

23 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. *CoRR*, abs/2201.09749, 2022. arXiv:2201.09749.

24 Pat Morin. A fast algorithm for the product structure of planar graphs. *Algorithmica*, 83(5):1544–1558, 2021. doi:10.1007/s00453-020-00793-5.

25 Pat Morin, Vida Dujmović, Sergey Norin, and David Wood. Graph product structure theory. Banff Int., November 21-26 2021.

26 Michal Pilipczuk and Sebastian Siebertz. Polynomial bounds for centered colorings on proper minor-closed graph classes. *J. Comb. Theory, Ser. B*, 151:111–147, 2021. doi:10.1016/j.jctb.2021.06.002.

27 Torsten Ueckerdt, David R. Wood, and Wendy Yi. An improved planar graph product structure theorem. *CoRR*, abs/2108.00198, 2021. arXiv:2108.00198.

Hardness of Approximation for H-Free Edge Modification Problems: Towards a Dichotomy

Tatiana Belova
St. Petersburg Department of Steklov Mathematical Institute of the RAS, Russia

Ivan Bliznets
St. Petersburg Department of Steklov Mathematical Institute of the RAS, Russia

Abstract

For a fixed graph H, the H-free Edge Deletion/Completion/Editing problem asks to delete/add/edit the minimum possible number of edges in G to get a graph that does not contain an induced subgraph isomorphic to the graph H. In this work, we investigate H-free Edge Deletion/Completion/Editing problems in terms of the hardness of their approximation. We formulate a conjecture according to which all the graphs with at least five vertices can be classified into several groups of graphs with specific structural properties depending on the hardness of approximation for the corresponding H-free Edge Deletion/Completion/Editing problems. Also, we make significant progress in proving that conjecture by showing that it is sufficient to resolve it only for a finite set of graphs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized complexity, Hardness of approximation, Edge modification problems

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.24

Related Version *Full Version*:
`https://drive.google.com/file/d/1jgaKmHye9dABKLt-6jt9haVcL4cNAPDC/view?usp=sharing`

Funding The research is supported by Russian Science Foundation (project 18-71-10042).

1 Introduction

A graph modification problem is usually formulated in the following way. There is a fixed set of allowed graph modifications, e.g., removing vertices or edges. Given a graph G, the problem asks to find the minimum number of allowed modifications such that the resulting graph satisfies a certain property. Some of the well-known graph problems can be formulated as graph modification problems. For example, in the Vertex Cover problem, the task is to remove the minimum possible number of vertices to make the resulting graph contain no edges. Usually, graph modification problems are NP-hard. In [19], Lewis and Yannakakis showed that if the property that a graph should satisfy after modifications is non-trivial (that means that it holds for an infinite set of graphs and does not hold either for an infinite set of graphs) and inherited (that means that if a graph G satisfies that property then each induced subgraph of G also satisfies that property), and the only allowed modification is removing vertices, then such problems are NP-hard. However, for edge modification problems the situation is more complicated. In the early works [21, 12] devoted to edge modification problems, it has been observed that although we can obtain results for specific classes of problems, it can be difficult to generalize them to all edge modification problems.

Graph modification problems are extensively studied from the point of view of parameterized complexity [1, 4, 5, 7, 9, 11, 13, 16, 20], because for practical reasons the number of allowed modifications is usually small compared to the size of the graph, and then it is reasonable to consider the number of modifications as a parameter. For a detailed survey of parameterized algorithms for modification problems we recommend [9].

One of the typical representatives of graph modification problems are H-free Edge Deletion, H-free Edge Completion and H-free Edge Editing for every fixed graph H. In the H-free Edge Deletion problem, given a graph G, the task is to remove the minimum

number of edges from G such that the resulting graph does not contain an induced subgraph isomorphic to H. Similarly, in H-free Edge Completion we need to add the smallest number of edges and in H-free Edge Editing we are allowed to use both edge deletion and edge completion.

Using a branching method, Cai [6] showed that H-free Edge Deletion (Completion, Editing) can be solved in $c^k \cdot poly(n)$ time, where k is an upper bound for the number of allowed modifications, n is the size of graph, and c depends only on the graph H, which is fixed for each problem. Also, in general it is unlikely that H-free Edge Deletion (Completion, Editing) can be solved in time significantly less than $c^k \cdot poly(n)$, because Aravind et al. [1] showed that for all graphs H with at least two edges (non-edges), H-free Edge Deletion (Completion) are NP-hard and cannot be solved in $2^{o(k)} \cdot poly(n)$ time, if the ETH holds, and for all graphs H with at least three vertices, H-free Edge Editing is NP-hard and cannot be solved in $2^{o(k)} \cdot poly(n)$ time, if the ETH holds.

H-free Edge Deletion (Completion, Editing) are also considered from the kernelization point of view, in particular, in terms of polynomial kernel existence. Kratsch and Wahlström [18] presented a graph H such that H-free Edge Deletion and H-free Edge Editing do not admit a polynomial kernel unless $NP \subseteq coNP/poly$. Later, Guillemot et al. [15] showed that for all graphs H, which are cycles on at least four vertices or paths on at least seven vertices, H-free Edge Deletion does not admit a polynomial kernel unless $NP \subseteq coNP/poly$. That result was improved in [7, 8] by Cai et al. They showed that if H is a path or a cycle, then H-free Edge Deletion (Completion, Editing) does not admit a polynomial kernel unless $NP \subseteq coNP/poly$, if and only if H contains at least four edges. Also, Cai et al. [7, 8] showed that if H is a 3-connected graph and $NP \not\subseteq coNP/poly$, then H-free Edge Deletion and H-free Edge Editing do not admit a polynomial kernel if and only if H is not a complete graph, and H-free Edge Completion does not admit a polynomial kernel if and only if H contains at least two non-edges. Marx and Sandeep [20] made an attempt to classify all graphs H on at least five vertices depending on the existence of polynomial kernels for the corresponding problems. They formulated conjectures that if $NP \not\subseteq coNP/poly$ then, for a graph H on at least five vertices, H-free Edge Editing admits polynomial kernel if and only if H is complete or empty, and H-free Edge Deletion admits polynomial kernel if and only if H is either complete or contains at most one edge. Despite the fact that those conjectures are not yet proven, Marx and Sandeep [20] showed that to prove them it is sufficient only to show that graphs from some finite set do not admit a polynomial kernel.

H-free Edge Deletion (Completion, Editing) are also studied in terms of approximation complexity. Since removing and adding edges can create new subgraphs isomorphic to H, it is unlikely to find an exact or approximate solution using greedy methods. Generally, there is a connection between the existence of polynomial kernel and $poly(OPT)$-approximation for a problem, because usually one can find $poly(OPT)$-approximation having polynomial kernel, and vice versa. Despite the fact that this connection is not formal and there are problems [14] that act differently, it is quite possible that such connection exists for the H-free Edge Deletion (Completion, Editing) problems and we can use results for kernelization complexity in the case of approximation complexity. Bliznets et al. [3] showed that for H which is a 3-connected graph with at least two non-edges, and for H, which is a cycle on at least four vertices or a path on at least five vertices, H-free Edge Deletion and H-free Edge Completion do not admit $poly(OPT)$-approximation unless $P = NP$. Those results are similar to results for kernelization complexity obtained by Cai et al. in [7, 8]. However, in contrast to the kernelization case, in case of approximation complexity, the existence of at least two non-edges in H is crucial for H-free Edge Deletion. Bliznets et al. [3] also showed

Table 1 Conjecture.

	Deletion	Completion	Editing
Exact	$\overline{K_n}$, $\overline{K_n - e}$	$K_n, K_n - e$	—
Constant	K_n	$\overline{K_n}$	$\overline{K_n}$, K_n
Min Horn Deletion	$K_n - e$	$\overline{K_n - e}$	$K_n - e$, $\overline{K_n - e}$
no poly(OPT)	all other graphs	all other graphs	all other graphs

For each of H-free Edge Deletion/Completion/Editing problems, the table represents a partition of all graphs on at least five vertices into classes depending on the hardness of approximation for the corresponding problems.

Deletion, *Completion* and *Editing* columns correspond to H-free Edge Deletion, H-free Edge Completion and H-free Edge Editing problems, respectively.

Exact row corresponds to the existence of exact polynomial-time algorithm.

Constant row corresponds to the existence of constant approximation.

Min Horn Deletion row corresponds to Min Horn Deletion-completeness.

no poly(OPT) row corresponds to non-existence of *poly*(OPT)-approximation unless $P = NP$.

that $(K_n - e)$-free Edge Deletion problem is Min Horn Deletion-complete for every $n \geq 5$. Min Horn Deletion-complete problems form a separate class of problems, and for now we do not know much about them. Khanna et al. [17] showed that those problems belong to the class poly-APX, but they do not admit $2^{\log^{1-\varepsilon} n}$-approximation for any $\varepsilon > 0$ unless $P = NP$.

Our results. Taking into account a significant progress in classification of H-free Edge Modification problems from the point of view of kernelization in [20] and mentioned similarities of approximation and kernelization [2], we make an attempt to classify all graphs on at least five vertices depending on approximation complexity of the corresponding H-free Edge Deletion (Completion, Editing) problems. Let us formulate a conjecture about such classification for each of the H-free Edge Deletion, H-free Edge Completion and H-free Edge Editing problems.

▶ **Conjecture 1.** *Let H be a graph with at least five vertices. Then, the approximation complexity of H-free Edge Deletion (Completion, Editing) can be determined according to Table 1.*

In fact, this conjecture repeats the conjectures for the case of kernelization, except that it has an additional claim about Min Horn Deletion-complete problems. Thus, on the one hand, we believe that approximation complexity and kernelization complexity for the H-free Edge Deletion (Completion, Editing) problems behave very similarly, and on the other hand, in the case of approximation complexity, it seems that we observe a slightly more complicated situation.

Unfortunately, we were unable to prove the statement of the conjecture. However, we manage to prove the following theorem.

▶ **Theorem 2.** *There exists a set $\mathcal{G}$ (see Table 2) of seventeen graphs such that:*

(i) *$\forall H \in \mathcal{G} \cup \overline{\mathcal{G}}$, H-free Edge Deletion (Completion) does not admit poly(OPT)-approximation $\Leftrightarrow$ conjecture for H-free Edge Deletion (Completion) holds;*

(ii) *$\forall H \in \mathcal{G}$, H-free Edge Editing does not admit poly(OPT)-approximation $\Leftrightarrow$ conjecture for H-free Edge Editing holds.*

We think we achieve a significant progress in classification of H-free Edge Deletion/ Completion/Editing problems as well as a huge step in resolving conjecture from [3] that assumes that H-free edge modification problems behave in similar way from approximation and kernelization points of view.

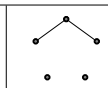
As we note before, for the edge modification problems there are similarities between kernelization and approximation results. However, the proofs of approximation and kernelization results are far from being identical, one of the reasons of this stems from the fact that $(K_n - e)$-Free Edge Deletion is only Min Horn Deletion-complete from approximation point of view and yet it does not admit a polynomial kernel unless $NP \subseteq coNP/poly$. This fact could potentially lead to the existence of a significantly bigger class of graphs H for which H-Free Edge Deletion is Min Horn Deletion-complete.

It seems that from approximation point of view classification is more complicated than from kernelization point of view. For example, in [7], a full classification for H being 3-connected graph was given, while in its approximation analog paper [3] almost a full classification was given with only exception of $H = K_n - e$. It was shown that $(K_n - e)$-Free Edge Deletion is Min Horn Deletion-complete, however, it neither rules out the possibility of the existence of $poly(OPT)$-approximation nor shows that such an approximation exists. Similarly, in [20], for full classification of kernelization it is left to prove a lack of kernel for 9 graphs for H-Free Edge Editing problem and 19 graphs for H-Free Edge Deletion/Completion problems, while the respective numbers that we achieve for approximation classification are 17 and 33 graphs.

We note that we significantly extend results for H-free Edge Deletion (Completion) from work [3], but we also consider Editing version of the H-free Edge Modification problem. The H-free Edge Editing problem was not considered in [3].

All proofs here are omitted due to space constraints, they can be found in the full version of the paper.

Table 2 $\mathcal{G} \cup \overline{\mathcal{G}}$.

G_1		$\overline{G_1}$		G_7		$\overline{G_7}$		G_{13}		$\overline{G_{13}}$	
G_2		$\overline{G_2}$		G_8		$\overline{G_8}$		G_{14}		$\overline{G_{14}}$	
G_3		$\overline{G_3}$		G_9		$\overline{G_9}$		G_{15}		$\overline{G_{15}}$	
G_4		$\overline{G_4}$		G_{10}		$\overline{G_{10}}$		G_{16}		$\overline{G_{16}}$	
G_5		$\overline{G_5}$		G_{11}		$\overline{G_{11}}$		G_{17}		$\overline{G_{17}}$	
G_6		$\overline{G_6}$		G_{12}		$\overline{G_{12}}$					

2 Preliminaries

In this paper, we consider only simple graphs. All used notations for graphs are standard and can be found in [10]. For a fixed graph G, let V_ℓ be the set of vertices in G with the minimum degree, and let V_h be the set of vertices with the maximum degree. For a set of graphs $\mathcal{X}$, let $\overline{\mathcal{X}}$ be the set of complements of the graphs in $\mathcal{X}$. According to [20], we define the following sets of graphs: $\mathcal{H}$, $\mathcal{A}$, $\mathcal{D}$, $\mathcal{B}$, $\mathcal{S}$, $\mathcal{F}$ and their complements $\overline{\mathcal{H}}$, $\overline{\mathcal{A}}$, $\overline{\mathcal{D}}$, $\overline{\mathcal{B}}$, $\overline{\mathcal{S}}$ and $\overline{\mathcal{F}}$. The graphs from sets $\mathcal{H}$, $\overline{\mathcal{H}}$, $\mathcal{A}$, $\overline{\mathcal{A}}$, $\mathcal{D}$, $\overline{\mathcal{D}}$, $\mathcal{B}$, $\overline{\mathcal{B}}$, $\mathcal{S}$ and $\overline{\mathcal{S}}$ are shown in Tables 3, 4, 5 and 6. $\mathcal{F}$ is a union of sets of graphs $\mathcal{F}_i$ which are shown in Table 7. Let $\mathcal{W}$ be the set $\mathcal{H} \cup \overline{\mathcal{H}} \cup \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{D} \cup \overline{\mathcal{D}} \cup \mathcal{B} \cup \overline{\mathcal{B}} \cup \mathcal{S} \cup \overline{\mathcal{S}} \cup \mathcal{F} \cup \overline{\mathcal{F}}$. We notice that $\overline{\mathcal{W}} = \mathcal{W}$.

In the H-free Edge Deletion problem, given a graph G and an integer k, we need to determine whether one can delete at most k edges from G such that the resulting graph does not contain an induced graph isomorphic to H. Let us call a set of edges F a *solution* for an instance (G, k) of H-free Edge Deletion if $|F| \leq k$ and $G - F$ does not contain an induced graph isomorphic to H. H-free Edge Completion and H-free Edge Editing and solutions for them are defined in a similar way. In the optimization version of the H-free Edge Deletion (Completion, Editing) problem, given a graph G, we need to determine the minimum possible k such that there exists a solution F for an instance (G, k). We will refer to both decision and optimization versions, it should be clear from the context which version we refer to.

In the Sandwich H-free Edge Deletion problem, given a graph G and a set $D \subseteq E(G)$ of *undeletable* edges, we need to determine whether there exists a subset $F \subseteq E(G) \setminus D$ such that $G - F$ does not contain H as an induced subgraph. Let us call such F a *solution* for the instance (G, D) of Sandwich H-free Edge Deletion. Sandwich H-free Edge Completion and its solution are defined in a similar way. The size of solution F is not important for us, we are interested only in its existence.

Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function. An *$f(OPT)$-approximation for an instance I* of a minimization problem X is a solution F for I such that $|F| \leq f(OPT(I)) \cdot OPT(I)$ where $OPT(I)$ is the size of the optimal solution for I. An algorithm for a minimization problem X is called *$f(OPT)$-approximation algorithm* if for any instance I of X it finds an $f(OPT)$-approximation for I. We say that a minimization problem X *admits $f(OPT)$-approximation* if there exists an $f(OPT)$-approximation algorithm for X working in polynomial time. It is easy to show the following results.

▶ **Lemma 3.**

(i) *H-free Edge Deletion admits $f(OPT)$-approximation $\Leftrightarrow$ $\overline{H}$-free Edge Completion admits $f(OPT)$-approximation.*

(ii) *H-free Edge Editing admits $f(OPT)$-approximation $\Leftrightarrow$ $\overline{H}$-free Edge Editing admits $f(OPT)$-approximation.*

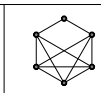
Table 3 $\mathcal{H} \cup \overline{\mathcal{H}}$.

H_1		$\overline{H_1}$		H_4		$\overline{H_4}$		H_7		$\overline{H_7}$	
H_2		$\overline{H_2}$		H_5		$\overline{H_5}$		H_8		$\overline{H_8}$	
H_3		$\overline{H_3}$		H_6		$\overline{H_6}$		H_9		$\overline{H_9}$	

Table 4 $\mathcal{A} \cup \overline{\mathcal{A}}$.

A_1		$\overline{A_1}$		A_4		$\overline{A_4}$		A_7		$\overline{A_7}$	
A_2		$\overline{A_2}$		A_5		$\overline{A_5}$		A_8		$\overline{A_8}$	
A_3		$\overline{A_3}$		A_6		$\overline{A_6}$		A_9		$\overline{A_9}$	

Table 5 $\mathcal{D} \cup \overline{\mathcal{D}} \cup \mathcal{B} \cup \overline{\mathcal{B}}$.

D_1		$\overline{D_1}$		B_1		$\overline{B_1}$		B_3		$\overline{B_3}$	
D_2		$\overline{D_2}$		B_2		$\overline{B_2}$					

▶ **Lemma 4.** *$\overline{K_n}$-free Edge Deletion, $\overline{K_n - e}$-free Edge Deletion, K_n-free Edge Completion and $(K_n - e)$-free Edge Completion admit polynomial-time algorithms.*

▶ **Lemma 5.** *K_n-free Edge Deletion, $\overline{K_n}$-free Edge Completion, K_n-free Edge Editing and $\overline{K_n}$-free Edge Editing admit $\binom{n}{2}$-approximation.*

3 Min Horn Deletion-complete problems

Now let us consider graphs H such that H-free Edge Deletion (Completion, Editing) is Min Horn Deletion-complete. At first, we describe which problems are called Min Horn Deletion-complete and introduce some definitions. In [17], there was an attempt to classify all constraint satisfaction problems by their approximation complexity. For some problems, the complexity of approximation was not established, but they were divided into classes in which all problems are equivalent to each other with respect to A-reduction, which we define below. Then, for each such class, we can choose any problem as a representative. Min Horn Deletion problem was chosen as one of those representatives. In Min Horn Deletion problem, we are given a Boolean formula φ in CNF, such that each clause of φ contains either a single literal or three literals where exactly one literal is negative. The task is to find an assignment that minimizes the number of unsatisfied clauses of φ.

We say that there is an *A-reduction* from a problem P to a problem Q, or P *A-reduces* to Q, if there exist two polynomial-time algorithms F and G and a constant α such that:

1. Let I be an instance of P. Then $F(I)$ is an instance of Q.
2. Let I be an instance of P, let S be a solution for $F(I)$. Then $G(I, S)$ is a solution for I.
3. Let I be an instance of P, and $r \geq 1$. Then, if S is an r-approximation for $F(I)$, then $G(I, S)$ is an (αr)-approximation for I.

We observe that A-reductions are transitive, that is if P A-reduces to Q, and Q A-reduces to R then P A-reduces to R.

Let P and Q be two problems. We say that P is *Q-complete with respect to A-reductions*, or *Q-complete*, if there exist A-reductions from P to Q and from Q to P. Bliznets et al. [3] showed that $(K_n - e)$-free Edge Deletion is Min Horn Deletion-complete.

Table 6 $\mathcal{S} \cup \overline{\mathcal{S}}$.

S_1		$\overline{S_1}$		S_{13}		$\overline{S_{13}}$		S_{25}		$\overline{S_{25}}$	
S_2		$\overline{S_2}$		S_{14}		$\overline{S_{14}}$		S_{26}		$\overline{S_{26}}$	
S_3		$\overline{S_3}$		S_{15}		$\overline{S_{15}}$		S_{27}		$\overline{S_{27}}$	
S_4		$\overline{S_4}$		S_{16}		$\overline{S_{16}}$		S_{28}		$\overline{S_{28}}$	
S_5		$\overline{S_5}$		S_{17}		$\overline{S_{17}}$		S_{29}		$\overline{S_{29}}$	
S_6		$\overline{S_6}$		S_{18}		$\overline{S_{18}}$		S_{30}		$\overline{S_{30}}$	
S_7		$\overline{S_7}$		S_{19}		$\overline{S_{19}}$		S_{31}		$\overline{S_{31}}$	
S_8		$\overline{S_8}$		S_{20}		$\overline{S_{20}}$		S_{32}		$\overline{S_{32}}$	
S_9		$\overline{S_9}$		S_{21}		$\overline{S_{21}}$		S_{33}		$\overline{S_{33}}$	
S_{10}		$\overline{S_{10}}$		S_{22}		$\overline{S_{22}}$		S_{34}		$\overline{S_{34}}$	
S_{11}		$\overline{S_{11}}$		S_{23}		$\overline{S_{23}}$		S_{35}		$\overline{S_{35}}$	
S_{12}		$\overline{S_{12}}$		S_{24}		$\overline{S_{24}}$		S_{36}		$\overline{S_{36}}$	

▶ **Lemma 6** ([3]). *For any $n \geq 5$, $(K_n - e)$-free Edge Deletion is Min Horn Deletion-complete with respect to A-reductions.*

We observe that H-free Edge Deletion and $\overline{H}$-free Edge Completion are equivalent, so the same result follows for $\overline{K_n - e}$-free Edge Completion. We extend the results for editing case and prove the following lemma.

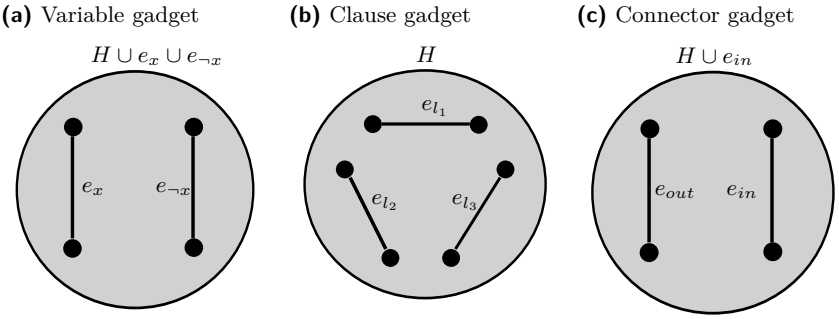
▶ **Lemma 7.** *For any $n \geq 5$, $(K_n - e)$-free Edge Editing and $\overline{K_n - e}$-free Edge Editing are Min Horn Deletion-complete with respect to A-reductions.*

As a result, at this point we can prove statements mentioned in the first three rows of Table 1. The only remaining statements of the conjecture are the ones mentioned in the last row. We observe that if P = NP then they are trivially true, so from now on, we assume that $P \neq NP$.

Table 7 $\mathcal{F} \cup \overline{\mathcal{F}}$.

$\mathcal{F}_1$	$K_{2,t}$	$\overline{\mathcal{F}_1}$	$K_t \cup K_2$	$t \geq 4$
$\mathcal{F}_2$	$K_{1,t}$	$\overline{\mathcal{F}_2}$	$K_t \cup K_1$	$t \geq 5$
$\mathcal{F}_3$	$K_2 \boxtimes tK_1$	$\overline{\mathcal{F}_3}$	$K_t \cup 2K_1$	$t \geq 4$
$\mathcal{F}_4$	$T_{t,1}$	$\overline{\mathcal{F}_4}$	$\overline{T_{t,1}}$	$t \geq 4$
$\mathcal{F}_5$	$\overline{(K_t - e) \cup 2K_1}$	$\overline{\mathcal{F}_5}$	$(K_t - e) \cup 2K_1$	$t \geq 4$
$\mathcal{F}_6$	$\overline{(K_t - e) \cup K_2}$	$\overline{\mathcal{F}_6}$	$(K_t - e) \cup K_2$	$t \geq 4$
$\mathcal{F}_7$	$K_{1,t} \cup K_2$	$\overline{\mathcal{F}_7}$	$\overline{K_{1,t} \cup K_2}$	$t \geq 4$
$\mathcal{F}_8$	$\overline{(K_t - e) \cup K_1}$	$\overline{\mathcal{F}_8}$	$(K_t - e) \cup K_1$	$t \geq 6$
$\mathcal{F}_9$	J_t	$\overline{\mathcal{F}_9}$	$\overline{J_t}$	$t \geq 3$
$\mathcal{F}_{10}$	Q_t	$\overline{\mathcal{F}_{10}}$	$\overline{Q_t}$	$t \geq 3$

Figure 1 Gadgets for Sandwich H-free Edge Deletion.



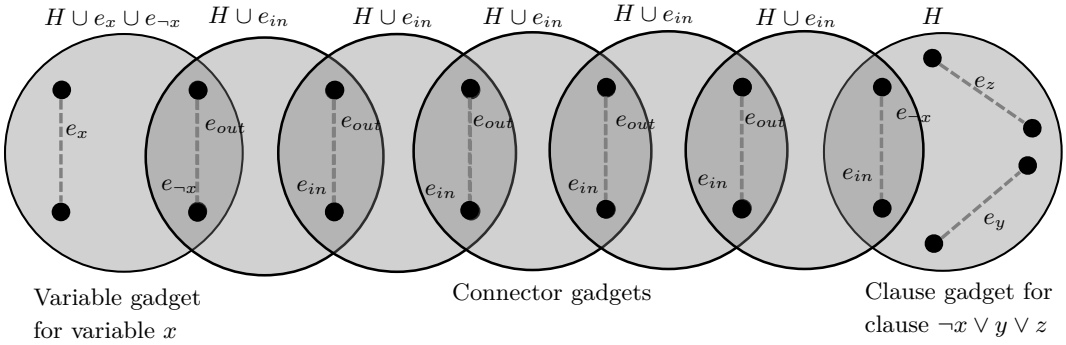
4 Problems without $poly(OPT)$-approximation

In this section, we consider several classes of graphs H such that H-free Edge Deletion (Completion, Editing) does not admit $poly(OPT)$-approximation. To prove it, we use a technique introduced in [3]. We also generalize that technique and use it to prove that H-free Edge Editing does not admit $poly(OPT)$-approximation for some graphs H.

Let us first prove NP-hardness of Sandwich H-free Edge Deletion and Sandwich H-free Edge Completion by constructing polynomial reductions from 3-SAT, which is a well-known NP-hard problem (we show those results only for H with some specific properties). In the 3-SAT problem, we are given a Boolean formula φ in CNF where each clause contains exactly 3 literals, and the task is to determine whether φ is satisfiable or not.

Let us define Variable, Clause and Connector gadgets for Sandwich H-free Edge Deletion (see Figure 1). Variable gadget for a variable x is a copy of H with two added edges e_x and $e_{\neg x}$ (so H has at least two non-edges), where only these two edges are allowed to be deleted from the graph by making all other edges undeletable. Removing the edge e_x corresponds to $x = 1$, and removing the edge $e_{\neg x}$ corresponds to $x = 0$. Clause gadget for a clause $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ is a copy of H where only three of its edges e_{ℓ_1}, e_{ℓ_2} and e_{ℓ_3} are allowed to be deleted by making all other edges undeletable (so H has at least three edges). Removing the edge e_{ℓ_i} corresponds to the case when the literal ℓ_i satisfies the clause c. Connector gadget is a copy of H with one added edge e_{in}, where e_{in} and some other edge e_{out} that does not share any endpoint with e_{in}, are allowed to be deleted, and all other edges are made undeletable. Using Connector gadgets, we connect Clause gadgets with Variable gadgets in such a way

Figure 2 An illustration of how a Connector gadget connects a Variable gadget with a Clause gadget.



that the edges corresponding to variables and their literals are removed consistently (see Figure 2). We give the exact construction below. For Sandwich H-free Edge Completion, we define those gadgets in a similar way with slight differences, for example e_{in} becomes a non-edge as it will be on a place of some deleted edge in H.

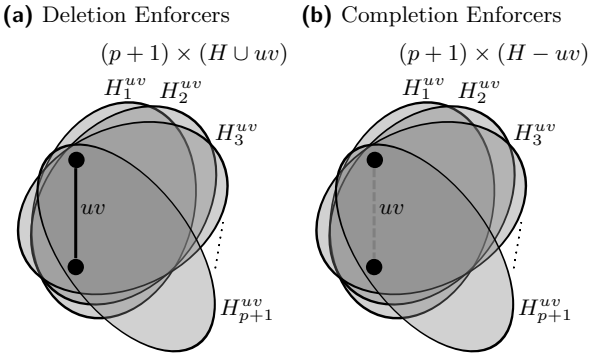
▶ **Construction 1.** *Given an instance φ of 3-SAT with an assignment τ and a set $S = \{Variable, Clause, Connector\}$ of gadgets for Sandwich H-free Edge Deletion (Completion), we construct a graph G. For each variable we introduce its own copy of Variable gadget, and for each clause we introduce its copy of Clause gadget. For each clause $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ and each literal $\ell \in c$, we introduce a chain of Connector gadgets $G_1^{\ell,c}, G_2^{\ell,c}, \ldots, G_{p+2}^{\ell,c}$ where $p = |V(H)|$. For each $1 \leq i \leq p+1$, we glue $G_i^{\ell,c}$ to $G_{i+1}^{\ell,c}$ in such a way that the edge (non-edge) e_{out} of $G_i^{\ell,c}$ coincide with the edge (non-edge) e_{in} of $G_{i+1}^{\ell,c}$. Similarly, we identify the edge (non-edge) e_{out} of $G_{p+2}^{\ell,c}$ with the edge (non-edge) e_ℓ of the Variable gadget for the variable corresponding to the literal ℓ. We also identify the edge (non-edge) e_{in} of $G_1^{\ell,c}$ with the edge (non-edge) e_ℓ of the Clause gadget for the clause c. Let G be the resulting graph, and let D be the set of undeletable edges (not fillable non-edges). Let $F \subseteq E(G)$ ($F \subseteq \overline{E}(G)$) contain all edges (non-edges) e_ℓ from Variable and Clause gadgets, where $\tau(\ell) = 1$. Also, let F contain all edges (non-edges) e_{in} and e_{out} from Connector gadgets $G_i^{\ell,c}$, where $\tau(\ell) = 1$. The result of the construction based on φ, τ and gadgets is the graph $G \triangle F$.*

We say that H *intersects* a set {Variable, Clause, Connector} of gadgets if there exist an instance φ of 3-SAT and a satisfying assignment τ for φ, so if G_φ^τ is the result of applying Construction 1 to φ, τ and {Variable, Clause, Connector}, then there exists an induced subgraph of G_φ^τ isomorphic to H, which is not fully contained within one gadget.

▶ **Lemma 8.** *Let H be a graph, let {Variable, Clause, Connector} be a set of gadgets for Sandwich H-free Edge Deletion (Completion), and let H not intersect {Variable, Clause, Connector}. Then Sandwich H-free Edge Deletion (Completion) is NP-hard.*

Now we introduce two more gadgets: Deletion Enforcer and Completion Enforcer. Deletion Enforcer for a graph H is a copy of H with one added edge, and Completion Enforcer for a graph H is a copy of H with one deleted edge. The added (deleted) edge we call *highlighted edge (non-edge)*. By using these two gadgets, for some graphs H we show that H-free Edge Deletion (Completion, Editing) does not admit $poly(OPT)$-approximation.

Figure 3 Examples of how we enforce an edge to be undeletable or a non-edge to be non-fillable.



We say that H *intersects* Deletion (Completion) Enforcer X if there exists a graph G such that if we join X and G by identifying a highlighted edge (non-edge) of X with some edge (non-edge) of G, then the resulting graph contains an induced copy of H, which contains a vertex from $V(X) \setminus V(G)$ and a vertex from $V(G) \setminus V(X)$. Gluing a large amount of Deletion (Completion) Enforcers to an instance of Sandwich H-free Edge Deletion (Completion) (see Figure 3), we can convert it into H-free Edge Deletion (Completion) in some sense. Precisely, we can prove the following lemma.

▶ **Lemma 9.** *Let H be a graph, let X be a Deletion (Completion) Enforcer for H, and let H not intersect X. Then (i) Sandwich H-free Edge Deletion (Completion) is NP-hard $\Rightarrow$ H-free Edge Deletion (Completion) does not admit poly(OPT)-approximation; (ii) H-free Edge Completion (Deletion) does not admit poly(OPT)-approximation $\Rightarrow$ H-free Edge Editing does not admit poly(OPT)-approximation.*

It was shown in [3] that H-free Edge Deletion and H-free Edge Completion do not admit *poly*(OPT)-approximation when H is a 3-connected graph with at least two non-edges or when H is a cycle with at least four vertices or a path with at least five vertices. Taking into account Lemma 3, the following lemma was shown.

▶ **Lemma 10** ([3])**.** *Let H be a 3-connected graph with at least two non-edges, or a cycle with at least four vertices or a path with at least five vertices. Then H-free Edge Deletion (Completion) and $\overline{H}$-free Edge Deletion (Completion) do not admit poly(OPT)-approximation.*

Using Lemma 9, we can prove a similar result for H-free Edge Editing.

▶ **Lemma 11.** *Let H be a 3-connected graph with at least two non-edges, or a cycle with at least four vertices, or a path with at least five vertices. Then H-free Edge Editing and $\overline{H}$-free Edge Editing do not admit poly(OPT)-approximation.*

Slightly adapting results from [20] for our needs, we obtain the following lemma.

▶ **Lemma 12.** *Let H be a regular graph which is not complete or empty. Then H-free Edge Deletion (Completion, Editing) does not admit poly(OPT)-approximation.*

5 General case

In this section, we show that to complete the proof of the conjecture it is sufficient to show that H-free Edge Deletion (Completion, Editing) does not admit *poly*(OPT)-approximation for any H from some specific set of graphs. To do this, we introduce an algorithm that, given

a graph H, constructs another graph H', such that if H-free Edge Deletion (Completion, Editing) admits $poly(OPT)$-approximation then H'-free Edge Deletion (Completion, Editing) also admits $poly(OPT)$-approximation.

Our algorithm is similar to the algorithm given by Marx and Sandeep in [20], where they studied the existence of polynomial kernels for H-free Edge Deletion (Completion, Editing). In order to present the algorithm we introduce two sets $\mathcal{X}$ and $\mathcal{Y}$.

$$\begin{aligned}\mathcal{X} &= \{C_\ell, \overline{C_\ell} \mid \ell \geq 4\}\\ &\cup \{P_\ell, \overline{P_\ell} \mid \ell \geq 5\}\\ &\cup \{H \mid H \text{ is regular but is neither complete nor empty}\}\\ &\cup \{H \mid H \text{ is 3-connected with at least two non-edges}\}\\ &\cup \{H \mid \overline{H} \text{ is 3-connected with at least two non-edges}\}\\ \mathcal{Y} &= \{K_n, \overline{K_n} \mid n \geq 5\}\\ &\cup \{K_n - e, \overline{K_n - e} \mid n \geq 5\}\\ &\cup \{H \mid H \neq C_4, \overline{H} \neq C_4, |V(H)| \leq 4\}\end{aligned}$$

Note that to finish the proof of the conjecture it is sufficient to prove that for graphs $H \notin \mathcal{Y}$ H-free Edge Deletion (Completion, Editing) does not admit $poly(OPT)$-approximation. Let us show that it is sufficient to prove that only for some subset of those graphs. To do this, we consider the following algorithm.

Churn(H) :

Step 1: if H is regular, then return H.

Step 2: if $H - V_\ell \notin \mathcal{Y}$, then return Churn($H - V_\ell$).

Step 3: if $H - V_h \notin \mathcal{Y}$, then return Churn($H - V_h$).

Step 4: return H.

▶ **Lemma 13.** *If H-free Edge Deletion (Completion, Editing) admits $poly(OPT)$-approximation, then Churn(H)-free Edge Deletion (Completion, Editing) also admits $poly(OPT)$-approximation.*

Thus, if there exists a graph $H \notin \mathcal{Y}$ such that H-free Edge Deletion (Completion, Editing) admits $poly(OPT)$-approximation, then Churn(H)-free Edge Deletion (Completion, Editing) also admits $poly(OPT)$-approximation, so it is sufficient to prove that there is no $poly(OPT)$-approximation only for graphs from the set $\{\text{Churn}(H) \mid H \notin \mathcal{Y}\}$. Our goal is to obtain a finite set of graphs for which the conjecture holds if and only if it holds for the set of all graphs with at least five vertices. In order to do that, we first consider the set $\{\text{Churn}(H) \mid H \notin \mathcal{Y}\}$, and then we gradually reduce it.

Let us call a set of graphs $\mathcal{L}$ *good for Deletion (Completion, Editing) problems*, if it does not contain graphs from $\mathcal{Y}$ and the following property holds. If H contains at least five vertices, $H \notin \mathcal{Y}$, and H-free Edge Deletion (Completion, Editing) admits $poly(OPT)$-approximation then there exists a graph $H' \in \mathcal{L}$ such that H'-free Edge Deletion (Completion, Editing) admits $poly(OPT)$-approximation and $|V(H')| \leq |V(H)|$. We call a set $\mathcal{L}$ *good*, if it is good for Deletion, Completion and Editing problems simultaneously. Note that both $\{H \mid H \notin \mathcal{Y}\}$ and $\{\text{Churn}(H) \mid H \notin \mathcal{Y}\}$ are good sets.

▶ **Lemma 14.** *Let $\mathcal{L}$ be good for Deletion (Completion, Editing) problems. If for each graph $H \in \mathcal{L}$ H-free Edge Deletion (Completion, Editing) does not admit poly(OPT)-approximation, then for each graph $H \notin \mathcal{Y}$ H-free Edge Deletion (Completion, Editing) does not admit poly(OPT)-approximation.*

Now, we want to understand which graphs form the set $\{\text{Churn}(H) \mid H \notin \mathcal{Y}\}$. First of all, it can be regular graphs. Observe, that we run Churn(H) only on graphs H that are not complete or empty, and we never obtain a complete or empty graph during the algorithm, since such a graph belongs to $\mathcal{Y}$. If we obtain a regular graph H as an output of the algorithm, then H is not complete or empty, and then $H \in \mathcal{X}$. If we obtain a graph H which is not regular, then we know that $H - V_\ell \in \mathcal{Y}$ and $H - V_h \in \mathcal{Y}$.

Let $\mathcal{W}$ be a set of graphs introduced in Section 2. In [20] (Lemmas 3.7 – 3.23), Marx and Sandeep proved the following lemma.

▶ **Lemma 15** ([20])**.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$, $H - V_\ell \in \mathcal{Y} \setminus \{K_n - e \mid n \geq 5\}$ and $H - V_h \in \mathcal{Y} \setminus \{K_n - e \mid n \geq 5\}$. Then $H \in \mathcal{W}$.*

This lemma covers almost all graphs $H \notin \mathcal{X} \cup \mathcal{Y}$ such that $H - V_\ell \in \mathcal{Y}$ and $H - V_h \in \mathcal{Y}$. It remains to understand what happens to graphs H for which $\{H - V_\ell, H - V_h\} = \{K_n - e, S\}$, where $n \geq 5$ and $S \in \mathcal{Y}$. We can show that the only case when Lemma 15 is not applicable, is when $S = \overline{K_m - e}$, where $m \geq 5$. Let us call the set of such graphs $\mathcal{U}$. We obtain the following lemma.

▶ **Lemma 16.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$, $H - V_\ell \in \mathcal{Y}$ and $H - V_h \in \mathcal{Y}$. Then $H \in \mathcal{W} \cup \mathcal{U}$.*

▶ **Corollary 17.** *Let $H \notin \mathcal{Y}$. Then Churn$(H) \in \mathcal{X} \cup \mathcal{W} \cup \mathcal{U}$.*

Using Lemma 13, Corollary 17 and the fact that problems corresponding to graphs from $\mathcal{X}$ do not admit $poly(OPT)$-approximation, we obtain the following lemma.

▶ **Lemma 18.** *$\mathcal{W} \cup \mathcal{U}$ is a good set.*

We note that at this stage in [20], $\mathcal{W}$ was shown to be a good set for kernelization lower bounds. The necessity to handle graphs from $\mathcal{U}$ is one of the key differences between approximation and kernelization cases.

From now on, we consider only graphs from $\mathcal{W} \cup \mathcal{U}$. Later we remove some graphs from this set and obtain a finite good set.

▶ **Lemma 19.** *Let $\mathcal{L}$ be a good set for Deletion (Completion, Editing) problems. If $H \in \mathcal{L}$ is a graph such that H-free Edge Deletion (Completion, Editing) does not admit poly(OPT)-approximation, then $\mathcal{L} \setminus \{H\}$ is a good set for Deletion (Completion, Editing) problems.*

▶ **Lemma 20.** *Let $\mathcal{L}$ be a good set for Deletion (Completion, Editing) problems. Let $H \in \mathcal{L}$, and let H' be such a graph that the fact that H-free Edge Deletion (Completion, Editing) admits poly(OPT)-approximation implies that H'-free Edge Deletion (Completion, Editing) admits poly(OPT)-approximation. If $H' \notin \mathcal{Y}$ and $|V(H')| < |V(H)|$ then $\mathcal{L} \setminus \{H\}$ is good for Deletion (Completion, Editing) problems.*

Let us call a graph H *uninteresting for Deletion (Completion, Editing) problems*, if either H-free Edge Deletion (Completion, Editing) does not admit $poly(OPT)$-approximation or there exists a graph $H' \notin \mathcal{Y}$ such that $|V(H')| < |V(H)|$ and the existence of $poly(OPT)$-approximation for H-free Edge Deletion (Completion, Editing) implies the existence of $poly(OPT)$-approximation for H'-free Edge Deletion (Completion, Editing). Let us call a graph H *uninteresting*, if it is uninteresting for Deletion, Completion and Editing problems simultaneously. Using Lemmas 19 and 20, we obtain the following corollary.

▶ **Corollary 21.** *Let $\mathcal{L}$ be a good set, and let H be an uninteresting graph. Then $\mathcal{L} \setminus \{H\}$ is a good set.*

Thus, we have shown that we can remove uninteresting graphs from a good set and still get a good set. Now let us show that all graphs from $\mathcal{U}$ are uninteresting and can be left out of consideration. In order to do that, we introduce some definitions.

For a fixed $n \geq 5$, a graph $G = (V, E)$ is called $K_n - e$ *with V-shapes*, if there exists a partition $V = V_1 \sqcup V_2$ such that $G[V_1]$ is isomorphic to $K_n - e$, $G[V_2]$ does not contain any edges, and each vertex in V_2 has degree 2 in G. Each vertex $v \in V_2$ we call a *peak*. A subgraph that contains v, its two neighbours and two edges incident to v we call a *V-shape*. A set of two vertices that are neighbours of the peak in a V-shape we call *base* of the V-shape.

▶ **Lemma 22.** *Let H be $K_n - e$ with V-shapes for some $n \geq 5$. Let there be two V-shapes with non-intersecting bases. Then, if Sandwich H-free Edge Deletion (Completion) is NP-hard, then H-free Edge Deletion (Completion) does not admit $poly(OPT)$-approximation.*

▶ **Lemma 23.** *Let H be $K_n - e$ with V-shapes for some $n \geq 5$. Then, if H-free Edge Deletion does not admit $poly(OPT)$-approximation, then H-free Edge Editing also does not admit $poly(OPT)$-approximation.*

Using those results, we can now prove that all graphs from $\mathcal{U}$ are uninteresting. For $H \in \mathcal{U}$, $\{H - V_\ell, H - V_h\} = \{K_n - e, \overline{K_m - q}\}$, where $n, m \geq 5$. We consider several cases depending on location of edge q and non-edge e in H, and show that all graphs from $\mathcal{U}$ are uninteresting. Then, using Corollary 21, we obtain the following lemma.

▶ **Lemma 24.** *$\mathcal{W}$ is a good set.*

Using the technique by Marx and Sandeep from [20], we show that all graphs from $\mathcal{F} \cup \overline{\mathcal{F}} \cup \mathcal{S} \cup \overline{\mathcal{S}}$ are uninteresting. Hence, we prove the following lemma.

▶ **Lemma 25.** *$\mathcal{H} \cup \overline{\mathcal{H}} \cup \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{D} \cup \overline{\mathcal{D}} \cup \mathcal{B} \cup \overline{\mathcal{B}}$ is a good set.*

Additionally, we show that H-free Edge Deletion (Completion, Editing) does not admit $poly(OPT)$-approximation for some of the remaining graphs. Our proofs for these graphs are constructed according to the scheme introduced earlier: we show that Sandwich H-free Edge Deletion (Completion) is NP-hard, then, using some Deletion (Completion) Enforcer, we show that H-free Edge Deletion (Completion) does not admit $poly(OPT)$-approximation, and then, using some Completion (Deletion) Enforcer, we show that H-free Edge Editing does not admit $poly(OPT)$-approximation.

As a result, we show that graphs A_2, $\overline{A_2}$, A_3, $\overline{A_3}$, $A_4 = \overline{A_4}$, $A_5 = \overline{A_5}$, B_1, $\overline{B_1}$, B_3, $\overline{B_3}$ are uninteresting. So, the set of graphs $\mathcal{G} \cup \overline{\mathcal{G}}$ presented in Table 2 is a good set. Observe that, by Lemma 3, if $\forall H \in \mathcal{G}$ H-free Edge Editing does not admit $poly(OPT)$-approximation, then $\forall H \in \overline{\mathcal{G}}$ H-free Edge Editing also does not admit $poly(OPT)$-approximation. So, we have proved Theorem 2.

We note that it is left to resolve the conjecture for more graphs than in the kernelization case. In [20], Marx and Sandeep also use a two step proof. They first prove that there is no polynomial kernel for Restricted H-free Edge Deletion (Completion), which is a version of Sandwich H-free Edge Deletion (Completion) where we also want to minimize the number of deleted (added) edges. Then, using similar enforcers, they give a reduction from Restricted H-free Edge Deletion (Completion) to H-free Edge Deletion (Completion, Editing). To prove that Restricted H-free Edge Deletion (Completion) does not admit a polynomial kernel, they use a reduction from Propagational-f Satisfiability on 3-regular conjunctive

formulas (every variable appears exactly three times) that does not admit a polynomial kernel. In the case of approximation, we cannot use the same trick, because Propagational-f Satisfiability on 3-regular conjunctive formulas is Min Horn Deletion-complete, so it gives us only Min Horn Deletion-hardness, and we need stronger results. That is why we use a technique from [3]. But in this case, we need to show hardness for Sandwich H-free Edge Deletion (Completion) instead of Restricted H-free Edge Deletion (Completion), and it can be harder, or even impossible. For example, Sandwich $(K_n - e)$-free Edge Deletion admits a polynomial-time solution, while Restricted $(K_n - e)$-free Edge Deletion (Completion) does not admit a polynomial kernel unless $NP \subseteq coNP/poly$ [7].

6 Conclusion

In this work, we consider the hardness of approximation for H-free Edge Deletion (Completion, Editing) and we formulate Conjecture 1, according to which all graphs on at least five vertices can be classified into several groups of graphs with specific structural properties depending on the hardness of approximation for H-free Edge Deletion (Completion, Editing) problems corresponding to them. We think that we make a significant progress in proving those conjectures by proving Theorem 2. To make the complete classification of H-free Edge Deletion (Completion, Editing) problems for graphs H on at least five vertices by the hardness of their approximation, it is now enough to determine the hardness of approximation for those problems only for graphs $H \in \mathcal{G} \cup \overline{\mathcal{G}}$, where $\mathcal{G}$ is a set of seventeen graphs. The hardness of approximation for Min Horn Deletion-complete problems and the hardness of approximation for H-free Edge Deletion (Completion, Editing) problems for graphs H on at most four vertices are still open questions.

Also, we trace the relationship between the complexity of approximation and the complexity of kernelization for H-free Edge Deletion (Completion, Editing). We have shown that, in this case, the approaches introduced to prove that a problem does not admit a polynomial kernel can be used to prove that a problem does not admit $poly(OPT)$-approximation. We believe that the complexity of approximation for H-free Edge Deletion (Completion, Editing) is closely connected with the complexity of kernelization for those problems. However, in case of the approximation complexity, the situation is more complicated, because of the problems corresponding to graphs with exactly one edge or non-edge that form a separate class. So, in the paper, we make a significant step towards resolving conjecture from [3] asking whether kernelization and approximation behave in the same way for H-free Edge Modification.

References

1 NR Aravind, RB Sandeep, and Naveen Sivadasan. Dichotomy results on the hardness of h-free edge modification problems. *SIAM Journal on Discrete Mathematics*, 31(1):542–561, 2017.

2 Ivan Bliznets, Marek Cygan, Pawel Komosa, Lukáš Mach, and Michał Pilipczuk. Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1132–1151. SIAM, 2016.

3 Ivan Bliznets, Marek Cygan, Paweł Komosa, and Michał Pilipczuk. Hardness of approximation for h-free edge modification problems. *ACM Transactions on Computation Theory (TOCT)*, 10(2):1–32, 2018.

4 Ivan Bliznets, Marek Cygan, Paweł Komosa, Michał Pilipczuk, and Lukáš Mach. Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. *ACM Transactions on Algorithms (TALG)*, 16(2):1–31, 2020.

5 Ivan Bliznets, Fedor V Fomin, Marcin Pilipczuk, and Michał Pilipczuk. Subexponential parameterized algorithm for interval completion. *ACM Transactions on Algorithms (TALG)*, 14(3):1–62, 2018.

6 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

7 Leizhen Cai and Yufei Cai. Incompressibility of h-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015.

8 Yufei Cai. *Polynomial kernelisation of H-free edge modification problems*. PhD thesis, Chinese University of Hong Kong, 2012.

9 Christophe Crespelle, Pål Grønås Drange, Fedor V Fomin, and Petr A Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv preprint*, 2020. `arXiv:2001.06867`.

10 R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006.

11 Pål Grønås Drange. *Parameterized graph modification algorithms*. PhD thesis, The University of Bergen, 2015.

12 Ehab S El-Mallah and Charles J Colbourn. The complexity of some edge deletion problems. *IEEE transactions on circuits and systems*, 35(3):354–362, 1988.

13 Fedor V Fomin, Petr A Golovach, and Dimitrios M Thilikos. On the parameterized complexity of graph modification to first-order logic properties. *Theory of Computing Systems*, 64(2):251–271, 2020.

14 Archontia C Giannopoulou, Daniel Lokshtanov, Saket Saurabh, and Ondrej Suchy. Tree deletion set has a polynomial kernel but no opt^o(1) approximation. *SIAM Journal on Discrete Mathematics*, 30(3):1371–1384, 2016.

15 Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-) existence of polynomial kernels for p l-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013.

16 Haim Kaplan, Ron Shamir, and Robert E Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.

17 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.

18 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. In *International Workshop on Parameterized and Exact Computation*, pages 264–275. Springer, 2009.

19 John M Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

20 Dániel Marx and RB Sandeep. Incompressibility of h-free edge modification problems: Towards a dichotomy. *Journal of Computer and System Sciences*, 125:25–58, 2022.

21 Mihalis Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981.

Hierarchical Categories in Colored Searching

Peyman Afshani
Aarhus University, Denmark

Rasmus Killmann
Aarhus University, Denmark

Kasper Green Larsen
Aarhus University, Denmark

Abstract

In colored range counting (CRC), the input is a set of points where each point is assigned a "color" (or a "category") and the goal is to store them in a data structure such that the number of distinct categories inside a given query range can be counted efficiently. CRC has strong motivations as it allows data structure to deal with categorical data.

However, colors (i.e., the categories) in the CRC problem do not have any internal structure, whereas this is not the case for many datasets in practice where hierarchical categories exists or where a single input belongs to multiple categories. Motivated by these, we consider variants of the problem where such structures can be represented. We define two variants of the problem called hierarchical range counting (HCC) and sub-category colored range counting (SCRC) and consider hierarchical structures that can either be a DAG or a tree. We show that the two problems on some special trees are in fact equivalent to other well-known problems in the literature. Based on these, we also give efficient data structures when the underlying hierarchy can be represented as a tree. We show a conditional lower bound for the general case when the existing hierarchy can be any DAG, through reductions from the orthogonal vectors problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Categorical Data, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.25

1 Introduction

Range searching is a broad area of computational geometry where the goal is to store a given set P of input points in a data structure such that one can efficiently report (*range reporting*) or count (*range counting*) the points inside a geometric query region $\mathcal{R}$. Sometimes, each point $p_i \in P$ is associated with a weight $w_i \in \mathbb{R}$ and the goal could be to find the sum of the weights in $\mathcal{R}$, or the maximum weight in $P \cap \mathcal{R}$ (*range max* problem). This is a very broad area of research and there are many well-studied variants. See a recent excellent survey by Agarwal for more information [1].

Colored (or *categorical*) range searching is an important variant where each input point is associated with a *category* which conceptually is represented as a color; the goal of the query is then to report or count the number of distinct colors inside the query range. Colored range searching has strong motivations since colors allow us to represent *nominal attributes* such as brand, manufacturer and so on and in practice a data set often contains a mix of nominal and ordinal attributes.

Colored range searching was introduced in 1993 by Janardan and Lopez [14] and it has received considerable attention since then. However, classical colored range searching only models a "flat" categorical structure where categories have no inherent structure. We consider variants of colored range counting to model such structures. We show that looking at colored range searching from this angle creates a number of interesting questions with non-trivial connections to other already existing problems.

1.1 Problem Definitions and Motivations

We begin by formally defining colored range counting.

▶ **Problem 1** (colored range counting). *Given an input set P of n points in $\mathbb{R}^d$, a set C of colors (i.e., categories) and a function $\mathfrak{C} : P \to C$, store them in a data structure such that given a query range $\mathcal{R} \subset \mathbb{R}^d$, it can count the number of distinct colors in $\mathcal{R}$, i.e., the value $|C_{\mathcal{R}}|$ where $C_{\mathcal{R}} = \{\mathfrak{C}(p) \mid p \in P \cap \mathcal{R}\}$.*

In the weighted version, input also includes a weight function $\mathcal{W} : C \to \mathbb{R}$ and the output of the query is the weighted sum of the distinct colors in $\mathcal{R}$, i.e., the value $\sum_{c \in C_{\mathcal{R}}} \mathcal{W}(c)$.

Colored range searching assumes that the colors are completely unstructured and that each point receives exactly one color. However, these assumptions can be inadequate as hierarchical categories are quite common. For example, biological classification of living organisms are done through a tree where inclusion in a category implies inclusion in all the ancestor categories. In fact, similar phenomena happen with respect to most notions of classification (e.g., classification of industries). In other scenarios, a point may have multiple categories (e.g., a car can have a "brand", a "color", "fuel type" and so on). While it is possible to view the set of categories assigned to a point as one single category, doing so ignores a lot of structure. For example, "a blue diesel car" is both a "blue car" and also a "diesel car" but by considering "blue diesel" as a single category, this information is lost. We believe it is worthwhile to study the notion of structures within categories from a theoretical point of view. We are not in fact the first in trying to do so and we will shortly discuss some of the previous attempts.

A natural way to represent hierarchical categories is to assume that vertices of a DAG $\mathcal{G}$ represent the set of categories where an edge $\vec{e} = (u, v)$ from (a category) u to (a category) v means that u is a sub-category of v. We call $\mathcal{G}$ a *category DAG* (or a *category tree* if it is a tree). For a vertex $v \in \mathcal{G}$, we define $\mathcal{G}_{\leq}(v)$ as the subset of vertices of $\mathcal{G}$ that can reach v (i.e., "sub-categories" of v), $\mathcal{G}_{\geq}(v)$ as the subset of vertices of $\mathcal{G}$ that v can reach (i.e., "super-categories" of v). Similarly, for a subset $H \subset V(G)$ we define $\mathcal{G}_{\geq}(H) = \cup_{v \in H} \mathcal{G}_{\geq}(v)$, and $\mathcal{G}_{\leq}(H) = \cup_{v \in H} \mathcal{G}_{\leq}(v)$.

Category trees allows us to represent hierarchical categories. Category DAGs allow us to capture cases where points can have multiple categories. Consider the car example again. We can define a category DAG $\mathcal{G}$ where a vertex $u \in \mathcal{G}$ represents the compound category $\{\text{diesel}, \text{blue}\}$ with edges to vertices d and b that represent "diesel" and "blue" categories respectively. Thus, the set $\mathcal{G}_{\leq}(d)$ represents all the "diesel" cars and it includes the category u, the "blue diesel" category, and similarly, the set $\mathcal{G}_{\leq}(b)$ represents all the "blue" cars which also includes the category u, the "blue diesel" category. Likewise, $\mathcal{G}_{\geq}(u)$ includes both b and d since a "blue diesel" car is both a "blue car" and a "diesel car".

We revisit colored range searching problems, using concepts of category DAGs or trees.

▶ **Problem 2** (sub-category range counting (SCRC)). *Consider an input point set $P \subset \mathbb{R}^d$ of n points, a DAG $\mathcal{G}$ with $O(n)$ edges, and a function $\mathfrak{C} : P \to \mathcal{G}$. The goal is to store the input in a data structure, such that given a query that consists of a query range $\mathcal{R} \subset \mathbb{R}^d$ and a query vertex $v_q \in \mathcal{G}$ it can output $|C_{\mathcal{R}} \cap \mathcal{G}_{\leq}(v_q)|$ where $C_{\mathcal{R}} = \{\mathfrak{C}(p) \mid p \in P \cap \mathcal{R}\}$.*

▶ **Problem 3** (hierarchical color counting (HCC)). *Consider an input point set $P \subset \mathbb{R}^d$ of n points, a DAG $\mathcal{G}$ with $O(n)$ edges, and a function $\mathfrak{C} : P \to \mathcal{G}$. The goal is to store the input in a data structure, such that given a query range $\mathcal{R} \subset \mathbb{R}^d$ one can output $|G_{\mathcal{R}}|$ where $G_{\mathcal{R}}$ is the set of colors in $\mathcal{R}$, i.e., $G_{\mathcal{R}} = \bigcup_{p \in \mathcal{R} \cap P} \mathcal{G}_{\geq}(\mathfrak{C}(p))$.*

In the weighted version of the problem, each vertex v (i.e., category) of $\mathcal{G}$ is associated with a weight $w(v)$ and given the query $\mathcal{R}$, the goal is to compute $\sum_{v \in G_{\mathcal{R}}} w(v)$ instead.

Related problems. Very recently, there have been other attempts to consider the structure of "colors" within the computational geometry community, e.g., He and Kazi [12] consider a problem very similar to SCRC on a tree $\mathcal{G}$; the only difference is that instead of $|C_{\mathcal{R}} \cap \mathcal{G}_{\leq}(v)|$, the query outputs $|C_{\mathcal{R}} \cap \pi(v, w)|$ where $\pi(v, w)$ is a path between two query vertices $v, w \in \mathcal{G}$. There are also other variants available. See [12] for further references.

1.2 Previous and Other Related Results

The study of colored range counting and its variants began in 1993 [14] and since then it has received considerable attention. See the survey on colored range searching and its variants [9]. The problem has at least three main variants: color range counting (CRC), color range reporting, and "type 2" color range counting (for every distinct color, count how many times it appears). Here, we only review colored range counting results.

In 1D, one can solve the CRC problem using $O(n)$ space and $O(\log n)$ query time by an elegant and simple transformation [10] which turns the problem into the unweighted 2D orthogonal range counting problem which can be solved within said bounds [5]. Interestingly, it is also possible to show an equivalence between the two problems [16]. The problem, however, is difficult for 2D and beyond. Kaplan et al. [15] showed that answering m queries on a set of n points requires $\Omega(n^{\omega/2-o(1)})$ time where ω is the boolean matrix multiplication exponent. Under some assumptions (e.g., the boolean matrix multiplication conjectures), this shows that $P(n) + mQ(n) \geq n^{3/2-o(1)}$ where $P(\cdot)$ and $Q(\cdot)$ are the preprocessing time and the query time of any data structure that solves the 2D CRC problem.

Note that the equivalence between 1D CRC and 2D range counting also applies to the weighted case of both problems, however, the status of the weighted 2D range counting is still unresolved. It can be solved with $O(n \log n / \log \log n)$ space and $O(\log n / \log \log n)$ query time [13] but it is not known if both space and query time can be improved simultaneously (it is possible to improve one at the expense of the other). The only available lower bound is a query time lower bound of $\Omega(\log n / \log \log n)$ [17].

Some other interesting problems related to our results are defined below.

▶ **Definition 1.** *The following problems are defined for an input that consists of a set $P \subset \mathbb{R}^d$ of n points. The goal is to build a data structure to answer the following queries.*

- *(orthogonal range counting) Given a query axis-aligned rectangle $\mathcal{R}$, count the number of points in $\mathcal{R}$. In the weighted version, the points have weights and the goal is to compute the sum of the weights in the query.*
- *(dominance range counting) This is a special case of orthogonal range counting where the query rectangle has the form $(-\infty, q_1] \times \cdots (-\infty, q_d]$ which is also known as a dominance range. Orthogonal range counting and dominance range counting are known to be equivalent if subtraction of weights are allowed (e.g., integer weights).*
- *(3-sided color counting). The input is in the plane ($d = 2$) and each point is assigned a color from a set C and the query is a 3-sided range in the form of $\mathcal{R} = [q_\ell, q_r] \times (-\infty, q_t]$ and the goal is to count the number of colors in $\mathcal{R}$. In the weighted version, the colors have weights and the goal is to compute the sum of the weights of the colors.*
- *(3-sided distinct coordinate counting) This is a special case of 3-sided color counting where an input point (x_i, y_i) has color y_i; in other words, given the query $\mathcal{R} = [q_\ell, q_r] \times (-\infty, q_t]$, we would like to count the number of distinct Y-coordinates inside it.*
- *(range max) Given a weight function $\mathcal{W} : P \to \mathbb{R}$ as part of the input, at the query time we would like to find the maximum weight inside a given query range $\mathcal{R}$.*

- *(sum-max color counting) This a combination of range max and color counting queries. Assume the points in P have been assigned colors from a set C and assume we have a weight function $\mathcal{W} : P \to \mathbb{R}$ on the points. Given a query range $\mathcal{R}$, we would like to compute the output $\sum_{c \in C} X_c(\mathcal{R})$ where $X_c(\mathcal{R})$ is the maximum weight of a point with color c inside $\mathcal{R}$; if no point of color c exists in $\mathcal{R}$, then $X_c(\mathcal{R}) = 0$.*
  A sum-max color counting query includes a number of the above problems as its special case: If all points have the same weight, then it reduces to a color counting query. If all points have the same color, then it reduces to a range max query. If all points have distinct colors, then it reduces to a weighted range counting query.

1.3 Our Results

Clearly, sub-category range counting (SCRC) is at least as hard as CRC. We also observe that hierarchical color counting (HCC) is also at least as hard. Thus, getting efficient results for $d \geq 2$ seems hopeless. Consequently, we focus on the 1D problem but for two different important DAG's: when $\mathcal{G}$ is a tree and also when $\mathcal{G}$ is an arbitrary (sparse) DAG. Our main results are the following.

For the SCRC problem, first, we observe that the following problems are equivalent:

1. SCRC when $\mathcal{G}$ is a single path on a one-dimensional input P.
2. 3-sided distinct coordinate counting (for a planar point set P).
3. 3-sided color counting (for a planar point set P).
4. 3D dominance color counting (for a 3D point set P).
5. 3D dominance counting (for a 3D point set P).

We start by observing that (1) and (2) are equivalent. It is also clear that (2) reduces to (3); the reduction from (3) to (4) is standard by mapping a 2D input point (x_i, y_i) to the 3D point $(-x_i, y_i, x_i)$ and the 3-sided query range $\mathcal{R} = [q_\ell, q_r] \times (-\infty, q_t]$ to the 3D dominance range $(-\infty, -q_\ell] \times (-\infty, q_t] \times (-\infty, q_x]$. The reduction from (4) to (5) was shown by Saladi [18]. We complete the loop by observing that (5) in turn reduces to (2). Note that the weighted versions of the problems are also equivalent by following the same reductions. Next, we show that SCRC can be solved using $O(n \log^2 n / \log\log n)$ space and with query time of $O(\log n / \log\log n)$ on any category tree $\mathcal{G}$; our query time is optimal which follows from the above reductions and using known results [17].

For the HCC problem on trees, we show that (weighted) HCC in 1D can be solved using a 2D (weighted) range counting data structure on $O(n \log n)$ points. For example, this yields a solution with $O(n \log^2 n / \log\log n)$ space and with $O(\log n / \log\log n)$ query time. Interestingly, we show that the following problems are in fact equivalent:

- Unweighted HCC in 1D when $\mathcal{G}$ is a (generalized) caterpillar graph.
- Weighted 2D range counting with $\Theta(\log n)$ bit long integer weights.
- 1D Colored range sum-max.

These reductions are quite non-trivial and they show a surprising equivalence between an unweighted problem (HCC) and the weighted 2D range counting. This allows us to directly apply known lower bounds or barriers for 2D range counting. First, there is an $\Omega(\log n / \log\log n)$ lower bound [17] for 2D range counting with near-linear space ($O(n \log^{O(1)} n)$ space) and by the above reductions, the same bound also applies to unweighted HCC in 1D.

When $\mathcal{G}$ can be any arbitrary sparse DAG, the problems become more complicated. By a reduction from the orthogonal vectors problem, we show that we must either have $\Omega(n^{2-o(1)})$ preprocessing time or the query time must be almost linear $\Omega(n^{1-o(1)})$ and this holds for both SCRC and HCC. Surprisingly, for the HCC problem, we can build a data structure

that has $O(\log n)$ query time using $\tilde{O}(n^{3/2})$ space, even though the data structure requires $\tilde{O}(n^2)$ preprocessing time. This is one of the rare instances where there is a polynomial gap between the space complexity and the preprocessing time of a data structure.

2 Technical preliminaries

A fundamental technique to decompose trees into paths with certain properties is called the *heavy path decomposition*. The technique was originally used as part of the amortized analysis of the link/cut trees introduced by Sleator and Tarjan [19] and later used in the data structure construction for lowest common ancestor by Harel and Tarjan [11]. The decomposition is simple and gives us the following properties.

▶ **Theorem 2.** *Given a tree T of size $O(n)$, we can partition the (vertices of the) tree into a set of paths $\pi_1, \pi_2, \ldots \pi_h$ such that on any root to leaf path in T, the number of different paths encountered is $O(\log(n))$.*

We study the HCC and SCRC problem in one dimension. First, we consider when $\mathcal{G}$ is a tree in Section 3 and then we consider the general case where $\mathcal{G}$ can be any (sparse) DAG in Section 4. The general case is more difficult to solve and we will show that through a reduction (a "conditional lower bound"). Our reduction relies on the Orthogonal Vectors conjecture which is implied by the Strong Exponential Time Hypothesis (SETH) [20].

▶ **Hypothesis** (Orthogonal vectors conjecture). *Given two sets A and B each containing n boolean vectors of dimension $d = \log^{O(1)}(n)$, deciding whether there exists two orthogonal vectors $a_i \in A$ and $b_j \in B$ cannot be done faster than $n^{2-o(1)}$ time.*

Assuming this conjecture, many near optimal time lower bounds have been proven within P, including Edit Distance, Longest Common Subsequence and Fréchet distance [3, 7, 4].

Finally, we say that a binary tree T is a generalized caterpillar tree if all the degree three vertices lie on the same path (a caterpillar tree is typically defined as the legs having size 1).

3 Hierarchical Color Counting on Trees

In the appendix (Section A), we observe that HCC is at least as hard as CRC, using a simple reduction. As a result, we focus on the 1D case. We start off by presenting a data structure to solve 1D HCC on a tree $\mathcal{G}$ and then show that unweighted HCC on generalized caterpillars is actually equivalent to weighted 2D dominance counting (up to constant factors). This allows us to conclude that HCC on generalized caterpillar graphs cannot be solved with $o(n \log n / \log\log n)$ space and $o(\log n / \log\log n)$ query time, unless the state-of-the-art on weighted 2D dominance counting can be improved.

3.1 A Data Structure

We will now focus on the 1D HCC problem on trees, as described in Problem 3. Our main result is the following.

▶ **Theorem 3.** *The HCC problem on a tree (both weighted or unweighted) in $\mathbb{R}$ can be solved using $S(n) = O(n \log^2(n) / \log\log n)$ space and $Q(n) = O(\log(n) / \log\log n)$ query time.*

We prove the above theorem in two steps, using the following lemma.

▶ **Lemma 4.**

(i) *The HCC problem in* $\mathbb{R}$ *can be reduced to* $O(\log n)$ *sub-problems of the sum-max problem in* $\mathbb{R}$ *on* n *points each.*

(ii) *A sum-max problem on* n *points can be reduced to a (weighted) 2D orthogonal range counting problem on* n *points.*

Proof. Our approach starts by looking at the underlying tree structure in $\mathcal{G}$. We split $\mathcal{G}$ into its heavy path decomposition. To prove part (i) of the lemma, we actually need to look at the specific details of the heavy-path decomposition which can be described as follows. Start from the root of $\mathcal{G}$ and follow a path to a leaf of $\mathcal{G}$ where at every node u of $\mathcal{G}$, we always descend to a child of u that has the largest subtree; this easily yields the property that after removal of π, $\mathcal{G}$ will be decomposed into a number of forests where each forest is at most half the size of $\mathcal{G}$. Then the heavy-path decomposition is built recursively, by recursing on every resulting forest. It is easily seen that the depth of the recursion is $O(\log n)$.

Let $\diamond_i$ be the set of paths obtained at the i'th-level of the recursion. An important observation here is that the paths in $\diamond_i$ are "independent" meaning, no vertex u of a path $\pi \in \diamond_i$ is a descendent or ancestor of a vertex v of a different path $\pi' \in \diamond_i$. As a result, we claim it is sufficient to solve the HCC problem on the paths $\diamond_i$, for every $i = 1, \cdots, O(\log n)$, and then sum up the $O(\log n)$ results; the set of paths in $\diamond_i$ defines the i-th sub-problem and thus it remains to show how it can be reduced to a sum-max problem.

We now build an instance of the sum-max problem on $\diamond_i$: we use the same input set P but with different colors and also the points will receive weights, as follows. For every path in $\diamond_i$, we define a new color class, i.e., for the set C in the definition of the sum-max problem we have $|C| = |\diamond_i|$. Let c be the (original) color of a point p in the HCC problem (i.e., in graph $\mathcal{G}$). Consider the position of the color c in $\mathcal{G}$ and the path π_c that connects it to the root of $\mathcal{G}$. Let $\pi_j \in \diamond_i$ be the path that intersects π_c (if there's no such path, p is not stored in the i-th subproblem) on a vertex v. The weight of p will be a prefix sum of the weights in π_j: we start from the root of π_j and add the weights all the way to v. Note that an unweighted HCC can be thought of as a weighted instance of HCC with weights one. By the definition of the sum-max problem, the answer to a sum-max query will yield the number of vertices (or the total weights of the vertices) of the paths in $\diamond_i$ that need to be counted in the HCC problem. This concludes the proof of part (i) of the lemma.

To prove part (ii), we transform each point into an orthogonal range in 2D, inspired by the previous solutions to 1D color counting. Consider an instance of a sum-max problem where the x-coordinate of a point p is p_x, its color is $c(p)$ and its weight is $w(p)$. For a point $p^{(i)}$, denote the first point of the same color and greater weight to its left (resp. right) with $p^{(\ell)}$ (resp. $p^{(r)}$). Observe $p^{(i)}$ is only counted by a sum-max query I if we have both $p_x^{(i)} \in I$ and $I \subset (p^{(\ell)}, p^{(r)})$. Based on this observation, we associate to $p^{(i)}$ the two dimensional region $(p_x^{(\ell)}, p_x^{(i)}] \times [p_x^{(i)}, p_x^{(r)})$ (If $p^{(\ell)}$ or $p^{(r)}$ does not exist, make the region unbounded in the corresponding direction). We do this transformation for all points in our input. For a query range $I = [I_1, I_2]$ we map it to the point $q = (I_1, I_2)$; by our observation, q precisely stabs the rectangles which correspond to the heaviest points of every color that lies inside I.

Thus, we have reduced the problem to the following: our input is a set of axis-aligned rectangles where each rectangle is assigned a weight and given a query point $q = (q.x, q.y)$, the goal is to sum up the weights of the rectangles that contain q, a.k.a, an instance of the rectangle stabbing problem. By a simple known reduction, this problem reduces to 2D orthogonal range counting: simply turn an input rectangle $[x_1, y_1] \times [x_2, y_2]$ where $x_1 < x_2, y_1 < y_2$ and with weight w into four points: points (x_1, y_1) and (x_2, y_2) with weight w and points (x_1, y_2) and (x_2, y_1) with weight $-w$. Computing the answer to the dominance query with point $(q.x, q.y)$ will count w only when q is inside the rectangle as otherwise the weights w and $-w$ will cancel each other out. ◀

Theorem 3 follows easily from Lemma 4 since we only need $O(\log n)$ sum-max data structures on $O(n)$ points; each reduces to weighted orthogonal range counting and the final observation is that we can combine all of the $O(\log n)$ data structures in one 2D orthogonal range counting data structure on $O(n \log n)$ points. Using known results, this can be solved with $O(n \log^2(n)/\log\log n)$ space and $O(\log(n)/\log\log n)$ query time [6] although other trade-offs are also possible. For example, by plugging in other known results for weighted 2D range counting, we can also obtain $O(n \log n)$ space and $O(\log^{2+\varepsilon} n)$ query time, for any constant $\varepsilon > 0$.

3.2 Lower Bounds and Equivalence

Here we will look at equivalent problems to the HCC problem in 1D. Aside from showing that this problem has interesting and non-trivial connections to other problems, the results in this section imply an $\Omega(\log n/\log\log n)$ query lower bound for our problem which shows that the query time of our data structure from the previous section is optimal.

▶ **Theorem 5.** *The following problems defined on an input set P of size n are equivalent, up to a constant factor blow up in space and query time and potentially an additive term in the query time for answering predecessor queries.*

- *[P1]: Unweighted HCC on a generalized caterpillar of size $O(n)$ in 1D.*
- *[P2]: The sum-max problem with $O(\log n)$ bit long integer weights in 1D.*
- *[P3]: 2D orthogonal range counting with $O(\log n)$ bits long integer weights.*

Proof. The argument in the previous section shows that P1 reduces to P2 since in a generalized caterpillar, there are only two levels in the heavy-path decomposition so there is no blow up of a $\log n$ factor in the space complexity. P2 in 1D reduces to P3 using standard techniques, using the same transformation from 1D color counting to 2D range counting [10]. The non-trivial direction is to reduce P3 to P1. We do this in a step-by-step fashion.

▷ Claim 6. P3 can be reduced to $O(1/\varepsilon)$ orthogonal range counting problems on $\varepsilon \log n$ bit-long integer weights, for any constant $\varepsilon > 0$.

Proof. Let $X = 2^{\varepsilon \log n}$. Given a weighted point set P for P3, store the weights modulo X in a structure for $\varepsilon \log n$ bit weights. Now, we can strip away the $\varepsilon \log n$ least significant bits of the original weights and repeat this process $1/\varepsilon$ times to prove the claim. ◁

▷ Claim 7. For any constant s, P3 can be reduced to $O(n^{1-2^{-s}})$ sub-problems of P3 on instances of size $O(n^{2^{-s}})$ where a query in the original problem can be reduced to $O(1)$ queries on some of the sub-problems.

Proof. We adapt the grid method by Alstrup et al. [2]. Observe that as weights are integers, summing up the weights inside a query rectangle can be done using additions and subtractions of four dominance queries of the form $(-\infty, x_1] \times (-\infty, x_2]$.

Build a $\sqrt{n} \times \sqrt{n}$ grid such that each row and column contains $\sqrt{n}$ input points. Then, use a $\sqrt{n} \times \sqrt{n}$ table T to store partial sums, as follows: the cell (i, j) of T stores the sum of all the weights in grid cells (i', j') with $i' \leq i$ and $j' \leq j$. After storing T, we recurse on the set of points stored in each row as well as each column and stop the recursion at depth s. At every sub-problem at depth s of the recursion, we have $O(n^{2^{-s}})$ points left and they become the claimed sub-problems in the lemma.

To bound the number of sub-problems, observe that if a sub-problem at depth i has m points, then it creates $2\sqrt{m}$ problems (one for every row and column) involving $\sqrt{m}$ points each in depth $i + 1$. By unrolling the recursion, we can see that at depth s of the recursion, we will have $2^s n^{1-2^{-s}} = O(n^{1-2^{-s}})$ sub-problems since s is a constant, as claimed.

Now consider a query $q = (-\infty, q_x] \times (-\infty, q_y]$. Observe that by using two predecessor queries, we can find the grid cells g that contains the query point. Assume g corresponds to the cell (i, j) in the table T and consider the cell $(i-1, j-1)$. We have stored the sum of all the weights in the cells (i', j') with $i' < i$ and $j' < j$. This value gives us the total sum of the weights in the grid cells that are completely contained in q. Next, q can be decomposed into two two queries, one in a row containing the (q_x, q_y) point and another one in the column containing the same point. Furthermore, the two queries can be made disjoint by having the cell (i, j) included in only one of them. These queries can then be answered recursively until we reach the s-th level of the recursion. Thus, in total we will need to answer $2^s = O(1)$ queries. ◁

▷ Claim 8. After performing the reductions in Claims 6 and 7, P3 can be reduced to an instance of P2 where there are at most n^ε colors and where the maximum weight is at most n^ε, for any constant $\varepsilon > 0$.

Proof. Pick s in Claim 7 such that each sub-problem has at most $0.5n^\varepsilon$ points. Consider one such sub-problem involving m points. We do a reduction inspired by Larsen and Walderveen [16]. Consider an input point $p = (x_i, y_i)$ with weight $w(p)$. p will be mapped to two points $(-x_i)$ and (y_i). They are first stored in a sum-max data structure with weight $w(p)$ and color i. They are also stored in a second sum-max data structure with weights $w(p)$ but with different colors of $2i$ and $2i + 1$.

Now, given a query range $q = (-\infty, q_x] \times (-\infty, q_y]$, we query both sum-max data structures with interval $[-q_x, q_y]$ and then subtract their results. If the point p is inside q, the first data structure counts $w(p)$ once but the second data structure counts them twice and thus their subtraction includes $w(p)$ only once. If p is not inside q, none of the data structures counts $w(p)$ or both counts $w(p)$ and those they cancel out in the subtraction.

Finally, note that the total number of colors is at most $2m \leq n^\varepsilon$. ◁

▷ Claim 9. An instance of P2 where there are at most n^ε colors and where the maximum weight is at most n^ε, for any constant $0.5 > \varepsilon > 0$, can be reduced to P1 on a generalized caterpillar of size $O(n)$.

Proof. We build a caterpillar graph $\mathcal{G}$ with a central path of length n^ε. Then, we attach a path of length n^ε to every vertex on the central path; call these attached paths, *legs*. The total size of the caterpillar graph is at most $n^{2\varepsilon} \leq n$.

Consider an input point p, with color i and weight $w \leq n^\varepsilon$. In our HCC problem, we assign it a color that corresponds to the w-th vertex of the i-th leg. Now, observe that given a query $I = [I_1, I_2]$ to the sum-max problem, asking the same query on $\mathcal{G}$ will produce the answer to the sum-max problem: all the vertices on a leg have the same color which is different from the color of all the other legs. Furthermore, at every leg we simply need to find its lowest vertex that is contained in the query range which is equivalent to finding the point of maximum weight in the same color class. Counting the number of vertices on the central path is equivalent to a range max query which is a special case of sum-max queries. ◁

Observe that the proof follows directly using the above claims. Note that at each step, we might blow up the query time and the space by a constant factor. Also, Claim 7 requires a constant number of predecessor queries. Depending on the assumptions on the coordinates of the points this can take a varying amount of time but it is dominated by the actual cost of answering the range counting queries in any reasonable model of computation. ◀

As a consequence of the above equivalence, we can get a number of conditional lower bounds for HCC queries on trees.

▶ **Corollary 10.** *The barrier of $S(n)Q(n) = \Omega(n(\log n/\log\log n)^2)$ for weighted 2D range counting data structure also applies to unweighted HCC queries, even for graphs as simple as generalized caterpillar graphs. The query lower bound of $\Omega(\log n/\log\log n)$ also applies to the HCC problem. Here $S(n)$ and $Q(n)$ refer to the space and query complexities.*

Finally we remark that if the graph $\mathcal{G}$ is a path, then the HCC problem simply reduces to the range max queries which do have more efficient solutions (e.g., with $O(n)$ space and $O(1)$ query time [8] plus $O(1)$ predecessor queries). And thus, caterpillar graphs are among the simplest graphs on which the above reduction is possible.

4 General Hierarchical Color Counting Queries

Now we shift our attention to the problem where the underlying graph $\mathcal{G}$ is a directed acyclic graph (DAG). This variant is clearly more complicated than the tree variant. However, we observe a very curious behavior, namely, the preprocessing bound is much higher than the space complexity. We show a conditional lower bound on the preprocessing time using a reduction from the orthogonal vectors problem.

4.1 A Reduction from Orthogonal Vectors

We will reduce the Orthogonal Vectors problem to the 1D HCC problem.

▶ **Theorem 11.** *Assuming the Orthogonal Vectors conjecture, any solution to the 1D hierarchical color counting problem on a DAG using $P(n)$ preprocessing time and $Q(n)$ query time, must obey $P(n) + nQ(n) \geq n^{2-o(1)}$.*

Note that in the HCC problem, a query time of $O(|\mathcal{G}|)$ is trivial by simple graph traversal methods. As a result, the above reduction shows that any non-trivial solution (besides $n^{o(1)}$ factor improvements) must have a large preprocessing time.

Proof. Let $\eta = \frac{n}{\log^c(n)}$ for a large enough constant c. We build an instance of HCC with η points, and a DAG $\mathcal{G}$ with $O(\eta)$ vertices but with $O(n)$ edges. We reduce the orthogonal vectors problem on η vectors of dimension $\log^c n$ to this instance of HCC, notice that $n^{2-o(1)} = \eta^{2-o(1)}$.

Given two sets A and B of η boolean vectors in $\{0,1\}^d$, we will construct the following DAG $\mathcal{G}$. $\mathcal{G}$ will have three layers. For each vector in A create a vertex (i.e., a category) in what we denote the first layer. Now for each of the d coordinates of the vectors create a vertex in the second layer. Lastly create a vertex in the third layer for each vector in B.

Create the following edges: For a vertex corresponding to vector a_i in the first layer create an outgoing edge to all coordinate vertices in the second layer in which a_i has a one at that corresponding coordinate. Then, for a coordinate vertex in the second layer create an outgoing edge to all vectors in the third layer where the corresponding vector in B has a one at that coordinate. This clearly takes $O(\eta d) = O(n)$ to construct (see Figure 1).

To figure out whether there exists a vector $a \in A$ and a vector $b \in B$ such that a and b are orthogonal, we do the following. Create a point in $\mathbb{R}$ for each of the vertices in the third layer; their locations do not matter as long as they are distinct. We use n queries by simply querying each point individually and thus each query interval has just one point inside it!

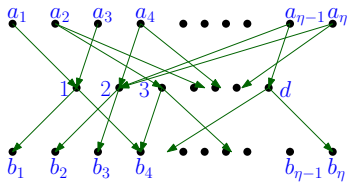


Figure 1 The underlying hierarchy DAG in the Orthogonal Vectors reduction.

▷ Claim 12. Consider a HCC query that contains the point p_i that corresponds to a vector $b_i \in B$. There is no vector in A that is orthogonal to b_i, if and only if the output size is $|b_i|_1 + 1 + \eta$ where $|b_i|_1$ is the number of ones in vector b_i.

Proof. First, let us consider the case when there is no vector in A that is orthogonal to b_i. Consider an arbitrary vector $a_j \in A$. Since a_j is not orthogonal to b_i, there exists a coordinate k where both b_i and a_j have a 1 at that coordinate. This implies that a_j is connected to b_i via the k-th vertex in the middle. As a result, all vectors in A are ancestors of b_i and since b_i is connected to $|b_i|_1$ vertices in the middle, the output of the HCC query will be as claimed.

The converse also follows by a similar argument. If a vector $a_j \in A$ does not share a 1 coordinate with b_i, then this corresponds to one of the vertices in the top layer not being counted by the query, hence the output is less than $|b_i|_1 + 1 + \eta$. ◁

Since one can easily store the values $|b_i|_1$ in $O(\eta)$ space, we can solve the Orthogonal Vectors problem using η queries on a solution for the HCC on the aforementioned DAG. The DAG has size $O(\eta d) = O(n)$ and thus we obtain the lower bound $P(n)+nQ(n) \geq \eta^{2-o(1)} = n^{2-o(1)}$. ◀

4.2 A Data Structure for General DAGs for HCC

Despite the lower bound in the previous section, it is possible to give a non-trivial data structure for HCC queries on a general DAG, however, our goal is to reduce the space complexity rather than the preprocessing time. Surprisingly, this is possible and in fact we can achieve a substantial improvement in space complexity.

▶ **Theorem 13.** *It is possible to solve the HCC problem (unweighted/weighted) on $O(n)$ points in $\mathbb{R}$ on a DAG $\mathcal{G}$ of size n using $\tilde{O}(n^2)$ preprocessing time, $\tilde{O}(n^{3/2})$ space, $O(\log(n))$ query time for unweighted and $O(\log\log(n))$ query time for weighted.*

Proof. We start by remarking that we cannot hope to reduce the preprocessing time, as shown by our conditional lower bound, however and rather surprisingly, we show that the space can be reduced to $\tilde{O}(n^{3/2})$.

Assume the input coordinates have been reduced to rank space (i.e., between 1 and n). Let $I_q = (i, j)$ be the query interval. Observe that we can afford to store the answer explicitly when $|I_q| \leq \sqrt{n}$ (i.e., "short" queries) since the number of such queries is $O(n^{3/2})$. This allows us to answer such queries in constant time. The subtle challenge, however, is to do it within $\tilde{O}(n^2)$ preprocessing time as the obvious solution could take much longer.

We start by computing the transitive closure $\mathcal{G}^c$ of $\mathcal{G}$, which takes $\tilde{O}(n^2)$ time. For a point p_i denote by c_i its color in $\mathcal{G}$ and let d_i be the number of parents of p_i in the transitive closure $\mathcal{G}^c$. We create d_i copies of the point p_i at the same position as p_i and assign each a unique parent of p_i as color. The end result will be a set of $O(n^2)$ points such that every point has a unique color and such that computing the number of colors in an interval $I = [I_1, I_2]$ will

yield the answer to the hierarchical query with the same interval. This essentially gives a "flat" representation of the hierarchical color structure, and consequently, using the existing solutions for CRC queries, we can compute the answer to all the short queries in $\tilde{O}(n^{3/2})$ time and store the results in a table. Thus, short queries can be answered in constant time, using $\tilde{O}(n^{3/2})$ space and $\tilde{O}(n^2)$ preprocessing time.

It remains to show how to deal with the long queries. Note that we can repeat the above process for all the queries to obtain a "flat representation" but doing so will yield a $\tilde{O}(n^2)$ space complexity. The key idea here is that we can "compress" the flat representation down to $O(n^{3/2})$ space, as follows. Keep in mind that the compressed representation only needs to deal with queries I_q such that $|I_q| \geq \sqrt{n}$. Partition the set of original n points into $2\sqrt{n}$ subsets of $\sqrt{n}/2$ consecutive points. For every subset P_i and every color class c (in the flat representation), delete all the points of color c in P_i except for the points in the smallest and the largest position. This will leave at most two points of color c in each subset and thus there will be at most $O(\sqrt{n})$ points of color c in all subsets. Over n colors this yields $O(n^{3/2})$ points. We store them in a data structure for (weighted or unweighted) CRC queries and this will take $\tilde{O}(n^{3/2})$ space.

The claim is that the compressed representation answers queries correctly. Consider a query interval I_q of size at least $\sqrt{n}$ and assume to the contrary that there used to be a point p of color c inside I_q in a subset P_i but p got deleted during the compression step. However, we have kept the rightmost point, p_1, and the leftmost point, p_2, of color c inside P_i. Now, observe that since P_i contains at most $\sqrt{n}/2$ points, either its rightmost point or its leftmost point (or both) must be inside I_q. As a result, either p_1 or p_2 must be inside I_q, a contradiction. The preprocessing time here is also trivially $\tilde{O}(n^2)$. ◀

5 Sub-category range counting and equivalence

Here, we will focus on SCRC queries.

5.1 Equivalences

We show that when the catalog tree $\mathcal{G}$ is a path, then the SCRC problem in 1D is equivalent to a number of well-studied problems, as follows.

▶ **Theorem 14.** *The following problems are equivalent: (i) SCRC when $\mathcal{G}$ is a single path on a one-dimensional input P, (ii) 3-sided distinct coordinate counting (for a planar point set P), (iii) 3-sided color counting (for a planar point set P), (iv) 3D dominance color counting (for a 3D point set P), and finally (v) 3D dominance counting (for a 3D point set P).*

To prove this, first in the appendix (Section B), we show the following lemma.

▶ **Lemma 15.** *When $\mathcal{G}$ is a path, the SCRC problem on a one-dimensional input P is equivalent to the 3-sided distinct coordinate counting problem.*

The above lemma shows that (i) and (ii) are equivalent. Next, we observe that (ii), (iii), and (iv) all reduces to (v): (iii) is a generalization of (ii), the reduction from (iii) to (iv) is standard by mapping a 2D input point (x_i, y_i) to the 3D point $(-x_i, y_i, x_i)$ and the 3-sided query range $\mathcal{R} = [q_\ell, q_r] \times (-\infty, q_t]$ to the 3D dominance range $(-\infty, -q_\ell] \times (-\infty, q_t] \times (-\infty, q_x]$. The reduction from (iv) to (v) was shown by Saladi [18]. Thus, the only remaining piece of the puzzle is to show a reduction from (v) to (ii). We do this next.

▶ **Theorem 16.** *3D dominance counting can be solved by 3-sided distinct coordinate counting.*

Proof. Consider an instance of 3D dominance counting. First, we reduce the instance to rank space which means we can assume that query coordinates are integers and the input coordinates are *distinct* integers between 1 and n. For an input point $p_i = (x_i, y_i, z_i)$, we create four points $p_i^1 = (-x_i, z_i)$, $p_i^2 = (y_i, z_i)$, $p_i^3 = (-x_i, z_i)$ and $p_i^4 = (y_i, z_i + 0.5)$. Next, we create two 2D 3-sided distinct Y-coordinate counting structures. In the first structure we put p_i^1 and p_i^2, and in the second structure we put p_i^3 and p_i^4. Given a query point $p = (x, y, z)$, we transform it into the 3-sided range $r_p = [-x, y] \times (\infty, z + 0.5]$.

We claim the number of dominated points of p is the difference between the outputs of the two data structures on r_p. If p dominates p_i then $x_i \leq x$, $y_i \leq y$ and $z_i \leq z$, hence $-x \leq -x_i$ and $z_i + 0.5 \leq z + 0.5$. This means that all four point are inside r_p. The first data structure counts 1, since the two points share the second coordinate and the second data structure counts 2 since they have distinct second coordinates.

The key observation is that if p does not dominate p_i at least one of $(-x_i, z_i)$ or (y_i, z_i) and $(y_i, z_i + 0.5)$ will not be in the range, which consequently implies hence the difference of the outputs will be zero (regarding p_i). The reduction is clearly linear. ◀

▶ **Corollary 17.** *SCRC problem in 1D on category tree can be solved using $O(n \log^2 n / \log\log n)$ space and with the optimal query time of $O(\log n / \log\log n)$.*

Proof. We split the tree into its heavy path decomposition. Once again, we need to look deeper into the details of the heavy-path decomposition. Start from the root of $\mathcal{G}$ and follow a path π to a leaf of $\mathcal{G}$ where at every node u of $\mathcal{G}$, we always descend to a child of u that has the largest subtree. For every node $v \in \pi$, consider the subset $P_v \subset P$ that have a color from the set $\mathcal{G}_{\leq}(v)$. Build another instance of SCRC problem where the category graph is set to π, and a point $p \in P_v$ is assigned color v. In this instance of SCRC, the category graph is a path and by Theorem 14 it is equivalence to 3D dominance counting and thus it can be solved with $O(n \log n / \log\log n)$ space and with $O(\log n / \log\log n)$ query time [13]. Observe that if for the query pair (I, v_q), consisting of an interval I and a node $v_q \in \mathcal{G}$, we have $v_q \in \pi$, then the query can readily be answered. Otherwise, v_q must not be on the central path π. To handle such queries, we simply recurse on every tree that remains after removing π.

By the heavy path decomposition, the depth of the recursion is $O(\log n)$. Furthermore, the paths obtained at the depth i of the heavy path decomposition are independent and thus in total they contain $O(n)$ points. Over all the $O(\log n)$ levels, it blows up the space by a factor of $O(\log n)$. Note that to answer a query (I, v_q), we need to find the level i of the heavy path decomposition and a path π_i that contains v_q. However, this can simply be answered by placing a pointer from v_q to the appropriate data structure. Then, after finding π_i, the query can be answered using a single dominance range counting query in $O(\log n / \log\log n)$ time. ◀

The query time of the above data structure is also optimal which follows from combining our reductions with previously known lower bounds [17].

5.2 A Conditional Lower Bound for SCRC

For SCRC where the underlying category graph is a DAG there is a trivial upper bound of $O(n^2)$ space and $O(\log n)$ query time: for each node v_i in the DAG store a 1D CRC structure on $\mathcal{G}_{\leq}(v_i)$. To answer a query (I, v_q), we simply query the CRC structure on v_q with the interval I. The conditional lowerbound on HCC can easily be extended to SCRC.

▶ **Corollary 18.** *Assuming the Orthogonal Vectors conjecture, any solution to the 1D subcategory range counting problem on a DAG using $P(n)$ preprocessing time and $Q(n)$ query time, must obey $P(n) + nQ(n) \geq n^{2-o(1)}$.*

References

1 Pankaj K. Agarwal. Range searching. In J. E. Goodman, J. O'Rourke, and C. Toth, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, Inc., 2016.

2 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

3 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58, 2015.

4 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014.

5 Bernard Chazelle. Functional approach to data structures and its use in multidimensional searching. *siamjour*, 17(3):427–462, 1988. doi:10.1137/0217026.

6 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988.

7 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, 2019.

8 Johannes Fischer. Optimal succinctness for range minimum queries. In *Proc. 9thLatin American Symposium on Theoretical Informatics (LATIN)*, pages 158–169, 2010.

9 P. Gupta, R. Janardan, S. Rahul, and M. Smid. Computational geometry: generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, chapter 67, pages 1–17. Chapman & Hall/CRC, 2017.

10 P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

11 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

12 Meng He and Serikzhan Kazi. Data Structures for Categorical Path Counting Queries. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 191, pages 15:1–15:17, 2021.

13 Joseph JaJa, Christian W. Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15thInternational Symposium on Algorithms and Computation (ISAAC)*, pages 558–568, 2004.

14 R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3:39–69, 1993.

15 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal of Computing*, 38(3):982–1011, 2008.

16 Kasper Green Larsen and Freek van Walderveen. Near-optimal range reporting structures for categorical data. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 265–277, 2013.

17 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. In *Proc. 49thProceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2008.

18 Rahul Saladi. Approximate range counting revisited. *Journal of Computational Geometry*, 12(1), 2021.

19 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.
20 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. International Colloquium on Automata, Languages and Programming (ICALP).

A HCC is at Least as Hard as Color Counting

Consider an instance of regular colored counting given by a point set P in $\mathbb{R}^d$ where each point is assigned a color from a set C. We build two hierarchical color counting data structures. In the first data structure, all colors in C are represented by leaf nodes in a balanced binary tree T_1; for simplicity we assume $|C|$ is a power of two; otherwise, we add dummy colors to C. In the second data structure, for every pair of sibling leaves c and c', we "collapse them" into their parent c_p, to obtain a second balanced binary tree T_2, meaning, any point that had color c or c' will receive c_p as its color. The resulting colored point set will be stored in the second hierarchical color counting data structure.

Given a query range $\mathcal{R}$ for the regular colored counting problem, we query both data structures with $\mathcal{R}$ and report their difference. We claim it will be the correct answer to the regular colored counting query.

To see this, we can consider two sibling leaf colors c and c' and their parent c_p in T_1. Case one is when none of them is in the query range $\mathcal{R}$. In this case, neither data structure will count anything and thus their difference will also not count either color. The second case is when exactly one of them, say c is in the query. In this case, the first data structure counts the leaf c and the path connecting c_p to the root of T_1 where as the second counts only the latter and thus the difference counts c exactly once. Lastly if both leaves are in the query, the first data structure counts two more leafs when compared to the second data structure (the two leaf nodes in the query) and we again get the correct output. The three cases are summarized in Figure 2.

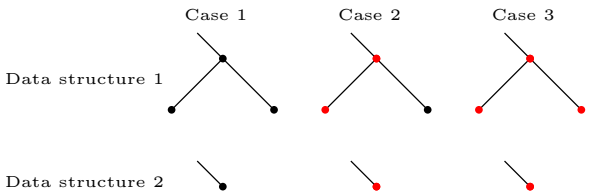


Figure 2 Each case, where the red dots corresponds to points counted.

B Reducing the SCRC Problem to the 3-sided Distinct Coordinate Counting Problem

Here, we would like to show that when $\mathcal{G}$ is a path, the SCRC Problem reduces to the 3-sided distinct coordinate counting problem.

To do that, first consider a 1D input point set for the SCRC problem. Map the input point x_i with color $c_i \in \mathcal{G}$, to the point $(x_i, h(c_i))$, where $h(c_i)$ is the number of nodes below c_i on the path $\mathcal{G}$. Store the resulting point set in a data structure for the distinct coordinate counting queries. Then, given a query interval $[a, b]$ and query node c_q, we create the 3-sided query range $\mathcal{R} = [a, b] \times (-\infty, h(c_q)]$. Observe that if a point $(x_i, h(c_q))$ lies inside $\mathcal{R}$, it implies that c_i is below c_q and that $x_i \in [a, b]$. Thus, the number of distinct Y-coordinates inside $\mathcal{R}$ is precisely the answer to the SCRC problem.

To show the converse reduction, consider a 2D point set P for the 3-sided distinct coordinate counting. We create a node $v(y)$ for each distinct Y-coodinate y in the input set, meaning, $\mathcal{G}$ is a path that has as many vertices as the number of distinct Y-coordinates in P.

For each point $p_i = (x_i, y_i)$, we create a 1D point with x-coordinate x_i and color $v(y_i)$. Consider a query range $\mathcal{R} = [a, b] \times (-\infty, c]$ and let y be the predecessor of c among the Y-coordinates of P. We create the 1D range $[a, b]$ and query the node $v(y)$. It is straightforward to verify that the answer to the SCRC is exactly the number of distinct Y-coordinates inside $\mathcal{R}$.

C Reducing the OV Problem to SCRC on a DAG

▶ **Corollary 18.** *Assuming the Orthogonal Vectors conjecture, any solution to the 1D subcategory range counting problem on a DAG using $P(n)$ preprocessing time and $Q(n)$ query time, must obey $P(n) + nQ(n) \geq n^{2-o(1)}$.*

Proof. We pick the same underlying graph as in Theorem 11 and the point set P of η points p_i such that the color of p_i is equivalent to node b_i. We query the SCRC η times, each time using a different a_i as our query node and an interval which spans all points of P. If for a query node a_i the output is less than η we know that a_i is orthogonal to some b_j, since they do not share a coordinate where they both have a one. ◀

How to Base Security on the Perfect/Statistical Binding Property of Quantum Bit Commitment?

Junbin Fang
Jinan University, Guangzhou, China

Dominique Unruh ✉
University of Tartu, Estonia

Jun Yan[1] ✉
Jinan University, Guangzhou, China

Dehua Zhou
Jinan University, Guangzhou, China

Abstract

The concept of quantum bit commitment was introduced in the early 1980s for the purpose of basing bit commitments solely on principles of quantum theory. Unfortunately, such unconditional quantum bit commitments still turn out to be impossible. As a compromise like in classical cryptography, Dumais et al. [17] introduce the conditional quantum bit commitments that additionally rely on complexity assumptions. However, in contrast to classical bit commitments which are widely used in classical cryptography, up until now there is relatively little work towards studying the application of quantum bit commitments in quantum cryptography. This may be partly due to the well-known weakness of the general quantum binding that comes from the possible superposition attack of the sender of quantum commitments, making it unclear whether quantum commitments could be useful in quantum cryptography.

In this work, following Yan et al. [43] we continue studying using (canonical non-interactive) perfectly/statistically-binding quantum bit commitments as the drop-in replacement of classical bit commitments in some well-known constructions. Specifically, we show that the (quantum) security can still be established for zero-knowledge proof, oblivious transfer, and proof-of-knowledge. In spite of this, we stress that the corresponding security analyses are by no means trivial extensions of their classical analyses; new techniques are needed to handle possible superposition attacks by the cheating sender of quantum bit commitments.

Since (canonical non-interactive) statistically-binding quantum bit commitments can be constructed from quantum-secure one-way functions, we hope using them (as opposed to classical commitments) in cryptographic constructions can reduce the round complexity and weaken the complexity assumption simultaneously.

2012 ACM Subject Classification Theory of computation; Theory of computation → Cryptographic primitives

Keywords and phrases Quantum bit commitment, quantum zero-knowledge, quantum proof-of-knowledge, quantum oblivious transfer

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.26

Related Version *Full Version*: `https://eprint.iacr.org/2020/621`

Funding *Junbin Fang*: Junbin Fang was supported by National Natural Science Foundation of China (Grant No. 62171202).
Dominique Unruh: Dominique Unruh was supported by the ERC consolidator grant CerQuS (Grant No. 819317), by the Estonian Centre of Exellence in IT (EXCITE) funded by ERDF, by PUT team grant PRG946 from the Estonian Research Council.

[1] The corresponding author

Jun Yan: Jun Yan was supported by National Natural Science Foundation of China (Grant No. 61602208), by PhD Start-up Fund of Natural Science Foundation of Guangdong Province, China (Grant No. 2014A030310333).
Dehua Zhou: Dehua Zhou was supported by Science and Technology Project of Guangzhou City (Grant No. 201707010320).

1 Introduction

Bit commitment is an important cryptographic primitive. A bit commitment scheme can be viewed as a digital analogue of a non-transparent sealed envelope. Informally, a classical bit commitment scheme is a classical two-stage interactive protocol between a *sender* and a *receiver*, both of whom can be formalized by probabilistic polynomial-time algorithms. First in the *commit* stage, the sender commits to a bit b such that the receiver should not be able to guess its value better than a random guess; this is known as the *hiding* property. Later in the *reveal* stage, the sender opens the bit commitment and reveals the bit b to the receiver. The *binding* property guarantees that any cheating sender should not be able to open the bit commitment as $1 - b$.

As quantum technology develops quickly, in this work we study *quantum bit commitments* that allow both the sender and the receiver of commitments to run *quantum* polynomial-time algorithms and exchange *quantum* messages[2] (whereas still a *classical* bit is secured) [17, 13, 23, 24, 10, 43]. Unfortunately, neither *unconditional* quantum bit commitments are possible [28, 26]. Based on quantum complexity assumptions, there are also *two flavors* of quantum bit commitments: (computationally-hiding) *statistically-binding* quantum bit commitments [17, 23, 24] and *statistically-hiding* (computationally-binding) quantum bit commitments [43].

One reason that we are interested in quantum bit commitment is because it can be made *non-interactive* in both the commit and the reveal stages (i.e. both stages consist of just a single message from the sender to the receiver), even based on the seemingly minimum quantum-secure one-way function assumption [43, 23, 24]. In contrast, classical constructions of non-interactive statistically-binding bit commitments and constant-round statistically-hiding bit commitments are only known relying on stronger complexity assumptions [19]; some negative results suggest that the interactivity seems inherent [27, 21].

Since (classical) bit commitments are extremely useful in classical cryptography, we naturally will ask whether this is also true for quantum bit commitments in quantum cryptography. In particular, we ask the following question that is the main motivation of this work:

> **Motivating question:** *If we use non-interactive quantum bit commitments in existing (classical or quantum) cryptographic constructions, then can we still base the (quantum) security of those constructions on that of quantum bit commitment?*

If the answer to the question is "yes", then by turning to non-interactive quantum bit commitments, we may reduce the round complexity and keep the complexity assumption of cryptographic constructions to the minimum simultaneously.

---

[2] A special case of quantum bit commitments considered in the post-quantum setting [1, 34, 36, 35], a.k.a. classical bit commitments secure against quantum attacks, have classical construction; that is, honest parties' computation and communication are restricted to be classical.

Inspired by the study of complete problems for quantum zero-knowledge proofs [38, 22, 40] and more general quantum interactive proofs [33, 11], *canonical*[3] (non-interactive) quantum bit commitments are introduced in [43]. Roughly speaking, a canonical quantum bit commitment scheme can be represented by a quantum circuit pair ensemble $\{Q_0(n), Q_1(n)\}_n$. To commit a bit $b \in \{0,1\}$, perform the quantum circuit Q_b on the quantum registers (C, R) initialized in all $|0\rangle$'s state, and the quantum state of the commitment register C will be treated as the commitment. The binding property of the scheme $\{Q_0(n), Q_1(n)\}_n$, a.k.a. *honest-binding*, requires that no unitary operation performing on the decommitment register R can send the quantum state $Q_0|0\rangle$ to $Q_1|0\rangle$, and vice versa. This binding property appears even weaker than *sum-binding*[4] (which is considered as the general binding property of quantum bit commitments [17, 13, 35]). Canonical statistically-binding quantum bit commitments can be based on quantum-secure one-way functions [43].

This work. In this work, we answer the motivating question above *affirmatively* when canonical statistically-binding quantum bit commitments are used. We remark that restricting to consider quantum bit commitments of the canonical form in applications does not lose generality (in theory); refer to Subsection 1.2. We also remark that another flavor of quantum bit commitments, i.e. those that are *computationally* binding, turn out to be more exotic [13, 16, 2, 36, 35] and beyond the scope of this work.

To the best of our knowledge, we are aware of no prior work besides [43] studying the application of non-interactive quantum bit commitments solely based on quantum-secure one-way functions in quantum cryptography. Follow-up work and recent developments are referred to Subsection 1.5.

1.1 On the difficulty of basing security on that of quantum bit commitment

New difficulties will arise when we try to use quantum instead of classical bit commitments in cryptographic applications and establish their (quantum) security. This was already realized in some pioneer works on quantum commitments [17, 13, 16, 34, 36]. For the purpose of presenting this work, these new difficulties can be understood by examining Blum's zero-knowledge protocol for the **NP**-complete language Hamiltonian Cycle [8] with a general quantum bit commitment scheme plugged in; we would like to show that the resulting protocol is both zero-knowledge and sound against quantum attacks.

For *zero-knowledge*, recall that in the classical security analysis, it relies on the hiding property of bit commitment; moreover, the security reduction will rewind the possibly cheating verifier. Though quantum hiding is a straightforward generalization of classical hiding, we cannot rewind a quantum verifier freely in general [37]. Thus, the classical analysis does not extend to the quantum setting straightforwardly. Fortunately, this can be rescued by using Watrous's remarkable quantum rewinding technique [39].

The more challenging part of the security analysis lies in showing *soundness*, which is to be (if possible) based on the binding property of quantum bit commitment. This is because it is well-known that the *general* quantum *sum-binding* [17, 35] is much weaker than the classical-style binding (or *unique-binding* hereafter). Roughly speaking, sum-binding only

[3] Originally, it was called "generic" quantum bit commitment in [43] and early drafts of this work. The name "canonical" is suggested by Ananth, Qian and Yuen [4] later, which (we agree) is more appropriate.

[4] But this does not exclude the possibility that the seemingly weak honest-binding may imply stronger binding properties such as sum-binding.

guarantees that any cheating sender of bit commitment cannot open it such that $p_0 + p_1 - 1$ is non-negligible, where p_0 (resp. p_1) is the success probability of opening the commitment as 0 (resp. 1). The reason of sum-binding for quantum bit commitments can be seen such a superposition attack of the sender of bit commitment as follows. A cheating sender can commit to an arbitrary *superposition* of the bit 0 and 1, in such a way that with this superposition as the control, executes the commitment stage of the quantum bit commitment scheme *honestly* [17, 13]. In this scenario, the "committed value" will become a superposition (as opposed to a classical bit). Later in the reveal stage, the bit commitment can be opened as the same superposition. At this moment, if the (honest) receiver measures (thus collapses) this superposition, then the outcome will be a distribution over $\{0, 1\}$. In particular, both 0 and 1 could be revealed with a *noticeable* probability, e.g. when the superposition is $1/\sqrt{2}(|0\rangle + |1\rangle)$.

Even worse, when quantum bit commitments are composed in parallel (to commit a binary string) and used in some larger protocol, it is possibly the sender of commitments who decides which bit commitments will be opened. In this case, not only the revealed value but also the classical information about positions of bit commitments that will be opened could be in an arbitrary superposition. This will make the quantum security analysis (if possible) much more complicated than classical analysis.

Specific to the soundness of Blum's protocol, the cheating prover (who will play the role of the sender of commitments) may try to either open *all* quantum bit commitments as a superposition of permuted input graphs (when the verifier's challenge is 0), or open a *superposition* of subsets (each corresponding to a possible location of Hamiltonian cycles) of quantum bit commitments as all 1's (when the verifier's challenge is 1). A more formal treatment about the cheating sender's superposition attack is referred to the full paper [18, Appendix A].

For the soundness analysis of Blum's protocol, the most straightforward way is trying to argue that superpositions can somehow be viewed as *collapsed* to their corresponding probability distributions, so that the classical soundness analysis can be applied. This is possible in the post-quantum setting (where quantum-secure classical bit commitments are used) by introducing stronger (computational) *collapse-binding* commitments [36]. However, current known constructions of collapse-binding commitments are interactive and rely on stronger quantum complexity assumptions than quantum-secure one-way functions in the standard model [35]. We still do not know if any non-interactive quantum bit commitment based on quantum-secure one-way functions can satisfy some "meaningful" collapse-binding property that could be useful in applications yet.

Alternatively, one can assume without loss of generality that the verifier will measure nothing other than the qubit indicating whether to accept or not and then carries out a more direct calculation involving superpositions in the analysis, like in [43]. A technical difficulty towards this approach lies in that there could be exponentially many terms in superpositions (even only polynomial many bits are committed), and naive applications of the triangle inequality will cause an exponential blow-up of errors. One should try to avoid this potential exponential blow-up when the binding error of commitments is only guaranteed negligible (as typical in cryptography). In an earlier draft of [43], this difficulty was circumvented by composing the given commitment scheme in parallel to reduce the statistical binding error to be *exponentially* small. The technique to handle negligible binding errors is called *perturbation*, as claimed in the final conference version of [43]. However, for some reasons, the final full version of [43] has never appeared[5].

[5] This is partly because its technique will be generalized and its main result will be reproved (in a conceptually much simpler way) in this paper. This will become clear later.

Besides collapse-binding commitments [36] just mentioned, two other works [13, 16] also try to base the security of cryptographic constructions on other binding properties of quantum commitments. However, it is still open whether these quantum commitments can be realized based on standard complexity assumptions.

1.2 Our contribution

In this work, we propose an *analysis framework* for basing security of cryptographic constructions on the *perfect/statistical* binding property of *canonical* (non-interactive) quantum bit commitment used within, and devise several techniques/tricks for this purpose. For *applications*, we plug canonical perfectly/statistically-binding quantum bit commitments in three well-known constructions, including zero-knowledge, oblivious transfer, and (zero-knowledge) proof-of-knowledge, and establish their (quantum) security. Our results exemplify that (statistically-binding) quantum bit commitment could be a useful primitive in quantum cryptography.

We remark that restricting to consider quantum bit commitments of the *canonical* form does not lose generality (in theory) for two reasons: (1) It turns out that any quantum bit commitment scheme can be compiled into the canonical form [41]; (2) We believe that statistically-binding quantum bit commitments of other forms can be handled similarly (as demonstrated by a more recent work [3]; refer to Subsection 1.5).

The analysis framework. It proceeds in *two* steps:

1. Lift the *classical or quantum* security of the construction based on the *perfect/statistical* unique-binding property of bit commitment to the *quantum* security that is based on the *perfect* binding property of canonical quantum bit commitment. This step may vary from application to application, but the *basic idea* is the same: introduce what we will call "commitment measurements" to collapse the potential superposition of the committed value underlying quantum bit commitments.
2. Extend the security based on quantum *perfect* binding to quantum *statistical* binding. We highlight that this step is *not* trivial, due to the potential exponential blow-up of the error aforementioned. This is in contrast to the classical setting, where such an extension is trivial by a simple union bound. The basic idea of this step is *perturbation*, as inspired by [43]. Specifically, by perturbation it can be shown that the error also (like in the classical setting, but for a completely different reason) grows *linearly* in the number of bit commitments that will be opened. We remark this second step is *standard*, almost the same for all applications. Hence, it allows us to focus on the first step of the analysis framework for the whole security analysis.

Techniques/tricks supporting the analysis framework will be introduced in Subsection 1.3.

Applications. We will apply the analysis framework in three applications listed as below:

1. Quantum zero-knowledge proof. We plug a canonical statistically-binding quantum bit commitment scheme in Blum's protocol for the **NP**-complete language Hamiltonian Cycle and establish its quantum security. The hard part of the security analysis lies in showing its soundness, which was firstly established in [43]. Here, we give an *alternative* proof following the analysis framework [18, Lemma 11, Corollary 12], which we believe is conceptually simpler. We also note that this analysis can be easily extended to any other GMW-type zero-knowledge protocols, which in particular include the GMW protocol for Graph 3-Coloring [20].

As an immediate corollary, we reprove the following theorem (also firstly proved in [43]):

▶ **Theorem 1.** *If quantum-secure one-way functions exist, then every language in* ***NP*** *has a three-round public-coin quantum (computational) zero-knowledge proof with perfect completeness and soundness error* $1/2 + o(1)$.

By the virtue of non-interactive (quantum) commitments used, the protocol guaranteed by the theorem above reduces one round compared with the classical protocol which uses interactive bit commitments [31].

2. Quantum oblivious transfer. We plug a canonical perfectly/statistically-binding quantum bit commitment scheme in the quantum oblivious transfer protocol [6, 12, 15] and establish its quantum security. The hard part of the security analysis lies in establishing the security against Bob, who is the receiver of oblivious transfer and plays the role of the sender of quantum bit commitments in this larger protocol. Following our analysis framework, we lift the security against Bob in the case where classical perfect unique-binding bit commitments are used [6, 12, 29, 44, 9] to our setting [18, Lemma 13, Corollary 14]. As an immediate corollary, we reprove the following well-known theorem:

▶ **Theorem 2.** *If quantum-secure one-way functions exist, then there exists a constant-round 1-out-of-2 quantum oblivious transfer that is computationally secure against Alice and unconditionally secure against Bob, with the security is in the following sense: after the interaction, Alice cannot guess Bob's choice bit, while Bob is not aware of the other bit owned by Alice.*

We stress that the security achieved by the theorem above is *not* the full *simulation-security* (as mostly desired in cryptography). However, it still could be useful in some applications. For example, it can be used to construct statistically-hiding (computationally-binding) quantum bit commitment [14, 41].

We also remark that there is no gain in the round complexity by using non-interactive quantum bit commitments in the quantum oblivious transfer protocol. This is because, for example when Naor's bit commitment scheme [31] is used, the first message of the bit commitment scheme can be absorbed into earlier Alice's messages prescribed by the larger quantum oblivious transfer protocol.

3. Quantum (zero-knowledge) proof-of-knowledge. Unruh [34] shows that if commitments satisfying some specific binding property exists, then plugging it in a variant of Blum's protocol gives rise to a quantum (computational) zero-knowledge proof-of-knowledge for the **NP**-complete language Hamiltonian Cycle [8]. Moreover, Unruh shows that such commitments can be based on *injective* quantum-secure one-way functions. Here we plug a canonical *perfectly/statistically*-binding quantum bit commitment in the same variant of Blum's protocol and show that the quantum proof-of-knowledge can also be fulfilled [18, Corollary 17, Corollary 18]. As an immediate corollary, we arrive at the following *new* theorem that is previously unknown:

▶ **Theorem 3.** *If quantum-secure one-way functions exist, then every language in* ***NP*** *has a three-round public-coin quantum (computational) zero-knowledge proof-of-knowledge with perfect completeness and knowledge error* $1/2$.

Compared with Unruh's (post-quantum) result, we make use of *quantum* construction and succeed in reducing the complexity assumption to general (removing the requirement of injectiveness) quantum-secure one-way functions. This answers an open question raised by Unruh [34] affirmatively.

Our analysis of quantum proof-of-knowledge is interesting. In some detail, in the analysis we will construct a canonical knowledge extractor like the one constructed in [34], which will make use of a quantum rewinding that is also similar to the one used in [34]. However, commitments used in [34] ensure that the whole quantum system is only slightly perturbed before the rewinding, whereas in our case the system may have collapsed significantly due to the possible superposition attack of the cheating prover (who will play the role of the sender of commitments). So why the quantum rewinding works in our setting seems for a *different* reason than that in [34]. Interestingly, it turns out that the underlying reason is also the same; but how to interpret it will be different, and not clear at all as it appears. Basically, it will only become clear if one views sending a bit commitment to a superposition (using a canonical (computationally-hiding) perfectly-binding quantum bit commitment scheme) as an *implicit measurement* of this committed superposition without leaking the outcome. More discussion on this point is referred to Subsection 1.4.

1.3 Our techniques/tricks

We give a brief overview of our techniques/tricks used in this work. Their formal treatment is referred to the full paper [18, Section 4].

(Imaginary) commitment measurement. In the case that the canonical quantum bit commitment scheme used is perfectly binding, we can introduce an *imaginary* binary projective measurement performed on each claimed quantum bit commitment; we will call it "commitment measurement" hereafter. It turns out that in many interesting situations, introducing commitment measurements will *not* affect the receiver's acceptance probability of opening quantum bit commitments later. In more detail, the *commitment measurement* is just the measurement that perfectly distinguishes the *honest* commitment (meaning the sender will follow the scheme in the commit stage) to 0 and that to 1, which is *not* efficiently realizable (otherwise, the commitment scheme is not computationally hiding). In spite of this, we can introduce it for the purpose of the security analysis. The *benefit* of doing so is that the superposition of the committed value underlying quantum bit commitments will then *collapse* to its corresponding probability distribution. In turn, by averaging over all possible committed values, it suffices for us to prove the security w.r.t. an arbitrary committed value. But now the analysis is similar to the one based on perfect unique-binding. In summation, this technique is useful in realizing Step 1 of the analysis framework. Similar techniques were also used in [32, 14].

Measurement manipulation. In our quantum security analysis, we may *add* or *remove* a measurement, or *replace* a measurement with other ones. This may seem tricky, but it turns out useful. For example, without affecting the security, sometimes we may try to collapse the quantum system as much as we can by introducing new measurements, so that the analysis based on perfect unique-binding can be lifted to the our setting where canonical perfectly-binding quantum bit commitments are used; in other situations, we may try to collapse less by removing measurements, so that some quantum-specific techniques can be applied, including the *perturbation* and the *quantum rewinding* that will be introduced shortly below. In summation, this technique is useful in realizing Step 1 of the analysis framework.

Commit to secret coins. The trick of letting a cheating party commit to secret coins it used in quantum cryptographic constructions was introduced by Unruh [34, 36]. It enables a quantum rewinding to work in the security analysis, where the commitments used there are unique binding. We find that the same trick also enables a similar quantum rewinding even if canonical statistically-binding quantum bit commitments (whose binding property is much weaker than unique-binding) are used. More detail is referred to the proof overview of quantum proof-of-knowledge in Subsection 1.4.

Perturbation. We devise a generic procedure for realizing Step 2 of the analysis framework. Our basic idea is to *perturb* the quantum circuit pair (Q_0, Q_1) that represents a canonical statistically-binding quantum bit commitment scheme. The resulting perturbed scheme, denoted by $(\tilde{Q}_0, \tilde{Q}_1)$, will be sort of *perfectly binding*. We note that quantum circuits $\tilde{Q}_0, \tilde{Q}_1$ may be of *super-polynomial* size; but this is not a problem for the purpose of security analysis. A *key observation* is that the error incurred by replacing the scheme (Q_0, Q_1) with the scheme $(\tilde{Q}_0, \tilde{Q}_1)$ in any quantum computation only grows *linearly* in the number of such replacements. The *zero-binding-error* guaranteed by the quantum perfect binding property of the scheme $(\tilde{Q}_0, \tilde{Q}_1)$ can help us avoid the potential exponential blow-up of errors in the security analysis as mentioned before. Similar techniques were also used in [11, 39].

A quantum rewinding lemma with improved bound. The quantum rewinding lemma in [43] enables a similar quantum rewinding as the one used in [34]. In this work, we will use the same lemma but with an *improved* lower bound on the success probability of the quantum rewinding. It allows us to obtain the asymptotically optimal knowledge error in the analysis of quantum proof-of-knowledge [18, Section 7].

1.4 Proof overview of our applications

While Step 2 of the analysis framework is standard, in the below we give an overview of Step 1 of applying the analysis framework to the three applications aforementioned (i.e. lifting the classical/quantum security based on the perfect unique-binding property of bit commitment to the quantum security based on the perfect binding property of canonical quantum bit commitment).

Zero-knowledge proof. Perform the commitment measurement on each claimed quantum bit commitment sent by the cheating prover. Then the classical soundness analysis can be lifted to our setting straightforwardly.

Quantum oblivious transfer. Call the sender of oblivious transfer Alice and the receiver Bob. Consider the security against Bob, who will play the role of sender of commitments. Compared with the GMW-type quantum zero-knowledge proof protocols, the quantum oblivious transfer protocol will continue after Alice's opening of quantum bit commitments. Thus, compared with the soundness analysis of quantum zero-knowledge proof, we need to take into account not only Alice's acceptance probability of its verification but also the *post-verification state* of the whole system. To show the security against Bob, we also perform the commitment measurement to each claimed quantum bit commitment sent by Bob. It turns out that via a simple reduction, the (quantum) analysis for the same quantum oblivious transfer protocol but with perfect unique-binding commitments plugged in [44, 9] can be lifted to our setting straightforwardly.

Quantum proof-of-knowledge. Compared with the two applications above, the main difficulty here comes from the quantum rewinding: our quantum rewinding lemma ([18, Lemma 10], which is similar to the one used in [34]) only allows us to measure the qubit indicating whether the verifier accepts or not; but for the purpose of extracting a witness, we need to measure more!

In [34], the difficulty caused by quantum rewinding as mentioned above was circumvented by a simple trick: let the prover additionally commit (using perfect unique-binding commitments) to its secret random coins used in its first message. In this way, the prover's second message will become "unique" for the verifier to accept. In turn, removing the measurement for extracting other classical information (than the single qubit just mentioned) will not affect the success probability of the quantum rewinding. However, this argument seems not to extend straightforwardly to our setting where canonical perfectly-binding quantum bit commitments are used. This is because the potential superposition of the committed value underlying commitments may collapse *significantly* by measurements for extracting other classical information.

Interestingly, after a few thought, it turns out that the trick used in [34] to enable quantum rewinding still works in our setting, and for a similar reason! But this is not clear at all at first glance; one can only see this after one has come to realize that sending a commitment to a superposition using a canonical (computationally-hiding) perfectly-binding quantum bit commitment scheme amounts to an *implicit measurement* of this committed superposition (due to perfect binding) without leaking the outcome (due to computational hiding). Here, the "implicit measurement" is in a similar sense as the standard unitary simulation of measuring a qubit in the computational basis[6]. Intuitively, one can view commitments in this way is because the commitment to 0 and that to 1 are orthogonal, and they will never be touched by the sender of commitments after they are sent.

In the analysis of quantum proof-of-knowledge, now that the prover has already *collapsed* the quantum state by additionally sending commitments to its secret random coins used in its first message, the (explicit) measurement of its response (i.e. its second message) by the knowledge extractor to extract classical information will cause no more collapses. Therefore, removing this measurement will enable us to apply our quantum rewinding lemma like in [34].

1.5 Follow-up work and recent developments

Since the first preprint of this paper uploaded to Cryptology ePrint Archive [18] back in 2020, significant progress has been made towards studying quantum commitments and their applications. Now let us mention some of them that are most relavant to this work.

In two follow-up works, Yan [41] study general properties of quantum bit commitments through the lens of canonical quantum bit commitments. Somewhat surprisingly, it turns out that any interactive (as opposed to just non-interactive, which is already known in [43]) quantum bit commitment scheme can be compiled into the canonical form. Yan [42] manages to base the computational soundness of Blum's zero-knowledge protocol on the computational binding property of canonical quantum bit commitment (when it is used in Blum's protocol). In its analysis, different techniques are used.

Also starting from Naor's scheme [31] like [43], Bitansky and Brakerski [7] construct another non-interactive statistically-binding quantum bit commitment scheme that achieves the unique-binding; Ananth, Qian and Yuen [3] construct a two-message statistically-binding

[6] That is, initialize an ancilla qubit in the state $|0\rangle$, and then store the measurement outcome in this qubit without further disturbed afterwards.

quantum bit commitment scheme but based on pseudorandom quantum states, an arguably weaker complexity assumption than quantum-secure one-way functions [25]. A common benefit of these two constructions is that their reveal stage consists of just a single *classical* message. The latter AQY scheme is also shown to satisfy a stronger quantum statistical binding property (*AQY-binding* hereafter) that turns out useful in applications (e.g. [5]).

After a careful examination, we find that the underlying idea of AQY-binding is almost the same as that of the analysis framework introduced in this work; this is also observed in a recent work by Morimae and Yamakawa [30, Appendix B]. Roughly speaking, AQY-binding is a quantum statistical binding property for *general* (as opposed to canonical) quantum commitments that is used in a similar way as our analysis framework in applications; and it can be shown by combining similar techniques as *commitment measurement* and *perturbation* introduced in this work. Actually, by tweaking these two techniques one can show that the statistical binding property of canonical quantum bit commitment implies the AQY-binding property [41, Appendix B]. As a consequence, following [3] canonical statistically-binding quantum bit commitments can also be used to instantiate the construction in [5] to achieve the *full simulation-secure* quantum oblivious transfer[7] (as opposed to the weaker one presented in Section 6 of this paper).

Comparing our analysis framework and the AQY-binding property in applications, all the technical detail in the former analysis will be hidden in establishing the latter binding property. Thus, AQY-binding is more readily usable by cryptographers. Another advantage of AQY-binding is that it is more general, i.e. not necessarily requires canonical quantum bit commitments. In spite of this, as argued at the beginning of Subsection 1.2, restricting to consider canonical statistically-binding bit commitments and use our analysis framework does not lose generality. Moreover, our analysis framework *explicitly* allows us to ignore the statistical binding error while focusing on perfectly-binding (canonical) quantum bit commitments in general. This will often make the security analysis conceptually simpler.

References

1 Mark Adcock and Richard Cleve. A quantum Goldreich-Levin theorem with cryptographic applications. In *STACS*, pages 323–334. Springer, 2002.

2 Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems: The hardness of quantum rewinding. In *FOCS*, pages 474–483, 2014.

3 Prabhanjan Ananth, Luowen Qian, and Henry Yuen. Cryptography from pseudorandom quantum states. Cryptology ePrint Archive, Report 2021/1663, 2021. URL: `https://ia.cr/2021/1663`.

4 Prabhanjan Ananth, Luowen Qian, and Henry Yuen, 2022. Private communication.

5 James Bartusek, Andrea Coladangelo, Dakshita Khurana, and Fermi Ma. One-way functions imply secure computation in a quantum world. In Tal Malkin and Chris Peikert, editors, *CRYPTO*, volume 12825 of *Lecture Notes in Computer Science*, pages 467–496. Springer, 2021.

6 Charles H. Bennett, Gilles Brassard, Claude Crépeau, and Marie-Hélène Skubiszewska. Practical quantum oblivious transfer. In *CRYPTO*, pages 351–366, 1991.

7 Nir Bitansky and Zvika Brakerski. Classical binding for quantum commitments. In Kobbi Nissim and Brent Waters, editors, *TCC*, volume 13042 of *Lecture Notes in Computer Science*, pages 273–298. Springer, 2021.

8 Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2, 1986.

[7] We believe that techniques/tricks introduced in this work suffices for this job, too.

9 Niek J. Bouman and Serge Fehr. Sampling in a quantum population, and applications. In *CRYPTO*, pages 724–741, 2010.

10 André Chailloux and Iordanis Kerenidis. Optimal bounds for quantum bit commitment. In *FOCS*, pages 354–362, 2011.

11 André Chailloux, Iordanis Kerenidis, and Bill Rosgen. Quantum commitments from complexity assumptions. In *ICALP (1)*, pages 73–85, 2011.

12 Claude Crépeau. Quantum oblivious transfer. *Journal of Modern Optics*, 41(12):2445–2454, 1994.

13 Claude Crépeau, Paul Dumais, Dominic Mayers, and Louis Salvail. Computational collapse of quantum state with application to oblivious transfer. In *TCC*, pages 374–393, 2004.

14 Claude Crépeau, Frédéric Légaré, and Louis Salvail. How to convert the flavor of a quantum bit commitment. In *EUROCRYPT*, pages 60–77, 2001.

15 Ivan Damgård, Serge Fehr, Carolin Lunemann, Louis Salvail, and Christian Schaffner. Improving the security of quantum protocols via commit-and-open. In *CRYPTO*, pages 408–427, 2009.

16 Ivan Damgård, Serge Fehr, and Louis Salvail. Zero-knowledge proofs and string commitments withstanding quantum attacks. In *CRYPTO*, pages 254–272, 2004.

17 Paul Dumais, Dominic Mayers, and Louis Salvail. Perfectly concealing quantum bit commitment from any quantum one-way permutation. In *EUROCRYPT*, pages 300–315, 2000.

18 Junbin Fang, Dominique Unruh, Jun Yan, and Dehua Zhou. How to base security on the perfect/statistical binding property of quantum bit commitment? Cryptology ePrint Archive, Report 2020/621, 2020. URL: `https://ia.cr/2020/621`.

19 Oded Goldreich. *Foundations of Cryptography, Basic Tools*, volume I. Cambridge University Press, 2001.

20 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

21 Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - a tight lower bound on the round complexity of statistically-hiding commitments. In *FOCS*, pages 669–679, 2007.

22 Hirotada Kobayashi. Non-interactive quantum perfect and statistical zero-knowledge. In *ISAAC*, pages 178–188, 2003.

23 Takeshi Koshiba and Takanori Odaira. Statistically-hiding quantum bit commitment from approximable-preimage-size quantum one-way function. In *TQC*, pages 33–46, 2009.

24 Takeshi Koshiba and Takanori Odaira. Non-interactive statistically-hiding quantum bit commitment from any quantum one-way function. *arXiv:1102.3441*, 2011.

25 William Kretschmer. Quantum pseudorandomness and classical complexity. In Min-Hsiu Hsieh, editor, *TQC*, volume 197 of *LIPIcs*, pages 2:1–2:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

26 Hoi-Kwong Lo and Hoi Fung Chau. Why quantum bit commitment and ideal quantum coin tossing are impossible. *Physica D: Nonlinear Phenomena*, 120(1):177–187, 1998.

27 Mohammad Mahmoody and Rafael Pass. The curious case of non-interactive commitments - on the power of black-box vs. non-black-box use of primitives. In *CRYPTO 2012*, pages 701–718, 2012.

28 Dominic Mayers. Unconditionally secure quantum bit commitment is impossible. *Physical Review Letters*, 78(17):3414–3417, 1997.

29 Dominic Mayers and Louis Salvail. Quantum oblivious transfer is secure against all individual measurements. In *Physics and Computation, 1994. PhysComp'94, Proceedings., Workshop on*, pages 69–77. IEEE, 1994.

30 Tomoyuki Morimae and Takashi Yamakawa. Quantum commitments and signatures without one-way functions. *Cryptology ePrint Archive, Report 2021/1691*, 2021. URL: `https://ia.cr/2021/1691`.

31 Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

32 Oded Regev. Witness-preserving amplification of QMA, 2006. Lecture notes of course Quantum Computation.

33 Bill Rosgen and John Watrous. On the hardness of distinguishing mixed-state quantum computations. In *CCC*, pages 344–354. IEEE Computer Society, 2005.

34 Dominique Unruh. Quantum proofs of knowledge. In *EUROCRYPT*, pages 135–152, 2012.

35 Dominique Unruh. Collapse-binding quantum commitments without random oracles. In *ASIACRYPT*, pages 166–195, 2016.

36 Dominique Unruh. Computationally binding quantum commitments. In *EUROCRYPT*, pages 497–527, 2016.

37 Jeroen van de Graaf. *Towards a formal definition of security for quantum protocols*. PhD thesis, Université de Montréal, 1997.

38 John Watrous. Limits on the power of quantum statistical zero-knowledge. In *FOCS*, pages 459–468, 2002.

39 John Watrous. Zero-knowledge against quantum attacks. *SIAM J. Comput.*, 39(1):25–58, 2009. Preliminary version appears in *STOC* 2006.

40 Jun Yan. Complete problem for perfect zero-knowledge quantum proof. In *SOFSEM*, pages 419–430, 2012.

41 Jun Yan. General properties of quantum bit commitments. Cryptology ePrint Archive, Report 2020/1488, 2020. URL: `https://ia.cr/2020/1488`.

42 Jun Yan. Quantum computationally predicate-binding commitments with application in quantum zero-knowledge arguments for NP. In *ASIACRYPT*, volume 13090 of *Lecture Notes in Computer Science*, pages 575–605. Springer, 2021.

43 Jun Yan, Jian Weng, Dongdai Lin, and Yujuan Quan. Quantum bit commitment with application in quantum zero-knowledge proof (extended abstract). In *ISAAC*, pages 555–565, 2015.

44 Andrew Chi-Chih Yao. Security of quantum protocols against coherent measurements. In *STOC*, pages 67–75, 1995.

Improved Compression of the Okamura-Seymour Metric

Shay Mozes
Reichman University, Herzliya, Israel

Nathan Wallheimer
University of Haifa, Israel

Oren Weimann
University of Haifa, Israel

Abstract

Let $G = (V, E)$ be an undirected unweighted planar graph. Let $S = \{s_1, \ldots, s_k\}$ be the vertices of some face in G and let $T \subseteq V$ be an arbitrary set of vertices. The *Okamura-Seymour metric compression* problem asks to compactly encode the S-to-T distances.

Consider a vector storing the distances from an arbitrary vertex v to all vertices $S = \{s_1, \ldots, s_k\}$ in their cyclic order. The *pattern* of v is obtained by taking the difference between every pair of consecutive values of this vector. In STOC'19, Li and Parter used a VC-dimension argument to show that in planar graphs, the number of *distinct* patterns, denoted $p_{\#}$, is only $O(k^3)$. This resulted in a simple $\tilde{O}(\min\{k^4 + |T|, k \cdot |T|\})$ space compression of the Okamura-Seymour metric.

We give an alternative proof of the $p_{\#} = O(k^3)$ bound that exploits planarity beyond the VC-dimension argument. Namely, our proof relies on cut-cycle duality, as well as on the fact that distances among vertices of S are bounded by k. Our method implies the following:
(1) An $\tilde{O}(p_{\#} + k + |T|)$ space compression of the Okamura-Seymour metric, thus improving the compression of Li and Parter to $\tilde{O}(\min\{k^3 + |T|, k \cdot |T|\})$.
(2) An optimal $\tilde{O}(k + |T|)$ space compression of the Okamura-Seymour metric, in the case where the vertices of T induce a connected component in G.
(3) A tight bound of $p_{\#} = \Theta(k^2)$ for the family of Halin graphs, whereas the VC-dimension argument is limited to showing $p_{\#} = O(k^3)$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Shortest paths, planar graphs, metric compression, distance oracles

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.27

Related Version *Full Version*: `https://cs.haifa.ac.il/~oren/Publications/Nathan.pdf`

Funding Israel Science Foundation grant 810/21.

1 Introduction

Planar metric compression. The shortest path metric of planar graphs is one of the most popular and well-studied metrics in computer science. The planar graph *metric compression* problem is to compactly encode the distances between a subset of k terminal vertices so that we can retrieve the distance between any pair of terminals from the encoding. On an n-vertex planar graph $G = (V, E)$, a naïve encoding uses $\tilde{O}(\min\{k^2, n\})$ bits (by either storing the $k \times k$ distance matrix or alternatively by storing the entire graph[1]). It turns out that this naïve bound is actually optimal (up to logarithmic factors) for *weighted* planar graphs, as shown by Gavoille et al. [16]. It is important to note that their lower bound

[1] Naïvely, this takes $O(n \log n)$ bits, but can be done with $O(n)$ bits [4, 10, 26, 30].

applies even when all terminals lie on a single face. The complexity of *unweighted undirected* planar graphs is also well-understood. Gavoille et al. [16] (see also [1]) gave a lower bound of $\Omega(\min\{k^2, \sqrt{k \cdot n}\})$, and Abboud et al. [1] gave a matching upper bound.

If we are willing to settle for *approximate* distances, then there are ingenious compressions requiring only $\tilde{O}(k)$ bits [21, 22, 29]. The problem has also been extensively studied (both in the exact [6, 7, 9, 17, 23, 24] and the approximate [2, 5, 8, 13, 14, 18, 19] settings) for the case where we require that the compression is itself a graph (that contains the terminals and preserves their distances).

The Okamura-Seymour metric compression. An important special case for which tight bounds are not yet known, is when the planar graph is unweighted and undirected and we want to encode the $S \times T$ distances between a set of k source terminals $S = \{s_1, s_2, \ldots, s_k\}$ lying consecutively on a single face and a subset of target terminal vertices $T \subseteq V$. A query (v, s_i) to the encoding (with $v \in T$ and $s_i \in S$) returns the v-to-s_i distance.

- When $T = S$, it is possible to exploit the *Unit-Monge* property to obtain an $O(k \log k)$ space encoding with $O(\log k)$ query time [1]. In fact, even if $T \neq S$ the Unit-Monge property implies an (optimal) $\tilde{O}(|T| + k)$ space encoding, as long as the vertices of T lie (not necessarily consecutively) on single face.
- When $T = V$, the MSSP data structure of Eisenstat and Klein [12] gives an $O(n)$ space encoding with $O(\log n)$ query time.
- For arbitrary S, T, Li and Parter [25] recently presented a compression of size $\tilde{O}(\min\{k^4 + |T|, k \cdot |T|\})$ and query time $O(1)$.[2] This compression is useful algorithmically. In the distributed setting, Li and Parter used it to compute the diameter of a planar graph in $\tilde{O}(poly(D))$ rounds where D is the graph's diameter. It was also used to develop an exact distance oracle with subquadratic space and constant query time [15].

The Li-Parter compression. At the heart of the Li-Parter compression [25], is the notion of a *pattern*. Let $d(\cdot, \cdot)$ denote the shortest path metric of G. The pattern of a vertex $v \in V$ is the vector $p_v = \langle d(v, s_2) - d(v, s_1), d(v, s_3) - d(v, s_2), \ldots, d(v, s_k) - d(v, s_{k-1}) \rangle$. Since the graph is unweighted, every entry of p_v is in $\{-1, 0, 1\}$ by the triangle inequality. This already gives an efficient way to encode v's distances to S: Instead of explicitly storing these distances (using $O(k \log n)$ bits), store p_v and $d(v, s_1)$ (using $O(k + \log n)$ bits). This way, any distance $d(v, s_i)$ can be retrieved by $d(v, s_i) = d(v, s_1) + \sum_{j=1}^{i-1} p_v[j]$.

The main contribution of Li and Parter in this context is in showing that, while there are overall n patterns in the graph, there are only $O(k^3)$ *distinct* patterns:

▶ **Theorem 1** ([25]). *The number of distinct patterns over all vertices of the graph is $O(k^3)$.*

The compression follows easily from the above theorem: Store one table that contains all the distinct patterns of vertices in T, and another table that contains for every $v \in T$ the value $d(v, s_1)$ and a pointer to p_v in the first table. Since there cannot be more than $|T|$ distinct patterns, the size of the first table is $O(\min\{k^4, k \cdot |T|\})$. The size of the second table is $\tilde{O}(|T|)$. The query time is $O(k)$ but can be improved to $O(1)$ by storing precomputed prefix-sums of every pattern (increasing the size of the first table by a logarithmic factor).

[2] The actual bound stated by Li and Parter [25, Theorem 1.2] is $\tilde{O}(k^3 \cdot D + |T|)$ where D is the diameter of the graph. The reason for the additional D factor is that they store all possible distance tuples $d_v = \{d(v, s_i)\}_{i=1}^{k}$ instead of all possible patterns p_v. The reason for the missing k factor is simply a mistake in their paper [25, p. 155].

The original proof of Theorem 1 [25]. Let us assume that the distinguished face is the infinite face. For convenience, we transform[3] the problem so that patterns are binary rather than ternary (i.e. over $\{-1,1\}$ instead of $\{-1,0,1\}$). To this end, we subdivide every edge of the graph to get a new (unweighted) graph G'. In particular, we replace each edge $\{s_i, s_{i+1}\}$ of the infinite face with a dummy vertex w_i and edges $\{s_i, w_i\}, \{w_i, s_{i+1}\}$. For every vertex u of G', let $\hat{p}_u$ be the pattern of u w.r.t. the set of vertices $S' = \{s_1, w_1, s_2, w_2, \ldots, s_k, w_k\}$. Observe that the parity of u-to-s_i distances is different from the parity of u-to-w_j distances, for all i, j's. Hence, $\hat{p}_u$ is a binary vector (i.e. over $\{-1,1\}$). Additionally, for every vertex v of G we can retrieve its pattern p_v from $\hat{p}_v$ since $p_v[i] = (\hat{p}_v[2i-1] + \hat{p}_v[2i])/2$. See Figure 1. Hence, we henceforth assume that patterns p_v are over $\{-1,1\}$ (i.e. we replace p_v with $\hat{p}_v$). For brevity, we also assume that patterns are of length $k-1$ (rather than $2k-1$).

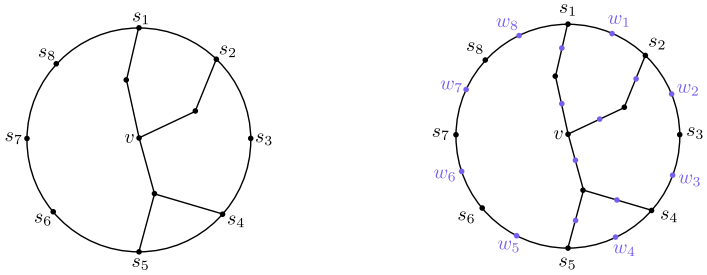


Figure 1 Before (left) and after (right) the transformation that makes all patterns binary. Every edge is subdivided and a new vertex (in color) is put in the middle. In this example, $p_v = \langle 0,1,-1,0,1,1,-1\rangle$ and $\hat{p}_v = \langle 1,-1,1,1,-1,-1,1,-1,1,1,1,1,-1,-1,-1\rangle$.

Li and Parter's VC-dimension argument is based on the simple observation that, by planarity, there cannot be two vertices v and u and 4 indices $a < b < c < d$ such that $p_u[a] = -1, p_u[b] = 1, p_u[c] = -1, p_u[d] = 1$ but $p_v[a] = 1, p_v[b] = -1, p_v[c] = 1, p_v[d] = -1$. The reason is that such $(-1,1,-1,1), (1,-1,1,-1)$ patterns correspond to an illegal configuration of shortest paths in planar graphs.

Consider arranging all the patterns as the rows of a binary matrix P. The *VC-dimension* of P is defined as the largest number d, such that there exists in P a submatrix of d columns that contains all possible 2^d different rows (i.e. all possible binary numbers with d digits). The above forbidden configuration implies that there is no submatrix with 4 columns or more that contains all possible rows, hence the VC-dimension of P is at most 3. By the well known Sauer's Lemma [27], this means that there are $O((k-1)^3) = O(k^3)$ distinct rows. This is the entire proof.

Limitations of the original proof. It remains an open problem whether the number of distinct patterns in planar graphs is $\Theta(k^3)$ or less (there is a simple $\Omega(k^2)$ lower bound). We do know however that there is no hope of improving $\Theta(k^3)$ using the VC-dimension argument: Consider the following set of sequences over $\{-1,1\}^{k-1}$:

$$\left\{(-1)^{x_1} \circ 1^{x_2} \circ (-1)^{x_3} \circ 1^{k-1-x_1-x_2-x_3} \mid x_1 + x_2 + x_3 < k\right\}$$

There is no pair of sequences in this set that contains the forbidden $(1,-1,1,-1), (-1,1,-1,1)$ configuration, and yet its cardinality is $\Theta(k^3)$. This means that any improvement to the $O(k^3)$ bound on the number of distinct patterns in planar graphs would have to further

[3] This transformation was suggested by Li and Parter in their STOC'19 talk.

exploit structural properties of planar graphs. In fact, even in the restricted family of Halin graphs, where we know that there are only $\Theta(k^2)$ distinct patterns (see Appendix A), the VC-dimension argument is limited to proving $O(k^3)$.

Our results and technique. We develop a new technique for analyzing and encoding the structure of patterns in a planar graph using *bisectors*. The bisector β_i associated with vertex s_i is a simple cycle in the dual graph such that all (primal) vertices on the same side of β_i have the same i'th bit in their patterns. We show that any two bisectors are arc-disjoint. This implies the following lemma:

▶ **Lemma 2.** *The patterns of every two adjacent vertices in G differ by at most two bits.*

We then show how to use this property to obtain the following compression (recall that $p_\#$ denotes the number of distinct patterns in G):

▶ **Theorem 3.** *There is an $\tilde{O}(p_\# + k + |T|)$ space compression of the Okamura-Seymour metric with $\tilde{O}(n)$ construction time and $\tilde{O}(1)$ query time. Moreover, for the special case where the vertices of T induce a connected component in G, the space is $\tilde{O}(k + |T|)$.*

By plugging $p_\# = O(k^3)$ from Theorem 1 (and the trivial compression that stores all $T \times S$ distances) we get an $\tilde{O}(\min\{k^3 + |T|, k \cdot |T|\})$ compression (i.e. a factor k improvement over Li and Parter). Moreover, for the special case where the vertices of T induce a connected component in G, we obtain an optimal $\tilde{O}(|T| + k)$ space encoding. Recall that, prior to our work, this bound was only known (using the Unit-Monge property) when the vertices of T all lie (not necessarily consecutively) on a single face [1]. In fact, even in such setting, our method gives $\tilde{O}(|T| + k)$. Thus, our method strictly dominates the one based on Unit-Monge.

An additional benefit of working with bisectors is that they can be used to bound the number $p_\#$ of distinct patterns. We show that every two bisectors can cross only $O(k)$ times. Our proof relies not only on the planar structure, but also on the fact that the distance between any two vertices of S is bounded by k (this property is not used in the VC-dimension argument). The set of all bisectors partitions the plane into regions. All (primal) vertices in the same region have the same pattern because they all lie on the same side of every bisector. Since there are $O(k^2)$ pairs of bisectors, and each pair crosses $O(k)$ times, there are only $O(k^3)$ regions (and hence only $O(k^3)$ distinct patterns). This provides an alternative proof of Theorem 1. We believe that our new technique may prove useful in settling the question of the number of distinct patterns in a planar graph. In particular, it may be that a similar argument that uses stronger structural properties will be able to show that the partition induces only $O(k^2)$ regions. We demonstrate this potential of our technique by proving such a bound for a family of graphs that includes Halin graphs:

▶ **Theorem 4.** *The number of distinct patterns over all vertices of a Halin graph is $O(k^2)$. This bound is tight.*

In contrast, the VC-dimension argument is limited to proving $O(k^3)$, even on Halin graphs.

2 Preliminaries

Let $G = (V, E)$ be an unweighted, undirected planar embedded graph. We prefer to think of G as a directed planar graph with a set of arcs $\mathcal{A}$, such that there is a pair of arcs uv, vu (embedded on the same curve) for every edge $\{u, v\} \in E$. We refer to u and v as the *tail* and *head* of uv, respectively. We refer to uv as the *reverse* of vu, or simply $rev(vu)$. However, we

use the term edge whenever the orientation is not important or when we refer to any of the arcs (possibly both). We denote by $P_{u,v}$ an arbitrary directed shortest path from u to v. For $i < j$ we denote by $S[i,j]$ a path (along the infinite face) $s_i - s_{i+1} - \cdots - s_j$. We extend the definition of $rev(\cdot)$ to paths. We denote by $P[w,y]$ the subpath of P between vertices w and y. We similarly use $P(w,y]$, $P[w,y)$, and $P(w,y)$ to denote whether the subpath includes the corresponding endpoint(s) or not. We use $\circ$ to denote a concatenation of two paths. Let C be a directed non-crossing cycle in G. We denote by $\mathsf{left}(C)$ and $\mathsf{right}(C)$ the subgraphs of G that consist of all edges, vertices and faces that are lying to the left and right of C, respectively. The arcs of C and their reverses are in both $\mathsf{left}(C)$ and $\mathsf{right}(C)$.

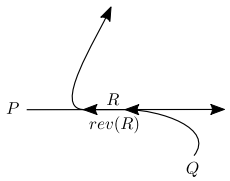


Figure 2 An example of two directed paths P and Q that cross at the crossing part R.

Let P and Q be directed paths or cycles. We say that they *cross at subpath* R if, when ignoring their orientation: (1) R is a proper (not a prefix or suffix) subpath of both P and Q, and (2) The edges of Q that follow and precede R are in different sides of P. See Figure 2. We refer to R as a *crossing part* of P and Q.

The dual graph of G is denoted by $G^* = (V^*, E^*)$. Again, we think of G^* as a directed graph with a set of arcs $\mathcal{A}^*$, defined as follows. For every arc $uv \in \mathcal{A}$, there is a corresponding arc $(uv)^* \in \mathcal{A}^*$ such that the tail and head of $(uv)^*$ are the faces that lie to the right and to the left of uv, respectively. We note that we slightly abuse the notation here, since the dual of $(uv)^*$ is $rev(uv)$ (and not uv). For $B \subseteq \mathcal{A}$, let $B^* = \{(uv)^* \mid uv \in B\}$. For a cut $X \subseteq V$, let $\delta(X) = \{uv \in \mathcal{A} \mid u \in X, v \in V \setminus X\}$. For a cycle C^* in the dual graph we say that $v \in V$ is in $\mathsf{left}(C^*)$ (resp. $\mathsf{right}(C^*)$) if the face of G^* that corresponds to v is in $\mathsf{left}(C^*)$ (resp. $\mathsf{right}(C^*)$).

3 A Bisector-Based Approach to the Okamura-Seymour Compression

In this section we present our new proof of Theorem 1 and the proofs of Lemma 2 and Theorem 3. Our main tool is the use of simple dual cycles that we call *bisectors*. In Section 3.1 we define bisectors, and prove that they are arc-disjoint and that this implies Lemma 2. In Section 3.2 we use it to prove Theorem 3. In Section 3.4 we show that every two bisectors can cross at most $O(k)$ times (this is the most technical proof of the paper). In Section 3.3 we show why this property implies a partition of the graph into $O(k^3)$ regions such that all vertices belonging to the same region have the same pattern (thus proving Theorem 1).

3.1 Bisectors

For $1 \leq i \leq k-1$, define the cut $A_i = \{v \in V \mid p_v[i] = -1\}$. Since we assume the patterns are over $\{-1, 1\}$, $V \setminus A_i = \{v \in V \mid p_v[i] = 1\}$. We define the *bisector* $\beta_i = \delta(A_i)^*$. Namely, β_i consists of all arcs $(uv)^* \in \mathcal{A}^*$ such that $p_u[i] = -1$ and $p_v[i] = 1$. Moreover, every edge $\{u, v\} \in E$ such that $p_u \neq p_v$ belongs to some bisector (possibly more than one). By cut-cycle duality, if the induced subgraphs of A_i and $V \setminus A_i$ are both connected, then β_i is a directed simple cycle in the dual graph. The next lemma implies that both induced subgraphs of A_i and $V \setminus A_i$ are connected.

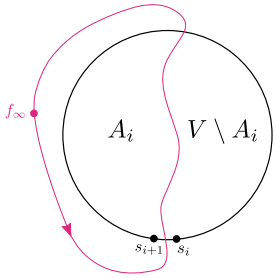


Figure 3 The bisector β_i and its corresponding cut A_i.

▶ **Lemma 5.** *For any $u \in A_i$ (resp. $V \setminus A_i$), the vertices of $P_{u,s_{i+1}}$ (resp. P_{u,s_i}) are in A_i (resp. $V \setminus A_i$).*

Proof. Assume that $u \in A_i$ (the proof of the other case is symmetric). Let v be any vertex of $P_{u,s_{i+1}}$ and assume for the sake of contradiction that $v \in V \setminus A_i$. By the definition of A_i and $V \setminus A_i$, we have that $d(u,s_i) = d(u,s_{i+1}) + 1$ and $d(v,s_{i+1}) = d(v,s_i) + 1$. By the triangle inequality, we get the following contradiction:

$$d(u,s_i) = d(u,s_{i+1}) + 1 = d(u,v) + d(v,s_{i+1}) + 1 = d(u,v) + d(v,s_i) + 2 \geq d(u,s_i) + 2. \quad \blacktriangleleft$$

By the above lemma, any two vertices $u, v \in A_i$ (resp. $V \setminus A_i$) are connected in the induced subgraph of A_i (resp. $V \setminus A_i$) by the path $P_{u,s_{i+1}} \circ rev(P_{v,s_{i+1}})$ (resp. $P_{u,s_i} \circ rev(P_{v,s_i})$). This yields the following corollary.

▶ **Corollary 6.** *β_i is a directed simple cycle in the dual graph.*

We note that the above corollary implies that for any face f, every bisector contains at most two arcs incident to f. This shows that there are only $O(k)$ total bit changes between patterns as we go along the vertices of a face f.

Another useful corollary comes from the fact that any edge whose dual is in β_i contains endpoints that are both in A_i and $V \setminus A_i$. Therefore:

▶ **Corollary 7.** *For any $u \in A_i$ (resp. $u \in V \setminus A_i$), the dual edges of $P_{u,s_{i+1}}$ (resp. P_{u,s_i}) are not in β_i.*

Note that β_i has two arcs incident to f_∞, one of them being $(s_{i+1}s_i)^*$. We think of $(s_{i+1}s_i)^*$ as the *first* arc of β_i. See Figure 3. The following lemma shows that bisectors are arc-disjoint.

▶ **Lemma 8.** *Every pair of bisectors β_i, β_j are arc-disjoint.*

Proof. Assume for contradiction that arc $(uv)^*$ appears both in β_i and in β_j. By definition, u belongs to A_i and A_j, and v belongs to $V \setminus A_i$ and $V \setminus A_j$. We first prove that under our assumption, either $P_{u,s_{i+1}}$ intersects with P_{v,s_j} or $P_{u,s_{j+1}}$ intersects with P_{v,s_i}. To see why, first note that since P_{v,s_i} and P_{v,s_j} are shortest paths, we can choose them to follow a common maximal-length prefix $P_{v,s_i}[v,w] = P_{v,s_j}[v,w]$ for some w, and they do not intersect again after w. Consider the directed cycle $C = rev(P_{w,s_j}) \circ P_{w,s_i} \circ S[i,j]$ (see Figure 4). Notice that by our choice of P_{v,s_i} and P_{v,s_j} and by the fact that $S[i,j]$ lies on the infinite face, C is not necessarily simple but it does not self-cross. We have two cases:

Case 1: $u \in \mathsf{left}(C) \setminus C$. Since $s_{i+1} \in \mathsf{right}(C)$ then (by the Jordan curve theorem and the fact that all vertices of S lie on the infinite face) $P_{u,s_{i+1}}$ must intersect with $rev(P_{v,s_j}) \circ P_{v,s_i}$. However, by Lemma 5, $P_{u,s_{i+1}}$ cannot intersect with P_{v,s_i}, therefore it intersects with $rev(P_{v,s_j})$ (and hence with P_{v,s_j}).

Case 2: $u \in \mathsf{right}(C)$. Notice that $s_{j+1} \in \mathsf{left}(C)$. If s_{j+1} is in P_{v,s_i} then $P_{u,s_{j+1}}$ intersects with P_{v,s_i} and we are done. Otherwise, since s_{j+1} is not in P_{v,s_j} by Lemma 5, then $s_{j+1} \in \mathsf{left}(C) \setminus C$. But then again (by the Jordan curve theorem) $rev(P_{u,s_{j+1}})$ (and hence $P_{u,s_{j+1}}$) must intersect with $rev(P_{v,s_j}) \circ P_{v,s_i}$. However, by Lemma 5, $P_{u,s_{j+1}}$ cannot intersect with $rev(P_{v,s_j})$, therefore it intersects with P_{v,s_i}.

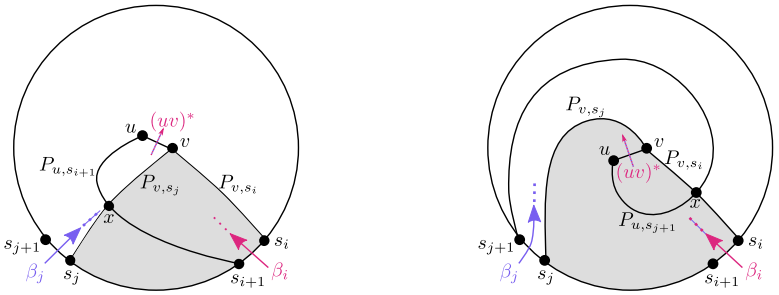


Figure 4 The two cases in the proof of Lemma 8. Case 1 on the left. Case 2 on the right. $\mathsf{right}(C)$ is shaded. For clarity, $v = w$ and C is a simple cycle in this example.

We can therefore continue under the assumption that $P_{u,s_{i+1}}$ and P_{v,s_j} intersect at a vertex x (the other case is symmetric). By the triangle inequality:

$$d(u, s_{j+1}) \leq d(u, x) + d(x, s_j) - 1$$
$$d(u, x) + d(x, s_{i+1}) \leq d(u, v) + d(v, s_i) - 1$$
$$d(v, s_i) \leq d(v, x) + d(x, s_{i+1}) - 1$$
$$d(v, x) + d(x, s_j) \leq d(v, u) + d(u, s_{j+1}) - 1.$$

Since $d(u, v)$=$d(v, u)$=1, summing these inequalities we get the contradiction $0 \leq -2$. ◀

The above lemma shows that two bisectors cannot share an arc. Note however that it is still possible that a bisector contains *reversed* arcs of another bisector. This proves Lemma 2.

3.2 A proof of Theorem 3

We begin by describing how to compute all the bisectors of the graph and report their arcs in $\tilde{O}(n)$ time. We split every edge $\{s_i, s_{i+1}\}$ by adding a dummy vertex y_i and edges $\{s_i, y_i\}, \{y_i, s_{i+1}\}$ of weight $\frac{1}{2}$. Consider a shortest path tree T_i rooted at y_i. Notice that the arcs of β_i which are not incident to f_∞, are the duals of arcs whose tail is in the subtree rooted at s_{i+1} and head is in the subtree rooted at s_i. In the interdigitating tree of T_i (i.e., the tree in the dual graph whose edges are the duals of the edges not in T_i), they are precisely the f_i-to-f_∞ path without the last arc, where $f_i \neq f_\infty$ is the face incident to $\{s_i, s_{i+1}\}$ in G. We can therefore run the MSSP algorithm of Klein [22] in $\tilde{O}(n)$ time, and report for every $1 \leq i \leq k-1$ all the arcs of β_i in $\tilde{O}(|\beta_i|)$ time. To report the two arcs of β_i which are incident to f_∞, one of them is trivially $(s_{i+1}s_i)^*$ and the other one is determined by last arc of the above f_i-to-f_∞ path. Since by Lemma 8 the arcs of bisectors are disjoint, this takes time $\tilde{O}(n + \sum_{i=1}^{k-1} |\beta_i|) = \tilde{O}(n)$ time. We label every edge $\{u, v\} \in E$ by the (at most two) bisectors that use $(uv)^*$ and $(vu)^*$. I.e., the bits that change between p_u and p_v.

We next describe the compression scheme. Recall that, by storing $d(v, s_1)$ for every $v \in T$, a query $d(v, s_i)$ (with $v \in T$ and $s_i \in S$) boils down to extracting p_v and computing its $(i-1)$'th prefix-sum. Let $\mathcal{T}$ be a spanning tree of G. Label each edge $\{u, v\}$ of $\mathcal{T}$ by the (at

most two) bits that change between the patterns of u and v. Note that there could be many (potentially $\Omega(n)$) nodes of $\mathcal{T}$ that correspond to the same pattern. In order to decrease the size of $\mathcal{T}$ to be $p_\#$ (the number of distinct patterns in G), we root $\mathcal{T}$ at some arbitrary node u. Then, for every two nodes v, w of $\mathcal{T}$ s.t $p_v = p_w$ (and w.l.o.g. v is not a descendent of w) we remove the node w and turn all it's children to be children of v (their edge labels remain the same). We repeat this process until the size of the tree is $p_\#$. We denote the resulting tree by $\mathcal{T}'$. Let Q be an Euler-tour of $\mathcal{T}'$ starting from the root u. Consider the patterns of the nodes as we go along Q, starting from p_u. In each step, the pattern only changes in at most two bits (according to the edge labels). Therefore, we can maintain all these $O(|Q|) = O(p_\#)$ versions of the pattern using a *persistent* [11] data structure for prefix-sum (e.g., using persistent segment trees [3]). Such a data structure supports both updates and prefix-sum queries to any version in $\tilde{O}(1)$ time and uses $\tilde{O}(|Q| + k) = \tilde{O}(p_\# + k)$ space. Finally, for every vertex $v \in T$ let q_v be a node in Q whose corresponding pattern is p_v. We store a pointer from v to the version of the persistent data structure at q_v, using additional $\tilde{O}(|T|)$ bits overall.

We now give a randomized $\tilde{O}(n)$ time algorithm for constructing $\mathcal{T}'$ (and hence the compression). An arbitrary spanning tree $\mathcal{T}$ can be computed in $O(n)$ time. Assume that every edge $\{w, v\}$ of $\mathcal{T}$ is labeled by the (at most two) bits that change between p_w and p_v. Let us compute the pattern of the root u of $\mathcal{T}$ with a single-source shortest-paths computation in G. We also compute the *Karp-Rabin fingerprint* [20] $\phi(p_u)$ of p_u. Such fingerprints are appealing because: (1) for any $p_w \neq p_v$, we have that $\phi(p_w) \neq \phi(p_v)$ with high probability, and (2) given $\phi(S_1)$ and $\phi(S_2)$ of two strings S_1, S_2 we can compute in $O(1)$ time the fingerprint of the concatenation $\phi(S_1 \circ S_2)$. Thus, if we maintain a complete binary tree on top of the pattern where each node contains the fingerprint of its subtree (and in particular, the root contains the fingerprint of the entire pattern), then we can update this tree in $O(\log k)$ time after changing one or two bits in the pattern.

We maintain the fingerprints in a dictionary initially containing only $\phi(p_u)$. We process the nodes of $\mathcal{T}$ starting from u, maintaining a queue of next-to-visit nodes. When we process a node v, we compute $\phi(p_v)$ from the fingerprint of v's parent, by flipping the bits according to the edge label (in $O(\log k)$ time). We then try to add $\phi(p_v)$ to the dictionary. If we find a collision with some vertex w (namely, $\phi(p_v) = \phi(p_w)$) then we delete v from $\mathcal{T}$, and set the children of v to be children of w in $\mathcal{T}$. In any case, we add the children of v to the queue so they will be processed later. Notice that a node is visited only after all its ancestors have been visited. Therefore, we can always compute its fingerprint and we never move children from a vertex to its descendent, so $\mathcal{T}$ remains a tree. In addition, the parent of every node changes or gets deleted at most once, hence the running time is $\tilde{O}(n)$. Overall, in $\tilde{O}(n)$ time we construct $\mathcal{T}'$ and the dictionary (both of size $p_\#$).

- In the special case where the vertices of T induce a connected component in G, we can skip the first part of the algorithm and simply take a path Q that traverses only the vertices of T. The rest of the construction remains the same and since $|Q| = O(|T|)$, the size of the compression is $\tilde{O}(|T| + k)$.
- In the special case where the vertices of T all lie on a single face (but not necessarily consecutively), let Q be a path that visits all the vertices of the face in clockwise order. By Corollary 6, the total number of bit changes between patterns of consecutive vertices along Q is $O(k)$. Therefore, the number of patterns encountered is $O(k)$ and hence we get an $\tilde{O}(|T| + k)$ compression for this case as well.

This completes the proof of Theorem 3.

3.3 The bisector graph and the pattern graph

The *bisector graph* $G_{\mathcal{B}}$ is the subgraph of G^* composed of the union of all the bisectors. The faces of $G_{\mathcal{B}}$ represent the patterns of G in the following way.

▶ **Lemma 9.** *For every $u, v \in V$, if u and v are embedded inside the same face f of $G_{\mathcal{B}}$, then $p_u = p_v$.*

Proof. Notice that $G_{\mathcal{B}}$ is a connected graph because all the bisectors are incident to f_∞. Hence, f is a simple cycle in G^*. Let G_f be the subgraph of G embedded inside the face f. Since there are no bisector edges embedded inside f, then in G_f there is no pair of adjacent vertices that have different patterns. Since f is a simple cycle, then by cut-cycle duality G_f is a connected subgraph. Therefore, there exists a u-to-v path in G_f, and every pair of adjacent vertices in this path have the same pattern. Hence $p_u = p_v$. ◀

By the above lemma, every pattern p of G corresponds to a unique nonempty subset of faces of $G_{\mathcal{B}}$. More precisely, a pattern p corresponds to all the faces of $G_{\mathcal{B}}$ such that the vertices of G embedded in these faces have pattern p. In particular, the number of faces of $G_{\mathcal{B}}$ is an upper bound on the number of distinct patterns in G. Therefore, if we could prove that $G_{\mathcal{B}}$ has $O(k^3)$ faces we would be done. Unfortunately, this is not the case. There can be as many as $\Omega(n)$ faces of $G_{\mathcal{B}}$ that correspond to the same pattern (see Figure 5). To tackle this, we transform $G_{\mathcal{B}}$ into a new graph $G_{\mathcal{P}}$ (called the *pattern graph*) that has only $O(k^3)$ faces and whose faces still represent all the distinct patterns of G.

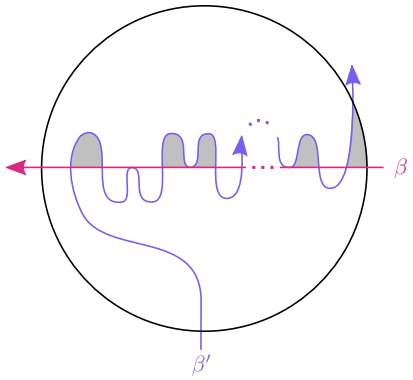


Figure 5 The shaded faces all correspond to the same pattern (assuming no other bisector crosses or separates them). They are formed when the two bisectors either cross each other or just touch (intersect without crossing). We will later see that two bisectors can cross each other at most $O(k)$ times, but, they can touch $\Omega(n)$ times, creating $\Omega(n)$ faces that correspond to the same pattern.

The *pattern graph* $G_{\mathcal{P}}$ is obtained by applying on $G_{\mathcal{B}}$ the following two-phase procedure:

(1) A *Peel* phase: Recall that while Lemma 8 says that every two bisectors are arc-disjoint, it is still possible that one bisector contains *reversed* arcs of another. In the peel phase, we re-embed the bisectors so that no bisector contains reversed arcs of another bisector. After the peel phase, crossings and touchings occur only at vertices (rather than subpaths).

(2) A *Merge* phase: In the merge phase, we merge faces that correspond to the same pattern and share a common vertex.

Peel Phase. For every two bisectors β and β', consider the set of maximal-length subpaths R, such that R is a subpath of β and $rev(R)$ is a subpath of β'. If the arc of β that follows R is in $\mathsf{right}(\beta')$ (resp. $\mathsf{left}(\beta')$), then we re-embed every arc of R on a new curve lying to the right (resp. left) of its reverse. See Figure 6. Note that the peel phase does not create any new crossings between β and β'.

Figure 6 Before (left) and after (right) a peel phase. In this example, R is a single arc, and the arc of β that follows R is in $\mathsf{left}(\beta')$.

Merge Phase. For every vertex $g \neq f_\infty$ of a bisector β, if any other bisector crosses β at g then we do nothing. Otherwise, we split g into two copies. All the arcs in $\mathsf{left}(\beta)$ that are incident to g are connected to one copy, and all the arcs in $\mathsf{right}(\beta)$ that are incident to g are connected to the other copy. Finally, we replace the arcs of β that are incident to g (say fg and gh) by a single arc fh. See Figure 7. Note that if g is not incident to any bisector other than β, then the merge phase simply contracts the arc gh. We repeat this process until there are no such bisector pairs in the graph.

Figure 7 Before (left) and after (right) a merge phase. The (shaded) face g' is obtained by merging faces h' and f'.

We now show that the above two-phase procedure maintains the relation between patterns in G and faces in $G_{\mathcal{P}}$. Namely, that every pattern in G corresponds to a unique nonempty subset of faces of $G_{\mathcal{P}}$. To this end, we extend the definition of patterns to faces of $G_{\mathcal{B}}$. This step is necessary since the peel phase creates faces that do not correspond to primal vertices.

We define the *pattern* of a face f of $G_{\mathcal{B}}$, denoted p_f, to be the length $k-1$ vector where $p_f[i] = -1$ (resp. $p_f[i] = 1$) if f is a face in $\mathsf{left}(\beta_{i+1})$ (resp. $\mathsf{right}(\beta_{i+1})$) in $G_{\mathcal{B}}$. The definition remains the same for any graph we obtain from $G_{\mathcal{B}}$ during the two-phase procedure. The following two propositions show that this definition is consistent with the original definition of patterns (of vertices).

▶ **Proposition 10.** *Let $v \in V$ be a vertex embedded inside a face f of $G_{\mathcal{B}}$. Then $p_v = p_f$.*

Proof. Let $0 \leq i \leq k-2$. If $p_f[i] = 1$ then f is in $\mathsf{right}(\beta_{i+1})$ by definition. Since $G_{\mathcal{B}}$ is a subgraph of G^*, then in G^* v is also embedded in $\mathsf{right}(\beta_{i+1})$. Hence, $v \in V \setminus A_i$ and therefore $p_v[i] = 1$. A symmetric argument shows that if $p_f[i] = -1$ then $p_v[i] = -1$. ◀

By Proposition 10, all the faces of $G_{\mathcal{B}}$ that correspond to a pattern p_v have the same face pattern. Notice that the peel phase does not change the patterns of existing faces. It can only add new faces to the graph, but no vertex of G is embedded in any of these new faces.

Hence, the relation is preserved after the peel phase. Next we show that after a merge step, every pattern still corresponds to a unique subset of faces (i.e., we show that we do not merge faces that corresponded to different patterns). Consider a single merge step happening at g (as illustrated in Figure 7). Denote by f' (resp. h') the face lying to the left of fg (resp. gh). Namely, f' and h' are the faces that get merged (a symmetric argument holds when they lie to the right of fg and gh). Let g' denote the face obtained by merging f' and h'.

▶ **Proposition 11.** $p_{f'} = p_{h'} = p_{g'}$.

Proof. Since no bisector crosses β at g, then fg and gh belong to the same side of every bisector. This, together with the fact that fg, gh, and their reverses do not belong to any other bisector, implies that f' and h' also belong to the same side of every bisector. Hence $p_{f'} = p_{h'}$. Now consider the arc fh after the merge. Since f and h belong to the same side of every bisector as fg (and gh), then g' also belongs to the same side of every bisector, hence $p_{f'} = p_{h'} = p_{g'}$. ◀

By proposition 10, if f' or h' are faces that correspond to p_v then they do not correspond to any $p_u \neq p_v$. By Proposition 11, we can set g' to correspond to p_v, and the set of faces corresponding to every pattern remains unique. This yields the following corollary.

▶ **Corollary 12.** *Every pattern of G corresponds to a unique subset of faces of $G_{\mathcal{P}}$.*

Finally, we show that the number of faces in $G_{\mathcal{P}}$ depends linearly on the number of bisector crossings. Let t be the total number of bisector crossings in $G_{\mathcal{P}}$. That is, t is the sum of the number of crossings between all pairs of bisectors.

▶ **Lemma 13.** *The number of faces in $G_{\mathcal{P}}$ is $O(t + k)$.*

Proof. By Euler's formula, it suffices to show that the number of arcs in $G_{\mathcal{P}}$ is $O(t + k)$. For every arc fg in $G_{\mathcal{P}}$, where neither f nor g is f_∞, the arc fg belongs to some bisector β. Moreover, there must exist some other bisector that crosses β at f. Otherwise, the arc would have been removed in the merge phase. Consider all the bisectors that cross β at f in a clockwise order around f starting at fg. Let β' be the one following fg. Then we charge fg to the crossing of β and β' at f. Notice that at most two arcs will be charged to this crossing of β and β' (the arc fg and the arc of β' whose tail is f). Overall, we have charged $O(t)$ arcs. The only arcs that did not get charged are the $2(k-1)$ arcs incident to f_∞. Therefore, the number of arcs in $G_{\mathcal{P}}$ is $O(t + k)$. ◀

In Section 3.4 we proved that every pair of bisectors can cross at most $O(k)$ times. Since there are $O(k^2)$ pairs of bisectors, the total number of crossings is then $t = O(k^3)$, which by Corollary 12 and Lemma 13 implies Theorem 1.

3.4 Two bisectors can cross at most $O(k)$ times

In this section we prove that every pair of bisectors can cross at most $O(k)$ times. Let β_i and β_j be two bisectors in G^* that cross each other at least once. Let $R_1, R_2, \ldots R_r$ be their crossing parts that do not contain f_∞, sorted by their order of appearance along β_i. We note that since β_i and β_j are simple cycles, the crossing parts must be disjoint. In Lemma 16 we show that the crossing parts appear in reverse order along β_j, and in Lemma 17 we use this fact to prove that the number of crossings r is at most $O(k)$. We begin by defining an important configuration of bisectors and shortest paths.

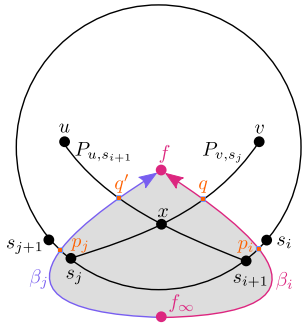


Figure 8 A configuration of bisectors β_i, β_j and shortest paths $P_{u,s_{i+1}}, P_{v,s_j}$ that must intersect. $\mathsf{right}(C^*)$ is shaded.

▶ **Lemma 14.** *Let $C^* = \beta_j[f_\infty, f] \circ rev(\beta_i[f_\infty, f])$ be a simple cycle, and let $u \in A_i, v \in V \setminus A_j$ (resp. $u \in V \setminus A_i, v \in A_j$) be two vertices in $\mathsf{left}(C^*)$ (resp. $\mathsf{right}(C^*)$). Then $P_{u,s_{i+1}}$ and P_{v,s_j} (resp. P_{u,s_i} and $P_{v,s_{j+1}}$) must intersect at some vertex x.*

Proof. We focus on the case where $u \in A_i$ and $v \in V \setminus A_j$ are vertices in $\mathsf{left}(C^*)$ (the proof of the other case is symmetric). See Figure 8. We assume that G and G^* are embedded on the same surface, such that for every $wy \in \mathcal{A}$, the curves of wy and $(wy)^*$ intersect in their middles at a single *point* p on the surface.

We refer to the two parts of the curve of wy as $(w - p)$ and $(p - y)$. For a path Q that contains arc wy, we slightly abuse notation and use $Q[\cdot, p]$ and $Q[p, \cdot]$ to denote a prefix and suffix of the curve of Q. In addition, we say that a path P of the primal graph crosses a path Q of the dual graph, if P contains an arc wy whose dual or reversed dual is in Q. In particular, it means that there exist a common point p (in the middle of wy), such that $(w - p)$ and $(p - y)$ are on different sides of Q. Let p_j be the point in the middle of $s_j s_{j+1}$, and let p_i be the point in the middle of $s_i s_{i+1}$.

Notice that $s_{i+1}, s_j \in \mathsf{right}(C^*) \setminus C^*$ by definition, and that $v \in \mathsf{left}(C^*) \setminus C^*$ by assumption (and the fact that v is not part of C^* because v is a primal vertex). Hence, P_{v,s_j} must cross C^*. However, by Corollary 7 it cannot cross $\beta_j[f_\infty, f]$, hence it must cross $\beta_i[f_\infty, f]$. This means that there is a point q that is common to both P_{v,s_j} and $\beta_i[f_\infty, f]$. In particular, let q be the last point along the curve of P_{v,s_j} that is also along the curve of C^*. Notice that $P_{v,s_j}[q, s_j] \circ (s_j - p_j)$ is a chord inside the cycle C^*. Similarly, there exists a point q' along the curve of $P_{u,s_{i+1}}$ such that $P_{u,s_{i+1}}[q', s_{i+1}] \circ (s_{i+1} - p_i)$ is a chord in C^*. The endpoints of the chords appear in clockwise order along C^* as (q, p_i, p_j, q'). It is well known that two chords in such a configuration must intersect. Therefore, there exist a primal vertex x that is common to both $P_{v,s_j}[q, s_j]$ and $P_{u,s_{i+1}}[q', s_{i+1}]$. ◀

It is important to remark that Lemma 14 holds even when the cycle C^* is a non-self-crossing non-simple cycle. Namely, if C^* intersects with itself, we let g be the first intersection vertex in $\beta_i[f_\infty, f]$, and define a cycle $\hat{C}^* = \beta_j[f_\infty, g] \circ rev(\beta_i[f_\infty, g])$. Note that since β_i and β_j are simple cycles, and by our choice of g, there are no intersections in $\hat{C}^*$. Clearly, we also have that $v, u \in \mathsf{left}(\hat{C}^*)$. Thus, we apply the Lemma to $\hat{C}^*$ instead of C^*.

▶ **Corollary 15.** *$P_{v,s_j}[v, x]$ (resp. $P_{u,s_{i+1}}[u, x]$) contains an edge whose dual is in β_i (resp. β_j).*

We are now in the position to prove the two main lemmas of this section.

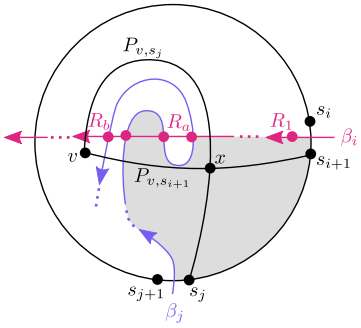


Figure 9 Case 1 in the proof of Lemma 16. The dotted parts represent parts in which there may be crossings. $\mathsf{right}(C^*)$ is shaded. In this example, P is crossed by $\beta_j[f_\infty, R_a]$ exactly twice.

▶ **Lemma 16.** *Let β_i and β_j be two crossing bisectors. Let $R_1, R_2, \ldots R_r$ be their crossing parts along β_i. Then, the crossing parts along β_j are reversed $R_r, R_{r-1}, \ldots R_1$.*

Proof. We assume that $i < j$. For $\ell \leq r$, we say that β_j enters R_ℓ from $\mathsf{left}(\beta_i)$ (resp. $\mathsf{right}(\beta_i)$) if the arc of β_j that precedes R_ℓ is in $\mathsf{left}(\beta_i)$ (resp. $\mathsf{right}(\beta_i)$). For brevity, we assume that every R_ℓ is a single vertex in V^*. Finally, we assume without loss of generality that s_j is in $\mathsf{left}(\beta_i)$ (the proof of the other case is symmetric).

Assume for the sake of contradiction that the order of appearance is not the reverse order. Then there exists a pair of crossing parts R_a and R_b such that: (1) $a < b$, (2) R_b is the crossing part following R_a in β_j, and (3) a is minimal among such pairs. Consider the cycle $C^* = \beta_j[f_\infty, R_a] \circ rev(\beta_i[f_\infty, R_a])$. Note that C^* is non-crossing since if there exists R_c for $c < a$ that $\beta_j[f_\infty, R_a]$ crosses, then a wouldn't be minimal. We have two cases:

Case 1: β_j enters R_a from $\mathsf{left}(\beta_i)$ (hence it enters R_b from $\mathsf{right}(\beta_i)$). Let $(uv)^*$ be the arc of β_j that follows R_b. We next show that v and C^* form the configuration of Lemma 14, in the special case of $v = u$.

First we show that $v \in V \setminus A_j$ and $v \in A_i$. Note that $v \in V \setminus A_j$ by definition, since v is the primal vertex lying to the right of an arc of β_j. Since $(uv)^*$ follows the crossing part R_b, and since β_j enters R_b from $\mathsf{right}(\beta_i)$, then $(uv)^* \in \mathsf{left}(\beta_i) \setminus \beta_i$. Hence, $v \in \mathsf{left}(\beta_i)$ and therefore $v \in A_i$.

Next we show that $v \in \mathsf{left}(C^*)$. For this, we show that $R_b \in \mathsf{left}(C^*) \setminus C^*$, hence the arcs incident to R_b and their corresponding primal vertices are in $\mathsf{left}(C^*)$. Consider the path $P = \beta_i[R_a, R_b]$. The first arc of P is in $\mathsf{left}(C^*)$ by the case assumption. To show that $R_b \in \mathsf{left}(C^*) \setminus C^*$ we will show that P crosses the cycle C^* an even number of times. Notice that P does not cross $\beta_i[f_\infty, R_a]$ since β_i is a simple cycle. It therefore remains to show that P crosses $\beta_j[f_\infty, R_a]$ an even number of times (equivalently, we show that $\beta_j[f_\infty, R_a]$ crosses P an even number of times). To this end, let us define a cycle $\hat{C}^* = P \circ rev(\beta_j)[R_b, R_a]$. Notice that $\hat{C}^*$ is non-crossing since R_b follows R_a in β_j. Notice that the first and last arcs of $\beta_j[f_\infty, R_a]$ are both in $\mathsf{left}(\beta_i)$. Also notice that $\mathsf{right}(\hat{C}^*)$ is included in $\mathsf{right}(\beta_i)$ (by the case assumption), therefore the first and last arcs of $\beta_j[f_\infty, R_a]$ are in $\mathsf{left}(\hat{C}^*)$. Hence, $\beta_j[f_\infty, R_a]$ crosses $\hat{C}^*$ an even number of times. Since $\beta_j[f_\infty, R_a]$ can only cross $\hat{C}^*$ at P (as β_j is a simple cycle), it must cross P an even number of times.

We can thus apply Lemma 14, and conclude that P_{v,s_j} and $P_{v,s_{i+1}}$ intersect at vertex x, and that $P_{v,s_{i+1}}[v, x]$ contains an edge whose dual is in β_j (by Corollary 15). Since the lengths of $P_{v,s_j}[v, x]$ and $P_{v,s_{i+1}}[v, x]$ are equal, $P_{v,s_{i+1}}[v, x] \circ P_{v,s_j}[x, s_j]$ is a shortest v-to-s_j path. However, since $P_{v,s_{i+1}}[v, x]$ contains an edge of β_j, this contradicts Corollary 7.

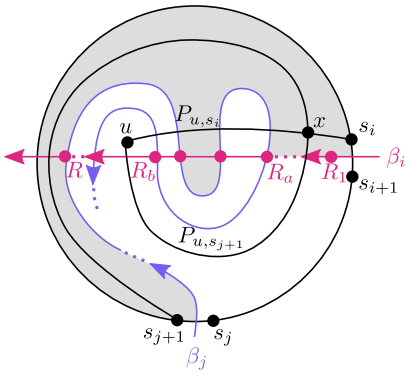


Figure 10 Case 2 in the proof of Lemma 16. The dotted parts represent parts in which there may be crossings. $\mathsf{left}(C^*)$ is shaded. In this example, P is crossed by $\beta_j[f_\infty, R_a]$ exactly twice.

Case 2: β_j enters R_a from $\mathsf{right}(\beta_i)$ (hence it enters R_b from $\mathsf{left}(\beta_i)$). Let $(uv)^*$ be the arc of β_j that follows R_b. We next show that u and C^* form the configuration of Lemma 14 (again, in the special case of $u = v$). By symmetric arguments to Case 1, we can show that $u \in A_j$ and $u \in V \setminus A_i$. We therefore only need to show that $u \in \mathsf{right}(C^*)$.
To show that $u \in \mathsf{right}(C^*)$, it suffices to show that $R_b \in \mathsf{right}(C^*) \setminus C^*$ (hence the arcs incident to R_b and their corresponding primal vertices are in $\mathsf{right}(C^*)$). Consider the path $P = \beta_i[R_a, R_b]$. The first arc of P is in $\mathsf{right}(C^*)$ by the case assumption. To show that $R_b \in \mathsf{right}(C^*) \setminus C^*$, we will show that P crosses the cycle C^* an even number of times. Since P does not cross $\beta_i[f_\infty, R_a]$, we need to show that $\beta_j[f_\infty, R_a]$ crosses P an even number of times (here is where the argument will differ from Case 1).
Let $\hat{C}^* = P \circ rev(\beta_j)[R_b, R_a]$ be a non-crossing cycle. Let R be the first crossing between $\beta_j[f_\infty, R_a]$ and β_i. R exists and is not R_a by the assumption that $s_j \in \mathsf{left}(\beta_i)$, and by the case assumption. Note that $\beta_j[f_\infty, R]$ does not cross β_i (and hence does not cross P) by definition of R, so it remains to show that the number of crossings between $\beta_j[R, R_a]$ and P is even. Notice that, since $s_j \in \mathsf{left}(\beta_i)$, the first and last arcs of $\beta_j[R, R_a]$ are both in $\mathsf{right}(\beta_i)$. Also note that $\mathsf{left}(\hat{C}^*)$ is contained in $\mathsf{left}(\beta_i)$. Therefore, the first and last arcs of $\beta_j[R, R_a]$ are in $\mathsf{right}(\hat{C}^*)$. Hence, $\beta_j[R, R_a]$ crosses $\hat{C}^*$ an even number of times. Since it can only cross $\hat{C}^*$ at P, then $\beta_j[R, R_a]$ crosses P an even number of times.
We can thus apply Lemma 14, and the contradiction follows similarly to Case 1. ◀

▶ **Lemma 17.** *Two bisectors can cross at most* $\frac{k}{2} + O(1)$ *times.*

Proof. We will prove that if two bisectors β_i, β_j cross r times then there exists a vertex $v \in V$ such that $d(v, s_j) - d(v, s_{i+1}) \geq 2r - \frac{k}{2} - O(1)$. Since the distance between any pair of vertices along the infinite face is at most $\frac{k}{2}$, then by the triangle inequality we have also $d(v, s_j) - d(v, s_{i+1}) \leq \frac{k}{2}$. Hence, we get $2r - \frac{k}{2} - O(1) \leq \frac{k}{2}$ and the lemma follows.

Again, let us assume without loss of generality that s_j is in $\mathsf{left}(\beta_i)$ (the proof of the other case is symmetric). For every $\ell < r$, consider the cycle $C^*_\ell = \beta_j[f_\infty, R_\ell] \circ rev(\beta_i[f_\infty, R_\ell])$. By the previous lemma, C^*_ℓ does not self-cross. For even (resp. odd) $r - \ell$, let $(u_\ell v_\ell)^*$ (resp. $(v_\ell u_\ell)^*$) be the arc of β_j that follows R_ℓ. See Figure 11. We assume that $r - \ell$ is even (the odd case is symmetric).

We claim that $C^*_\ell, v_\ell, v_{\ell+1}$ forms the configuration of Lemma 14. By definition of β_j we have that $v_\ell \in V \setminus A_j$. Thus, it remains to prove that $v_{\ell+1} \in A_i$ and that $v_\ell, v_{\ell+1} \in \mathsf{left}(C^*_\ell)$. By Lemma 16 and the assumption that $s_j \in \mathsf{left}(\beta_i)$, β_j reaches $R_{\ell+1}$ from $\mathsf{right}(\beta_i)$. Therefore, $(v_{\ell+1}u_{\ell+1})^* \in \mathsf{left}(\beta_i) \setminus \beta_i$. Hence, $v_{\ell+1}$ is in $\mathsf{left}(\beta_i)$ and therefore $v_{\ell+1} \in A_i$. To see that

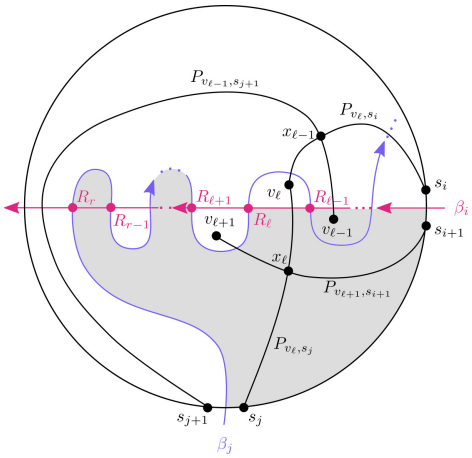


Figure 11 Two bisectors that cross r times and the vertices and shortest paths they induce. Observe that v_ℓ is in $\mathsf{right}(\beta_i)$ when $r-\ell$ is even, and otherwise in $\mathsf{left}(\beta_i)$. $\mathsf{right}(C^*_\ell)$ is shaded.

$v_\ell, v_{\ell+1} \in \mathsf{left}(C^*_\ell)$, note that by Lemma 16, $(v_{\ell+1}u_{\ell+1})^*$ is in $\beta_j[f_\infty, R_\ell]$, hence $v_{\ell+1} \in \mathsf{left}(C^*_\ell)$ by definition (as $\beta_j[f_\infty, R_\ell]$ is part of C^*_ℓ). Clearly, since β_j enters R_ℓ from $\mathsf{right}(\beta_i)$ we also have that $(u_\ell v_\ell)^* \in \mathsf{left}(C^*_\ell)$ and in particular $v_\ell \in \mathsf{left}(C^*_\ell)$.

We can thus apply Lemma 14 and conclude that P_{v_ℓ,s_j} and $P_{v_{\ell+1},s_{i+1}}$ must intersect at some vertex x_ℓ. In addition, we can conclude that $P_{v_{\ell-1},s_{j+1}}$ and P_{v_ℓ,s_i} intersect at some vertex $x_{\ell-1}$, and that $P_{v_{\ell-2},s_j}$ and $P_{v_{\ell-1},s_{i+1}}$ intersect at some vertex $x_{\ell-2}$. Therefore, by the triangle inequality and the assumption that patterns are binary, we have:

$$d(v_\ell, x_\ell) + d(x_\ell, s_j) \le d(v_\ell, x_{\ell-1}) + d(x_{\ell-1}, s_{j+1}) - 1$$
$$d(v_\ell, x_{\ell-1}) + d(x_{\ell-1}, s_i) \le d(v_\ell, x_\ell) + d(x_\ell, s_{i+1}) - 1$$
$$d(v_{\ell-1}, x_{\ell-2}) + d(x_{\ell-2}, s_{i+1}) \le d(v_{\ell-1}, x_{\ell-1}) + d(x_{\ell-1}, s_i) - 1$$
$$d(v_{\ell-1}, x_{\ell-1}) + d(x_{\ell-1}, s_{j+1}) \le d(v_{\ell-1}, x_{\ell-2}) + d(x_{\ell-2}, s_j) - 1.$$

Summing the above inequalities we get:

$$d(x_\ell, s_j) - d(x_\ell, s_{i+1}) + 4 \le d(x_{\ell-2}, s_j) - d(x_{\ell-2}, s_{i+1}). \tag{1}$$

Let $m \in \{1, 2\}$ be so that $r - m$ is even. Let $v = x_m$. By repeating Equation (1) we get:
$d(x_{r-2}, s_j) - d(x_{r-2}, s_{i+1}) + 4 \cdot \left(\frac{r}{2} - O(1)\right) \le d(v, s_j) - d(v, s_{i+1})$.
Since the distance between any two vertices along the infinite face is at most $\frac{k}{2}$, it follows that: $-\frac{k}{2} + 4 \cdot \left(\frac{r}{2} - O(1)\right) \le d(v, s_j) - d(v, s_{i+1}) \le \frac{k}{2}$. Therefore $r \le \frac{k}{2} + O(1)$. ◀

4 Conclusions

In this work we developed a technique for analyzing the structure and number of distinct patterns in undirected planar graphs. This technique leads to an improved $\tilde{O}(p_\# + k + |T|)$ space compression of the Okamura-Seymour metric, and to an optimal $\tilde{O}(k+|T|)$ compression in the special case where the vertices of T induce a connected component in G. Moreover, the technique leads to an alternative proof of the $p_\# = O(k^3)$ upper bound on the number of different patterns.

We have shown that for the family of Halin graphs, the original proof technique using VC-dimension is not tight, and that our approach easily proves the tight bound in this case. Going back to planar graphs, we were unable to come up with constructions of families of planar graphs that have $p_\# = \omega(k^2)$ patterns. We therefore make the following conjecture.

▶ **Conjecture.** *The number of distinct patterns over all vertices of a planar graph is* $O(k^2)$.

Note that, if the above conjecture turns out to be true, then our compression would automatically improve further to $\tilde{O}(\min\{k^2 + |T|, k \cdot |T|\})$ (i.e. by an additional factor of k). We hope that the tools we have developed in this work will be useful in proving this conjecture.

References

1 Amir Abboud, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal compression for the planar graph metric. In *29th SODA*, pages 530–549, 2018.

2 Amitabh Basu and Anupam Gupta. Steiner point removal in graph metrics, 2008. Unpublished Manuscript, available from `https://www.ams.jhu.edu/~abasu9/papers/SPR.pdf`.

3 Jon Louis Bentley. Solutions to klee's rectangle problems. *Unpublished manuscript*, pages 282–300, 1977.

4 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *21st CPM*, pages 138–150, 2010.

5 Hubert T.-H. Chan, Donglin Xia, Goran Konjevod, and Andréa W. Richa. A tight lower bound for the steiner point removal problem on trees. In *9th APPROX*, pages 70–81, 2006.

6 Hsien-Chih Chang, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal distance emulator for planar graphs. In *26th ESA*, pages 16:1–16:17, 2018.

7 Hsien-Chih Chang and Tim Ophelders. Planar emulators for monge matrices. In *32nd CCCG*, pages 141–147, 2020.

8 Yun Kuen Cheung. Steiner point removal – distant terminals don't (really) bother. In *29th SODA*, pages 1353–1360, 2018.

9 Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately – lower and upper bounds. In *43rd ICALP*, pages 131:1–131:14, 2016.

10 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *22nd SODA*, pages 506–515, 2001.

11 James R Driscoll, Neil Sarnak, Daniel D Sleator, and Robert E Tarjan. Making data structures persistent. *Journal of computer and system sciences*, 38(1):86–124, 1989.

12 David Eisenstat and Philip N Klein. Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs. In *45th STOC*, pages 735–744, 2013.

13 Arnold Filtser. Steiner point removal with distortion $O(\log k)$. In *29th SODA*, pages 1361–1373, 2018.

14 Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi. Relaxed voronoi: A simple framework for terminal-clustering problems. In *2nd SOSA*, pages 10:1–10:14, 2019.

15 Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen. Truly subquadratic exact distance oracles with constant query time for planar graphs. *arXiv preprint*, 2020. `arXiv:2009.14716`.

16 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

17 Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In *25th ESA*, pages 44:1–44:14, 2017.

18 Anupam Gupta. Steiner points in tree metrics don't (really) help. In *12th SODA*, pages 220–227, 2001.

19 Lior Kamma, Robert Krauthgamer, and Huy L. Nguyen. Cutting corners cheaply, or how to remove steiner points. *SIAM Journal of Computing*, 44(4):975–995, 2015.

20 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

21 Ken-ichi Kawarabayashi, Philip N Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *38th ICALP*, pages 135–146, 2011.

22 Philip N Klein. Multiple-source shortest paths in planar graphs. In *16th SODA*, volume 5, pages 146–155, 2005.

23 Robert Krauthgamer, Huy L Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014.

24 Robert Krauthgamer and Tamar Zondiner. Preserving terminal distances using minors. In *39th ICALP*, pages 594–605, 2012.

25 Jason Li and Merav Parter. Planar diameter via metric compression. In *51st STOC*, pages 152–163, 2019.

26 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *38th FOCS*, pages 118–126, 1997.

27 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

28 Maciej M Sysło and Andrzej Proskurowski. On halin graphs. In *Graph Theory*, pages 248–256. Springer, 1983.

29 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

30 György Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294, 1984.

A A $\Theta(k^2)$ Proof for Halin Graphs

In this section we prove Theorem 4. Namely, we show a tight $\Theta(k^2)$ bound on the number of patterns in a family of graphs that includes Halin graphs. The *Halin graph* family (see [28] for history and properties) is a restricted family of planar graphs. A Halin graph is obtained from an embedded tree $\mathcal{T}$ with no degree-2 vertices by attaching a cycle C to its leaves in their order of appearance according to the embedding. The cycle is then the boundary of the infinite face, and we denote its size by k. We will consider a more general family than Halin graphs. Namely, we allow the tree $\mathcal{T}$ to have degree 2 vertices, and we allow the cycle C to contain vertices that are not in $\mathcal{T}$. Such graphs can be seen as subdivided Halin graphs. We will refer to such graphs as *S-Halin* graphs. See Figure 12.

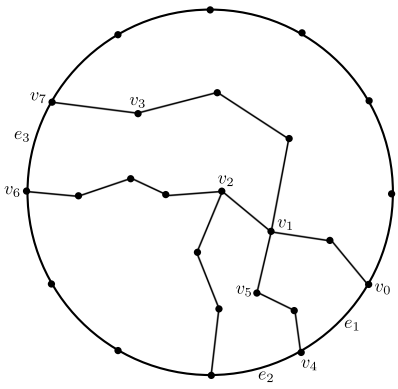


Figure 12 An S-Halin graph whose matrix of patterns has VC-dimension 3.

In this section, we show that the number of distinct patterns in S-halin graphs is only $O(k^2)$, and that this bound is tight. In contrast, we show that the VC-dimension argument is limited to proving $O(k^3)$ even on such graphs.

▶ **Lemma 18.** *Let $G = \mathcal{T} \cup C$ be an S-Halin graph, obtained by identifying the leaves of $\mathcal{T}$ with a subset of vertices of a cycle C. If the size of C is k then the number of distinct patterns in G is $O(k^2)$.*

Proof. Notice that the total number of vertices of G with degree larger than 2 is $O(k)$ (and hence, the number of faces in G is also $O(k)$). This is because C is of size k and $\mathcal{T}$ contains at most k leaves (and hence $O(k)$ vertices with degree larger than 2). Now consider the dual graph $G^* = (V^*, E^*)$. Since every bisector is a simple cycle in G^*, and since $|V^*| = O(k)$, then the number of arcs in the graphs $G_{\mathcal{B}}$ and $G_{\mathcal{P}}$ is $O(k^2)$. Therefore, by Corollary 12 the number of distinct patterns in G is $O(k^2)$. ◀

We next show that it is not possible to prove Lemma 18 using the VC-dimension argument. Namely, consider the S-Halin graph of Figure 12 and let P be the matrix whose rows are the patterns of the graph (recall that each pattern is in $\{-1, 1\}^{k-1}$).

▶ **Proposition 19.** *The VC-dimension of P is 3.*

Proof. Consider the following submatrix of P whose rows correspond to the vertices $v_0, \ldots, v_7$ and columns correspond to the edges e_1, e_2, e_3:

$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_2 & e_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{pmatrix} \end{array}$$

Since it contains all possible rows, the VC-dimension of P is at least 3. ◀

It is important to remark that we can generalize the example of Figure 12 to any large enough k, by adding vertices along the infinite face in the part between v_7 and v_0 (clockwise).

So far we have seen that S-Halin graphs have at most $O(k^2)$ distinct patterns (Lemma 18), and that the VC-dimension argument is limited to showing $O(k^3)$ distinct patterns (Proposition 19). To conclude this section, we prove that the $O(k^2)$ bound is tight:

▶ **Lemma 20.** *There exists an S-Halin graphs with $\Omega(k^2)$ distinct patterns.*

Proof. We assume that k is even and denote $k' = k/2$. We construct the tree $\mathcal{T}$ by taking the union of $k' + 1$ simple paths $P_0, P_1, \ldots, P_{k'}$ where every P_i is of length i and all P_i's originate from a common vertex $v_{0,0}$. Namely, $P_i = (v_{i,0} - v_{i,1} - \cdots - v_{i,i})$, and $v_{i,0} = v_{j,0}$ for every $i \neq j$. We choose the embedding of $\mathcal{T}$ so that in a clockwise tour around $v_{0,0}$, the order of appearance of the paths is $(P_1, P_2, \cdots, P_{k'})$.

We define an additional $v_{k',k'}$-to-$v_{1,1}$ path of length $k' + 1$ denoted $Q = (v_{k',k'} - q_1 - q_2 - \cdots - q_{k'} - v_{1,1})$. Let C be the cycle $Q \circ (v_{1,1} - v_{2,2} - \cdots - v_{k',k'})$. Let $G = \mathcal{T} \cup C$. See Figure 13. Note that $|C|$ is $2k' = k$.

Consider the patterns of G, when we choose the first vertex to be $v_{1,1}$, the second $v_{2,2}$, etc. Let $i \in [k'], j \in [i-1]$. We claim that the pattern of $v_{i,j}$ is:

$$p_{v_{i,j}} = 1^{i-j-1} \circ (-1)^j \circ 1^{k'+j+1-i} \circ (-1)^{k'-j-1} \tag{2}$$

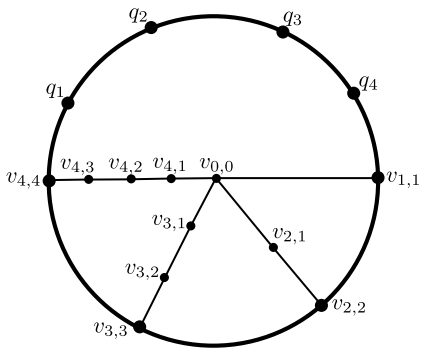


Figure 13 The S-Halin of Lemma 20 for $k = 8$.

To see why, consider the $v_{i,j}$-to-$v_{t,t}$ distances for $t \leq i$. Notice that for every $1 \leq t \leq i - j$ we have $d(v_{i,j}, v_{t,t}) = j + t$ and for every $i - j \leq t \leq i$ we have $d(v_{i,j}, v_{t,t}) = 2i - j - t$. In particular, they are equal at $t = i - j$. Therefore, the pattern of all the edges between $v_{1,1}$ and $v_{i,i}$ is $1^{i-j-1} \circ (-1)^j$.

Now consider the $v_{i,j}$-to-$v_{t,t}$ distances for $i \leq t \leq k'$. Notice that a shortest $v_{i,j}$-to-$v_{t,t}$ path will never use Q and instead will go through $(v_{i,i} - v_{i+1,i+1} - \cdots - v_{t,t})$. Namely, the distance is $d(v_{i,j}, v_{t,t}) = t - j$. Therefore, the pattern of all the edges between $v_{i,i}$ and $v_{k',k'}$ is $1^{k'-i}$.

Finally, consider the $v_{i,j}$-to-q_t distances for $1 \leq t \leq k'$. For every $1 \leq t \leq j + 1$ we have $d(v_{i,j}, q_t) = k' + t - j$ and for every $j + 1 \leq t \leq k'$ we have $d(v_{i,j}, q_t) = k' + j + 2 - t$. In particular they are equal at $t = j + 1$. Therefore the pattern of all the edges between $v_{k',k'}$ and q'_k is $1^{j+1} \circ (-1)^{k'-j-1}$. Overall, we get that $p_{v_{i,j}}$ is as in Equation (2).

Notice that $p_{v_{i,j}}$ is unique for every different $i \in [k'], j \in [i - 1]$. Since the number of such vertices is $\Omega(k'^2) = \Omega(k^2)$, there are $\Omega(k^2)$ distinct patterns in G. ◀

Improving the Bounds of the Online Dynamic Power Management Problem

Ya-Chun Liang
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Kazuo Iwama
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Chung-Shou Liao
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Abstract

We investigate the *power-down mechanism* which decides when a machine transitions between states such that the total energy consumption, characterized by execution cost, idle cost and switching cost, is minimized. In contrast to most of the previous studies on the offline model, we focus on the online model in which a sequence of jobs with their release time, execution time and deadline, arrive in an online fashion. More precisely, we exploit a different switching on and off strategy and present an upper bound of 3, and further show a lower bound of 2.1, in a dual-machine model, introduced by Chen et al. in 2014 [STACS 2014: 226-238], both of which beat the currently best result.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Online algorithm, Energy scheduling, Dynamic power management

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.28

Funding *Kazuo Iwama*: Supported by MOST Taiwan under Grants 110-2223-E-007-001, 110-2811-E-007-507 and KAKENHI Japan under Grant 16H02782.
Chung-Shou Liao: Supported by MOST Taiwan under Grants 110-2221-E-007-106-MY3 and 110-2622-8-007-017.

1 Introduction

Machines consume energy and many of them have the following property: namely machines have two states, ON and OFF, and change between the two states quite often. Furthermore, it requires a relatively large amount of energy to change from OFF to ON. For instance, a laptop goes to sleep or shutdown (OFF) if we do not touch it for, e.g., one hour, and comes back to ON if we press some key, but it is actually energy consuming. Copy machines also turn off automatically if there is no job for, e.g., ten minutes. Similarly, it consumes relatively large energy for heating them up when a new job arrives. There are many other similar examples, and it is obviously important to design online algorithms to minimize the energy consumption for this type of machines. Fortunately most of those problems are essentially equivalent to the well-known *ski-rental* problem and the optimal solution has been known for a long time both in deterministic and randomized cases [12, 16].

In this study, we investigate the online model of the dynamic power management problem, discussed by Irani et al. [14, 15] and Chen et al. [7], which aims at determining when to transition a machine between the states: *busy* or *idle* (ON) and *sleep* (OFF) to finish all input jobs such that the energy consumption is minimized. Suppose a machine M requires E units of energy to change its state from OFF to ON and needs ψ_σ units per unit of time

to keep it ON, i.e. the *idle* cost. The goal is to finish all input jobs, arriving in an online fashion, while minimizing the total energy consumption. Let us consider a simpler model first, where jobs arriving in an online fashion are required to be immediately performed without any delay. One can observe that the optimal deterministic online algorithm of the ski rental problem can be applied to solve this model, with the competitive ratio of 2. The intuition is quite straightforward, by turning off the machine M E/ψ_σ units of time after its last job is finished. Note that the performance of an online algorithm is typically evaluated by competitive analysis. More precisely, the quality of an online algorithm is measured by the worst case ratio, called *competitive ratio*, which is defined to be the fraction of the cost of the online algorithm over that of the offline optimal algorithm, where the offline algorithm is aware of all jobs in advance.

However, when considering a more complicated model, in which each job j is given as (a_j, d_j, c_j), where a_j denotes the arrival time, d_j the deadline and c_j the execution time, a dramatic change does happen. In other words, the execution of each job is allowed to be postponed, which obviously increases the problem hardness. There have been actually not many studies in the literature for the online model [12]. Irani et al. [14,15] initiated the study of combining the above power-down mechanism with *dynamic speed scaling*, where the latter technique has been widely explored in the past decades [5,8,9,15,19]. The basic concept of dynamic speed scaling is that a machine's processing speed can be adjusted dynamically, and the power consumption rate is usually represented by a convex function of the processing speed in terms of time expense. They proposed the first online algorithm with a constant competitive ratio of $\max\{c_1c_2 + c_1 + 2, 4\}$, where c_1 and c_2 are some constant parameters in a given convex power function [15]. For example, if the power function is quadratic, the upper bound of competitive ratio is 8. Chen et al. [7] considered a similar model but without using dynamic speed scaling, where an online algorithm can use two machines M_1 and M_2 instead, under the same assumption that all input jobs must be finished by the offline optimal scheduler using a single machine. This assumption is actually the so-called *single machine schedulability* condition [10], which is characterized by the following lemma.

▶ **Lemma 1.** (Chetto et al. [10]) *For any set of jobs $\mathcal{J}$, they can be optimally scheduled on one machine using the* earliest-deadline-first *(EDF) principle. That is, the job with the earliest deadline is always selected for execution at any moment, if and only if the following condition holds:*

$$\text{For any time interval } (\ell, r), \text{ we have } \sum_{j:j\in\mathcal{J},\ell\le a_j,d_j\le r} c_j \le r - \ell. \tag{1}$$

The condition is equivalent to the *earliest-deadline-first (EDF)* schedulability [19]. Here, we also remark that a feasible solution can be a preemptive schedule but the jobs can only be executed without migration. Chen et al. [7] gave an upper bound of 4 and a lower bound of 2.06 for the competitive ratio of this dual-machine problem. It is a significant contribution for online dynamic power management problems, but unfortunately, the above gap is not very small, for which there has been no improvement up to now.

Our Contribution. This paper improves both upper and lower bounds to 3 and 2.1, respectively, for exactly the same dual-machine model. Namely the competitive ratio gap between lower and upper bounds is improved from 1.94 in [7] to 0.9. Our algorithm has the same basic structure as the one in [7], but two critical differences, which exactly contributes the improvement, should deserve being mentioned. First, we delay jobs but turn on a machine earlier than its due time by a margin instead of "*energy-efficient anchor*" introduced in [7].

Second, our idle time after the machine is finished with its execution is not set to $\mathcal{B} = E/\psi_\sigma$, i.e. the so-called *break-even time*, but set to twice that value. Note that the break-even time has always been used for the class of similar problems including the famous ski-rental problem, as mentioned earlier. We hope this escape from the common tradition will help on several different occasions in the future.

Moreover, we use a standard math induction for the analysis, making our analysis significantly simpler than that of [7, 15], which will also contribute to further improvement of the bounds hopefully. In [7, 15], they made their competitive analysis by introducing what they call a "sleep interval" where the optimal offline schedule is OFF and an "awaken-interval" where their online algorithm is ON, and gave a key lemma saying a single sleep interval overlaps with at most two awaken intervals. This is clever since the analysis boils down to comparing a single (consecutively ON) interval of the online algorithm and that of the optimal offline schedule. However, a single such interval of the optimal offline schedule can still contribute to two such intervals of the online algorithm even if it can actually contribute to only one in many cases. Thus the analysis underestimates the cost of the optimal offline schedule. Our analysis splits the time line simply into "*phases*" that realizes the same intervals for an online algorithm and the optimal offline schedule, which makes it possible to use the standard math induction. Of course an online algorithm and the optimal offline schedule can execute different jobs in each phase, but that can be managed by considering "assets" and "liabilities" of jobs between phases. More details of the basic idea will be given in Sections 2 and 3.

Other Studies. In comparison with a few amount of research on *pure* dynamic power management, there have been relatively more studies which incorporate speed scaling into the power-down mechanism. Albers et al. [1] investigated the offline setting of speed scaling with a sleep state and presented a $\frac{4}{3}$-factor approximation algorithm. Antoniadis et al. [4] further improved the result to a fully-polynomial time approximation scheme. Considering the multi-machine systems, Demaine et al. [11] developed a polynomial-time algorithm based on dynamic programming. Albers et al. [2] then considered dynamic speed scaling with job migration. Very recently, Antoniadis et al. [3] presented a pseudo-polynomial time algorithm for a single machine based on a linear programming relaxation and a constant-factor approximation algorithm for the case of multiple machines. Readers may refer to the survey works [6, 13, 18] for more details.

2 Upper Bound

We formally introduce the dual-machine model proposed by Chen et al. [7]. Each machine requires a constant amount of energy to switch its state from *sleep* to either *busy* or *idle*, denoted by E. Suppose every machine uses a constant speed to execute jobs; that is, it consumes a constant amount of energy per unit of time, denoted by ψ_b when it is busy and ψ_σ when it is idle. The break-even time, as mentioned earlier, denoted by $\mathcal{B}$, is defined to be E/ψ_σ, which represents that when a machine is idle for $\mathcal{B}$ units of time, the energy consumption is equal to E, i.e. the energy consumption for switching on a machine from sleep to the other states. The standard assumption is that $\psi_\sigma \leq \psi_b$. Another assumption in this study is that $E = \psi_b = 1$ (and $\psi_\sigma \leq 1$). If E is a positive integer k, we can use our model by changing a unit time from 1 to $1/k$ (and changing the job and idle length accordingly). Thus our model does not lose any generality and this setting contributes to significantly simplified expositions. Recall that our input always satisfies Condition (1) in this model, where a feasible schedule allows job preemption but no migration.

Before introducing our online algorithm, we have some more notation. The dual-machine model has two machines: one of them is called M_P (a *primary machine*) and the other M_S (a *secondary machine*). We basically use the primary machine as far as possible and turn on the secondary machine if necessary. That is, we set M_P to be the main machine, and M_S to be the backup machine. When a job arrives, we always turn on M_P first and decide to turn on M_S only if it finds out that M_P is not able to finish all the pending jobs before their deadlines; i.e., M_P is overloaded. In other words, M_P and M_S are logical names and the two (physical) machines may swap their names in our algorithm. Let Q_{M_P} and Q_{M_S} denote the job queues for M_P and M_S, respectively, which are already scheduled using EDF; namely, the jobs in Q_{M_P} and Q_{M_S} are being executed continuously on M_P and M_S, respectively. We use Q to denote a queue for jobs not yet scheduled. If a job in Q_{M_P} or Q_{M_S} is finished, it is removed from the queue. Also, let $c'_j(t)$ denote the remaining execution time of job j at time t, and $W(t, t^\dagger) = \sum_{j \in Q(t,t^\dagger)} c'_j(t)$ denote the total remaining execution time in time interval $(t, t^\dagger)$, where $Q(t, t^\dagger)$ denotes the subset of jobs that have not yet finished their execution up to time t while their deadlines are not larger than $t^\dagger$, where $t^\dagger > t$. When a new job j', given by $(a_{j'}, d_{j'}, c_{j'})$, is arriving, there are two cases. One is that the current schedule for Q_{M_P} can accommodate its execution. If so, j' is inserted into Q_{M_P}, which is rescheduled by EDF (we say M_P is available). The other case is that there is no room for j' in Q_{M_P} (if jobs in Q_{M_P} would not have been delayed, this cannot happen because of the single machine schedulability assumption). We call such a j' *urgent*. If an urgent job comes, then M_S is turned on and j' is executed on M_S immediately (where M_P continues executing the already assigned jobs). We then swap M_P and M_S at this point of time, which means the original M_S can act as a new primary machine. Table 1 shows our algorithm, denoted by $\mathcal{A}$. Note that in the following, we use ALG to denote an online algorithm, OPT an offline optimal scheduler and CR a competitive ratio for simplicity.

Table 1 Algorithm $\mathcal{A}$.

Initially assign M_P and M_S to two machines arbitrarily.
The input satisfies Condition 1 of Lemma 1.
At any time t, $\mathcal{A}$ proceeds as follows:

1. Execution of jobs:
   - **a.** No machines are on.
     If there exist $t^\dagger$ and t^* such that $W(t^\dagger, t^*) \geq t^* - t^\dagger$ and $t \geq t^\dagger - u$, turn on one machine (M_P) and move all jobs in Q to Q_{M_P}.
   - **b.** Only M_P is on and a new job j comes.
     If there is no t^* such that $W(t, t^*) > t^* - t$ for jobs in $Q_{M_P} \cup \{j\}$ (i.e., M_P is available), add j to Q_{M_P} and reschedule it.
     Otherwise, turn on the other machine (M_S) and add j to Q_{M_S} that is empty.
     After that switch M_P and M_S.
   - **c.** M_S is on and a new job j comes.
     If M_S is available, add j into Q_{M_S} and reschedule it.
     Otherwise move it to Q_{M_P} (it is guaranteed that M_P is available).
2. Idle state:
   - **a.** M_S never has an idle state.
     That is, we immediately turn off M_S once Q_{M_S} becomes empty.
   - **b.** M_P becomes idle once Q_{M_P} becomes empty.
     We turn off M_P when its total idle time becomes $2/\psi_\sigma$.

Algorithm $\mathcal{A}$ has two key gadgets. First, we present a new notion of "*margin*", as a positive constant u, for delaying a job. That is, we start a job earlier than its due time by u. Fig. 1 shows examples to illustrate the idea. Here, we use a solid line segment from time a_j to d_j for each job $j = (a_j, d_j, c_j)$, and a gray box to represent its execution time c_j. Below those line segments and boxes, we illustrate how those jobs are executed by ALG and OPT. Dashed line segments show the idle time of ALG and dotted line segments the idle time of OPT (if any).

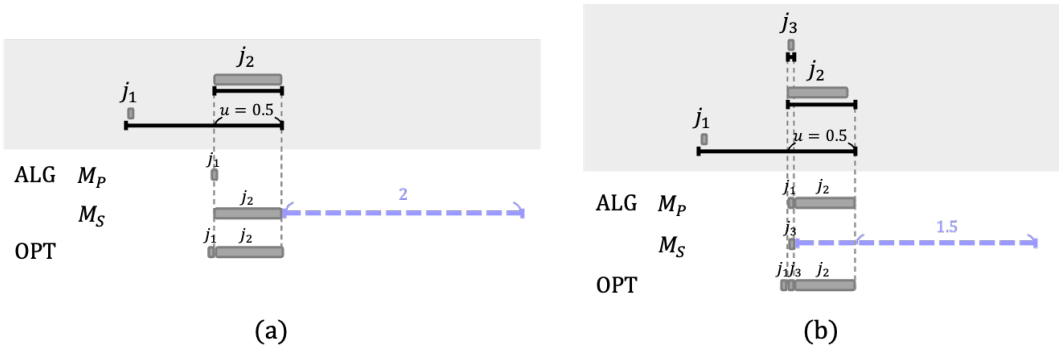


Figure 1 Two examples for illustrating the notion of margin.

As shown in Fig. 1 (a), j_1, having a tiny execution time of ϵ, starts at $d_1 - u - \epsilon$. Here we let $u = 0.5$ to get close to the optimal setting although its perfect setting seems hard. (We will show more details about the margin setting in the next section.) Thus, it is necessary that job j_2 that provokes M_S's turn-on has an execution time of at least 0.5 (otherwise it can interrupt j_1 and be inserted into M_P). See Fig. 1 (b) for a more complicated scenario. Now j_2 is a bit smaller than the one in (a) and can be inserted into the (old) M_P's execution gap caused by the margin u. Then a tiny j_3, which is urgent, makes the secondary machine ON, but its idle time can start 0.5 earlier than before. Note that the idle time of a machine is typically set to $\mathcal{B}$ after the last job is finished, where $\mathcal{B} = E/\psi_\sigma = 1$ since we are temporarily assuming $\psi_\sigma = 1$. It is actually a popular setting for the idle time that has been proved optimal in similar situations including the ski-rental problem. In contrast, the second key gadget of Algorithm $\mathcal{A}$ is that we set this value to twice, giving rise to that OPT has an interval of 1.5 from the end of j_2 (i.e. d_2) to the end of the idle state of M_P. On the other hand, recall that OPT has an interval of 2 in the previous scenario (see Fig. 1 (a)). One can observe that OPT could use this interval to execute the pending jobs and thus the longer the better for this interval. However, we have to use 1.5 instead of 2 for the analysis in Section 3.

Tight Analysis of Algorithm S [7]. The two gadgets of Algorithm $\mathcal{A}$ reveal the improvement of the upper bound in some sense. Here we recall Chen et al.'s 4-competitive algorithm, called *Algorithm* ***S***, and conduct tight analysis of their algorithm to clarify the merit of our proposed gadgets. That is, we present a tight example to show its worst competitive performance.

Basically, the main structure of Algorithm **S** is similar, but they let every job j be associated with a parameter $h_j = \max\{a_j, d_j - \lambda\mathcal{B}\}$, i.e. its *energy-efficient anchor*, to determine when the main machine M_P should be switched on for a pending job j, where λ is a constant (setting to be one in [7]). Set $\psi_b = \psi_\sigma = 1$, and $E = \mathcal{B} = k$, where k is a sufficiently large value, for simplicity. As shown in Fig. 2, we let $c_1 = c_3 = \mathcal{B}$ and $c_2 = c_4 = \epsilon$. Algorithm **S** turns on M_P at h_1. Since j_2 arrives at the same time, i.e. $a_2 = h_1$, obviously,

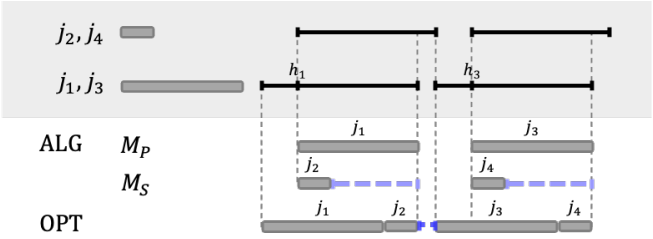


Figure 2 Tight example for Algorithm **S** [7].

M_P cannot finish j_2 before its deadline. Therefore, M_S is switched on to execute j_2. One can observe that the design of the margin in Algorithm $\mathcal{A}$ can provide the flexibility to escape from the scenario.

Next, the turn-off strategy of Algorithm **S** is as follows: if both machines are ON, M_P (new M_S) is turned off as soon as it becomes idle. Otherwise, if only one of the machines is ON, then when the machine becomes idle, it is switched off once the length between the current time and the moment when M_P was turned on has reached $\mathcal{B}$. In this example, M_S remains idle until M_P keeps ON for $\mathcal{B}$ units of time, resulting in M_P and M_S being switched off at the same time. One can observe that if the job executed on M_S is tiny, the malicious adversary can punish the turn-off strategy since it turns off M_S too early. We design the same worst-scenario for jobs j_3 and j_4 and consider the performance. As a result, the total energy cost of ALG is $4E + 4\mathcal{B} - 2\epsilon = (4k - \epsilon) \cdot 2$. By contrast, in the offline optimal solution, it turns on a single machine at a_1 and keep it ON until all the input jobs are finished. The minimum energy cost is $E + 2\mathcal{B} + 2\epsilon + \epsilon' = k + 2(k + \epsilon) + \epsilon'$.

It is not hard to see that, if we consider the above case of four jobs to be one round, and the energy cost of Algorithm **S** for $\frac{r}{2}$ rounds, where r is a multiple of 2, is $(4k - \epsilon) \cdot r$, while the energy cost of OPT is $k + r(k + \epsilon) + (r - 1) \cdot \epsilon'$. Letting the values of ϵ and ϵ' be very small, the ratio becomes

$$\frac{4kr}{k + kr} = \frac{k \cdot 4r}{k \cdot (r + 1)} = \frac{4r}{r + 1}.$$

When the value of r is sufficiently large, the CR approaches 4. The tight analysis reveals the advantages of our designed gadgets in Algorithm $\mathcal{A}$.

3 Analysis of the Algorithm

For analysis, we need to show that (i) $\mathcal{A}$ is correct, namely it can execute any sequence of jobs arriving under the single machine schedulability condition and (ii) its CR is at most 3. We mostly focus on (ii). (i) is not hard and the following observation should be enough to see every job is executed:

(1) Suppose when a new job j comes at time t, no machine is ON. Then if the condition of 1-a for $Q \cup j$ is not met, j just goes to queue Q. Once j enters Q, it must be executed eventually by 1-a. If the condition of 1-a is met, then all the jobs in Q go to Q_{M_P} and we go to 1-b. Here j is inserted to Q_{M_P} (if possible) or is executed on M_S that is turned on at t. Thus j is executed.

(2) Suppose only M_P is already ON but M_S is OFF when j comes. As (1), j is inserted to Q_{M_P} or executed on M_S that is turned on at t.

(3) Suppose M_S is ON when j comes. If M_S is available, j is executed on it. Otherwise, note that when M_S is turned on for the last time, all the jobs in Q go to M_S (recall M_P and M_S are swapped) and while M_S is on, all newly coming jobs are executed on M_P due to EDF. This execution is possible since if the whole input sequence satisfies Condition 1 of Lemma 1, its arbitrary suffix obviously does, too. j is one of them and must be executed.

To prove the CR, we first define a *phase*. Suppose the system changes its state from both machines OFF to at least one machine ON at t_1 and returns to the state of both machines OFF at t_e. Also let $a_1, a_2, \ldots, a_k$ be the arrival time of the jobs executed during t_1 to t_e and let t_0 be $\min\{a_1, \ldots, a_k\}$. Then the time slot from t_0 to t_e is called a phase. Note that an entire execution of $\mathcal{A}$ consists of some phases. Let $P(i)$ be the i'th phase.

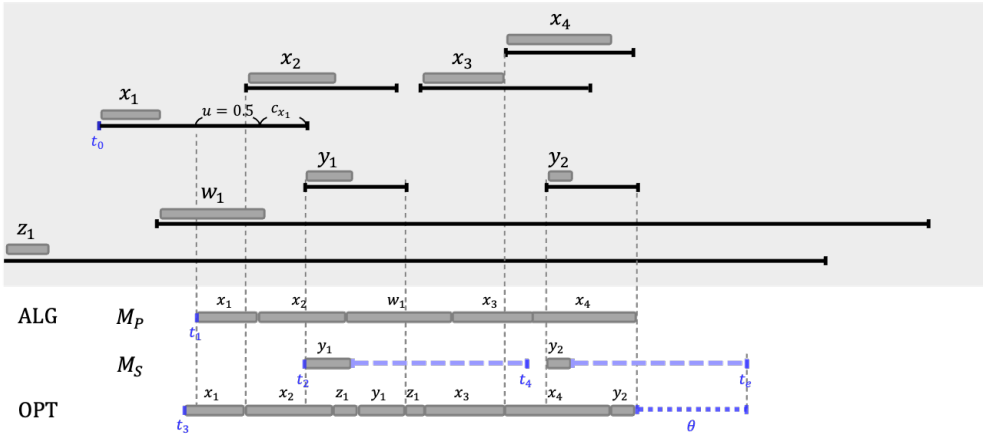


Figure 3 A single phase. Now a period of idle time is $2/\psi_\sigma$. M_P and M_S are swapped at a_{y_1}.

Fig. 3 illustrates how a single phase looks like. x_1 and x_2 are executed on M_P, then an urgent y_1 comes and executed on the new M_P (so w_1, x_3 and x_4 are executed on the new M_S). w_1 is a job moved from Q to Q_{M_P} when x_1 starts (more precisely, w_1 is scheduled after x_1, but x_2 is inserted later due to the EDF principle). x_3 and x_4 are added to Q_{M_S}. y_2 is a job that cannot be inserted to Q_{M_S} because of the existence of x_4; it goes to M_P. Thus M_P can be OFF for some period of time during a single phase (but M_S should be ON during that period by definition). The earliest arrival time of the jobs $\mathcal{A}$ executes in this phase is that of x_1, which is the start time, t_0, of this phase. Note that t_e at which the phase ends is the moment when the idle time after y_2 expires. An example of the OPT's execution sequence is given at the bottom of the figure. It must execute x_1, x_2, x_3, x_4, y_1 and y_2 since each of them has its arrival time and deadline within the phase. w_1 is not executed by OPT in this example. By contrast, z_1 which is performed by OPT is not executed by $\mathcal{A}$ (can be executed in the previous phase), so its arrival time is not counted to determine the beginning of the phase. (Note that OPT and $\mathcal{A}$ as well, may execute each job separately, so we have two z_1's in the figure.) Fig. 3 is also used for the later analysis of the competitive ratio. A phase is called a single-machine phase if only one machine (M_P) is ON in that phase and a dual-machine phase otherwise.

Our proof of the CR uses a math induction. Since $\mathcal{A}$ is deterministic, the set of jobs that are executed by $\mathcal{A}$ in $P(i)$ is uniquely determined once the input is given, which we denote by $J_A(i)$. For the jobs executed by OPT in $P(i)$, the situation is less clear; jobs whose arrival time and deadline are both within $P(i)$ must be executed, but jobs such that only

one of them is within $P(i)$ may or may not be executed even partially. Furthermore, OPT may execute some jobs outside phases, namely while no machine is ON. Fix an arbitrary execution sequence $S(i)$ of OPT in this phase. Then we define the following parameters used in the induction.

(1) $\alpha(i)$ ($\lambda(i)$, and $\delta(i)$, resp.) = the total execution time of the jobs executed by both $\mathcal{A}$ and OPT (only by $\mathcal{A}$ and only by OPT possibly partially, resp.)

(2) $A(i)$ is the cost of $\mathcal{A}$ in $P(i)$ that includes $\alpha(i)$, $\lambda(i)$, turn-on costs and idle costs.

(3) $O_f(i)$ is a lower bound for the cost of OPT, namely it includes $\alpha(i)$, $\delta(i)$ and idle costs if any. Note that it does not include the turn-on cost when $S(i)$ starts since OPT may not need it depending on its state at the end of the previous phase, but does include one(s) if $S(i)$ includes turn-off and turn-on in the middle of the phase. $O_n(i)$ is similar but we impose the condition that OPT is ON at the end of $P(i)$.

For instance, consider $P(i)$ whose execution sequence looks like Fig. 1 (a). Then $A(i) = 4.5$, $O_f(i) = 0.5$ and $O_n(i) = 2.5$, where 0.5 is for the execution of jobs and $2 = (2/\psi_\sigma)\psi_\sigma$ is the idle cost to keep it ON until the end of the phase.

▶ **Theorem 2.** *$\mathcal{A}$ is correct and its CR is at most 3 for $u = 0.5$.*

Proof. We omit the first part (see the previous observation). For the CR, we fix an arbitrary execution sequence S ($S(i)$ is its subsequence associated with $P(i)$) for the entire execution sequence of OPT and prove two lemmas.

▶ **Lemma 3.** *The following (2) and (3) hold for each phase for $r = 3$.*

$$rO_f(i) - A(i) \geq \delta(i) - \lambda(i) - r. \tag{2}$$

$$rO_n(i) - A(i) \geq \delta(i) - \lambda(i). \tag{3}$$

The proof will be given later. Let $O(i)$ be the (real) cost of OPT under the sequence $S(i)$ in $P(i)$. Also let m be an integer less than or equal to the number of phases.

▶ **Lemma 4.** *For $r = 3$, we have*

$$\sum_{i=1}^{m} (rO(i) - A(i)) \geq \begin{cases} \sum_{i=1}^{m} (\delta(i) - \lambda(i)) & \text{if OPT is OFF at the end of } P(m). \quad (6) \\ \sum_{i=1}^{m} (\delta(i) - \lambda(i)) + r & \text{if OPT is ON at the end of } P(m). \quad (7) \end{cases}$$

Proof. Suppose OPT is OFF at the end of $P(i)$. Then $O(i)$ should be at least $O_f(i)$ since the latter is a lower bound and similarly for $O_n(i)$ if OPT is ON at the end of the phase. For $m = 1$, note that OPT must spend the turn-on cost of 1 that is not included in either $O_f(i)$ or $O_n(i)$. Therefore if OPT is OFF at the end of the phase, we have

$$rO(1) - A(1) \geq rO_f(1) + r - A(1) \geq \delta(1) - \lambda(1)$$

by Lemma 3. Otherwise, if OPT is ON at the end of the phase, we have

$$rO(1) - A(1) \geq rO_n(1) + r - A(1) \geq \delta(1) - \lambda(1) + r$$

similarly. Now suppose the lemma is true for $m' = m - 1$. Then to prove that the lemma also holds for $m' = m$, we consider four cases and define $C \to D$ as the states of OPT at the end of $P(m-1)$ and $P(m)$ respectively: (i) OFF $\to$ OFF, (ii) OFF $\to$ ON, (iii) ON $\to$ OFF and (iv) ON $\to$ ON. For case (i), OPT must pay the turn-on cost in $P(m)$, so

$$\begin{aligned}\sum_{i=1}^{m}(rO(i)-A(i)) &= \sum_{i=1}^{m-1}(rO(i)-A(i)) + rO(m) - A(m)\\ &\geq \sum_{i=1}^{m-1}(\delta(i)-\lambda(i)) + 0\cdot r + rO_f(m) - A(m) + 1\cdot r\\ &\geq \sum_{i=1}^{m-1}(\delta(i)-\lambda(i)) + \delta(m) - \lambda(m)\\ &= \sum_{i=1}^{m}(\delta(i)-\lambda(i))\end{aligned}$$

by the induction hypothesis and Lemma 3. Here 0 and 1 before r are for handling the four cases. Since the current case is OFF $\rightarrow$ OFF, the first 0 means formula (4) does not have r on the right-hand side and the second 1 means that OPT must turn on in $P(m)$. The other cases are similar (just 0 and 1 before r change) and may be omitted. ◀

If there are m phases in total, it must be that $\sum_{i=1}^{m}(\delta(i)-\lambda(i)) = 0$ and thus the theorem is proved. What remains is to prove Lemma 3.

Proof. (Proof of Lemma 3) Suppose $x_1 = (a_1, d_1, c_1)$ is the first job executed in some phase P starting from t_0 and ending at t_e. Then since x_1 is executed in P, $t_0 \leq a_1$ (there may be another job executed in P and having an earlier arrival time). Also, since the ending time of x_1's execution is to be $d_1 - 0.5$ due to the delay (or not delayed at all if this amount of delay is impossible) and we have a mandatory idle time, $2/\psi_\sigma \geq 2$, after its (or a later job's) execution, it must be that $d_1 \leq t_k$. This means the period (a_1, d_1) is included in P, meaning OPT also executes x_1 in P. Thus, in each phase, both $\mathcal{A}$ and OPT execute at least one job. Also it turns out, by definition, that $\mathcal{A}$ never executes a job outside phases.

Note, however, that OPT may execute some job, say x, outside phases (this happens, e.g., if u_1 in Fig. 4 is executed by OPT after this phase and before the next phase). If that happens, we consider that x is executed in a "special" phase, $P(i')$, by extending the definition of a phase. Note that this phase has $A(i) = \lambda(i) = 0$ and both $O_f(i)$ and $O_n(i)$ are at least $\delta(i)$, so (2) and (3) obviously hold. A special phase may continue to/from a neighboring (normal) phase.

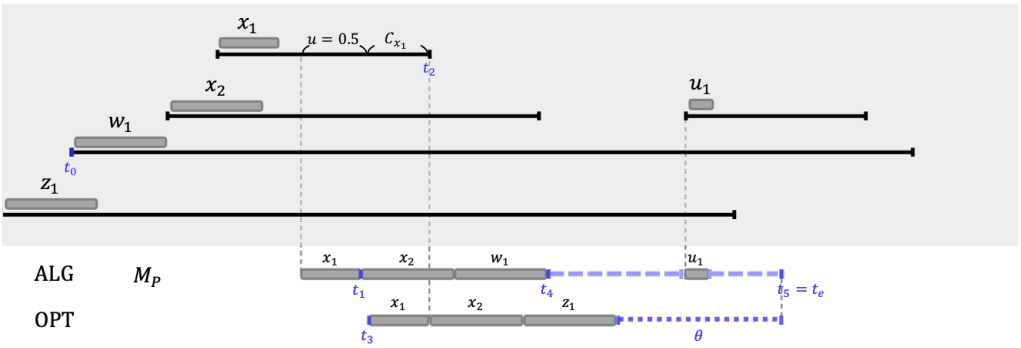


Figure 4 A single-machine phase. Recall a period of idle time of $\mathcal{A}$ is $2/\psi_\sigma$.

We first prove the lemma for a single-machine phase $P(i)$. See Fig. 4 which illustrates the execution sequences of $\mathcal{A}$ and OPT as with Fig. 3. In this figure, the phase starts at the arrival time of w_1 and ends when ALG's machine (M_P) turns off. x_1 is the first job executed in this phase, and x_2 is a job whose arrival time and deadline are both within this phase (so must be executed by both $\mathcal{A}$ and OPT in this phase). z_1 has only its deadline, and u_1 and w_1 have only their arrival times within the phase. Thus there are several different types of jobs, but what is important is whether or not each job is executed in this phase. In the example of this figure, x_1 and x_2, called *type-AO* jobs, are executed by both $\mathcal{A}$ and OPT, w_1 and u_1, *type-A*, by only $\mathcal{A}$, and z_1, *type-O*, only by OPT. Each type can include an arbitrary number of jobs, but as will be seen in a moment, those numbers are not important. So we will progress our analysis by using these five jobs, x_1, x_2, w_1, u_1, z_1, for a while and will mention the generalization after this analysis. It should also be noted that a single job may be executed by OPT in two or more phases. If this happens, we divide that job into two or more parts and allow each part can have a different job type, if necessary.

Set the following moments (see the figure) of time: t_1 and t_2: the ending time of execution and the deadline of x_1, respectively. t_3: the time when OPT becomes ON after time t_0 to execute x_1 or maybe another job. t_4 and t_5: the start and ending times of the idle state of $\mathcal{A}$, respectively. Thus one can observe that neither the number of jobs in each type nor their execution sequence is important to define these times, except x_1 that is first executed. For the five jobs, it turns out that $A(i) = c_{x_1} + c_{x_2} + c_{w_1} + c_{u_1} + 1 + 2$ (1 is the turn-on cost and 2 is the idle cost; recall once $\mathcal{A}$ enters an idle state, it continues until its total time becomes $2/\psi_\sigma$, i.e., until its total idle cost becomes $(2/\psi_\sigma)\psi_\sigma = 2$), and $O_f(i) \geq (c_{x_1} + c_{x_2} + c_{z_1})$. OPT's execution of some job may exceed the border of the phase. If that happens, as mentioned before, we can partition that job into two parts, the first one ends at the end of the phase and the second one is to be the remaining part, which is executed in the following special phase.

Now we have

$$\begin{aligned} rO_f(i) - A(i) &\geq 3(c_{x_1} + c_{x_2} + c_{z_1}) - (c_{x_1} + c_{x_2} + c_{w_1} + c_{u_1} + 3) \\ &= 2(c_{x_1} + c_{x_2} + c_{z_1}) + c_{z_1} - (c_{w_1} + c_{u_1}) - 3 \\ &\geq c_{z_1} - (c_{w_1} + c_{u_1}) - 3. \end{aligned}$$

Thus (2) holds for any nonnegative values of c_{x_1} through c_{u_1}. Similarly, letting θ be the idle time of OPT (if any) that keeps the OPT's machine ON until t_5 (OPT can turn off once and turn on at or before t_5, but OPT would then need an extra turn-on cost and our proof becomes easier), we have $O_n(i) \geq c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta$ and hence

$$\begin{aligned} rO_n(i) - A(i) &\geq 3(c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta) - (c_{x_1} + c_{x_2} + c_{w_1} + c_{u_1} + 3) \\ &\geq 2(c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta) + c_{z_1} - (c_{w_1} + c_{u_1}) - 3. \end{aligned}$$

Here we have the following claim.

▷ Claim 5.

$$c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta \geq 1.5$$

Proof. We have a sequence of time relations:

(i) $t_2 - t_1 \leq 0.5$ due to the margin,
(ii) $t_3 \leq t_2$ (x_1 must be executed before its deadline),
(iii) $t_1 \leq t_4$ (obviously),

(iv) $t_3 - t_4 \leq t_3 - t_1$ (by (iii)) $\leq t_2 - t_1$ (by (ii)) ≤ 0.5 (by (i)),
(v) $t_5 - t_4 \geq 2/\psi_\sigma$ (the total idle time),
(vi) $t_4 - t_5 \leq -2/\psi_\sigma$ (inversion of (v)), and
(vii) $c_{x_1} + c_{x_2} + c_{z_1} + \theta = t_5 - t_3 = -(t_3 - t_5) \geq 2/\psi_\sigma - 0.5$ (by (iv)+(vi)).

We may need a bit more explanation for (i). Recall that if x_1 is delayed and nothing happens, $t_2 - t_1$ is exactly 0.5. However, some (small) job that comes after x_1's execution has started may interrupt x_1 and be inserted. It should also be noted that if $d_{x_1} - (a_{x_1} + c_{x_1}) < 0.5$, then x_1 is not delayed in the first place. Now we have from (vii) (note $\psi_\sigma \leq 1$)

$$c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta \geq \psi_\sigma(c_{x_1} + c_{x_2} + c_{z_1} + \theta) \geq 2 - 0.5\psi_\sigma \geq 1.5. \qquad \lhd$$

Thus we have

$$rO_n(i) - A(i) \geq 2 \times 1.5 + c_{z_1} - (c_{w_1} + c_{u_1}) - 3 = c_{z_1} - (c_{w_1} + c_{u_1}),$$

meaning (3) is also true. In general, we have more (or less) jobs in each job type. However, one can see that the definitions of t_1 through t_5 are not affected by that (only the first job, x_1 is important). Also we can simply replace c_{x_1} by the sum of execution times of type-AO jobs and similarly for type-A and type-O jobs. Thus the extension to the general case is straightforward and details may be omitted.

We next consider a dual-machine phase as shown in Fig. 3, where there are two executions, y_1 and y_2 on M_P while M_S is busy. (Recall M_P and M_S are swapped when M_S turns on. Now M_S is busy and it may not be available for urgent jobs.) Let the first execution be E_1 and the second one E_2. We first consider the case that E_2 does not exist. Namely, there is no x_3, x_4 or y_2 and OPT has an idle time after having executed z_1. Thus the phase would be ending at the end of the idle state following the execution of y_1 and we prove the lemma for this case first. As before, we use specific examples for jobs to be executed in this phase, x_1, x_2 and y_1 for type-AO, w_1 for type-A and z_1 for type-O. Since the M_S's turn-on is provoked, there must be a set S of jobs such that their execution on M_P is impossible during the period from some t'_1 to some t'_2. We assume that $S = \{x_1, x_2, y_1\}$, where $t'_1 = t_1$ and t'_2 is the deadline of y_1. What we do for the generalization is the same as before, namely we replace $\{x_1, x_2, y_1\}$ by the real jobs in S, maybe more jobs for type-AO, replace w_1 by real type-A jobs, and z_1 by real type-O jobs.

Now we start with definitions of time moments (see the figure) as before. t_1, t_2 and t_3: the time when M_P and M_S become on and the time OPT becomes busy, respectively. t_4: the time when the idle state of M_P is ended and this is the end of the phase, too. For those five jobs, we have $A(i) = c_{x_1} + c_{x_2} + c_{y_1} + c_{w_1} + 2 + 2$ (we now need to turn on both M_P and M_S), $O_f(i) \geq (c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1})$, and $O_n(i) \geq (c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \psi_\sigma\theta)$, where θ is the idle time (if any) to keep the OPT's machine on until t_4.

To prove formulas (2), we have (recall $\psi_\sigma \leq 1$)

$$\begin{aligned} rO_f(i) - A(i) &\geq 3(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1}) - (c_{x_1} + c_{x_2} + c_{y_1} + c_{w_1} + 4.0) \\ &= 2(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1}) + c_{z_1} - c_{w_1} - 4.0. \end{aligned}$$

Here we can claim that $c_{x_1} + c_{x_2} + c_{y_1} \geq 0.5$. The reason is that there is a margin of 0.5 between the end of the execution of x_1 and its deadline (recall again if realizing this margin is impossible, M_S would not have been turned on). So if $c_{x_1} + c_{x_2} + c_{y_1} < 0.5$, then it follows of course $c_{x_2} + c_{y_1} < 0.5$, which means that x_2 and y_1 could have been executed using this margin time (they can interrupt the execution of x_1) on M_P, resulting in a contradiction. Thus $rO_f(i) - A(i) \geq 2 \times 0.5 + c_{z_1} - c_{w_1} - 4.0 = c_{z_1} - c_{w_1} - 3$ and we are done for (2).

For formula (3), we have ($\psi_\sigma \leq 1$)

$$\begin{aligned} rO_n(i) - A(i) &\geq 3(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \psi_\sigma\theta) - (c_{x_1} + c_{x_2} + c_{y_1} + c_{w_1} + 4.0) \\ &= 2((c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1}) + 3\psi_\sigma\theta + c_{z_1} - c_{w_1} - 4.0 \\ &\geq 2\psi_\sigma(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \theta) + c_{z_1} - c_{w_1} - 4.0. \end{aligned}$$

We again have the following time relations:

(i) $t_1 < t_2$ (M_S never turns on before M_P),

(ii) $t_3 < t_1$ (see below),

(iii) $t_4 - t_3 \geq t_4 - t_1$ (by (ii)) $\geq t_4 - t_2$ (by (i)) $\geq 2/\psi_\sigma$.

For (ii) recall that M_P cannot execute x_1, x_2 and y_1 from time t_1. Since OPT does execute those jobs by a single machine, it should have started their execution before t_1, meaning $t_3 < t_1$. Since $c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \theta = t_4 - t_3$, we finally have

$$rO_n(i) - A(i) \geq 2\psi_\sigma(2/\psi_\sigma) + c_{z_1} - c_{w_1} - 4.0 = c_{z_1} - c_{w_1},$$

and (3) is proved. The generalization to arbitrary number of jobs is the same as before and may be omitted.

Finally we consider the case that E_2 (or even more) exists, which can be simply done by considering that a new virtual phase, just an interval starting from t_4 and ending at t_e where we do the similar calculation as above. Note that $\mathcal{A}$ needs only one turn-on cost and so the energy consumption of the new virtual phase is the job execution costs $+3$ instead of $+4$ above. Therefore we do not need to lower bound the cost for executing type-AO jobs (as we did for $c_{x_1} + c_{x_2} + c_{y_1}$ above). Also, since the new virtual phase obviously includes the whole idle time of M_S, the proof for formula (3) is also straightforward. Details may be omitted.

Thus Lemma 3 is proved. ◀

And the proof of Theorem 2 is also concluded. ◀

4 Lower Bound

In this section we give our second result, a lower bound of 2.1, which improves 2.06 obtained in [7]. Throughout this section we set $\psi_\sigma = 1$ (which seems the worst case for online algorithms).

▶ **Theorem 6.** *The CR of any online algorithm for the online DPM job scheduling problem is at least 2.1.*

Proof. Our strategy is quite simple and standard. The adversary, Adv, gives requests, one by one, so that each request blames the last action of the algorithm. Fix an arbitrary algorithm ALG and a target CR lower bound, α, we want to prove. Before the formal proof, we briefly look at the basic strategy of Adv. Recall that $\psi_\sigma = 1$ in this proof.

The first request by Adv is $j_1 = (0, d_1, c_1)$, where its execution time c_1 is tiny and d_1 should not be too small. ALG must execute j_1 at some time before d_1, say at $d_1 - x_1$ on one of the two machines, say M_1 (Fig. 5). Here we have two cases.

Suppose that x_1 is relatively large. Then Adv sees how long ALG stays in the idle state after the execution of j_1. We can assume without loss of generality that this idle state continues at least until d_1 and may be more for additional y_1 as shown in Fig. 5. (If the idle state ends before d_1, then Adv immediately gives another tiny request with the same deadline d_1. This situation is similar to that ALG postpones its execution of j_1 up to this

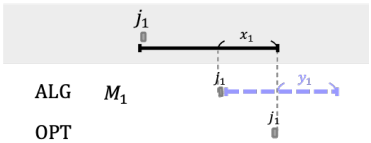


Figure 5 Illustration of the incoming job j_1.

time. Since ALG has already spent a turn-on cost of 1, it is not hard to show that Adv's job becomes easier. We omit details.) From the OPT side, it suffices to execute a tiny c_1 at d_1. Here ALG cannot have a long y_1 since the CR at this moment is $\frac{1+c_1+x_1+y_1}{1+c_1}$ (both ALG and OPT need a turn-on cost of 1, since this is the beginning of the game), which may exceed α and the game would end. So y_1 is relatively small, for which Adv gives a similar request right after the idle time expires. As shown in Fig. 6, OPT can manage these two requests by being ON from d_1 to a_2, thus blaming the small value of y_1.

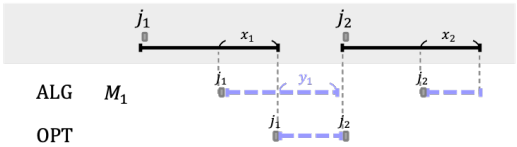


Figure 6 Illustration of the incoming job j_2.

What if x_1 is relatively small? Then Adv gives a request j_2 as shown in Fig. 7 immediately after ALG has started the execution of j_1. Note that j_2 has the same deadline as j_1 (i.e. $d_1 = d_2$) and its execution time is x_1 (i.e. $c_2 = x_1$). Thus ALG cannot execute j_2 on M_1 and it turns on M_2 meaning ALG has to pay a new turn-on cost. OPT can manage this by being on from slightly before $d_1 - x_1$ to d_1, thus blaming the shortness of x_1.

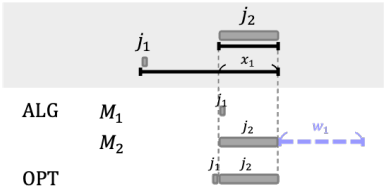


Figure 7 Case A with j_1 and j_2.

Now we start our formal proof. As mentioned above, Adv has basically two different strategies depending on the first response of ALG, namely $x_1 \leq \beta$ (Case A) and $x_1 > \beta$ (Case B), where β is some constant to be optimized later. Let us look at Case A first. As shown above, it proceeds to the situation illustrated in Fig. 7 and now ALG selects its new idle time w_1 on M_2. Of course M_1 can also have some idle time. However, as seen in a moment, our Adv always gives a next request after both machines become OFF. Therefore without loss of generality, we can assume only one machine which is busy until later than the other enters an idle state and the other turns off immediately when it finishes all the assigned requests. At the moment of Fig. 7, the CR is $\frac{2+c_1+x_1+w_1}{1+c_1+x_1}$ (recall $\psi_\sigma = 1$). Here we set $c_1 = 0$ for the exposition (and will do the same for a tiny execution time in the remaining part, too). This does not lose much sense since we can make c_1 arbitrarily small and it always appears

as a sum with far greater values. Thus our current CR is $\frac{2+x_1+w_1}{1+x_1}$. If this value is greater than α, Adv has achieved its goal and the game ends. For the game to continue it must be $\frac{2+x_1+w_1}{1+x_1} \leq \alpha$. This implies

$$w_1 \leq \alpha(1+x_1) - (2+x_1) \leq \alpha + (\alpha-1)x_1 - 2 \leq \alpha + (\alpha-1)\beta - 2 \ (= f_1)$$

since $\alpha \geq 1$ and $x_1 \leq \beta$. Let f_1 be the value of the right-hand side.

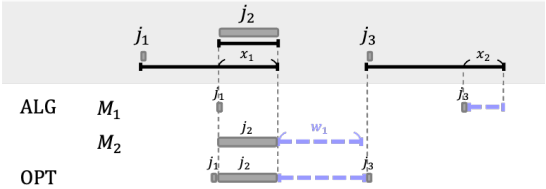


Figure 8 Case A with j_1, j_2 and j_3.

Next, the adversary releases another tiny request j_3 after the idle period of M_2. See Fig. 8. At this moment, both M_1 and M_2 are OFF, so we can use M_1 without loss of generality for j_3. Similarly as before, ALG executes j_3 at $d_3 - x_2$ and has an idle time of x_2. OPT executes j_3 immediately when it comes by keeping its idle state for w_1 assuming that $w_1 \leq 1$, which can be verified after we eventually fix all parameter values (indeed, $w_1 \leq 0.63197$ for our final setting of the parameters). Note that ALG needs three turn-ons and OPT one, so the current CR is $\frac{3+x_1+w_1+x_2}{1+x_1+w_1}$. For the game to be continued, it must be

$$x_2 \leq \alpha(1+x_1+w_1) - (3+x_1+w_1) \leq \alpha - 3 + (\alpha-1)\beta + (\alpha-1)w_1 \leq f_2 + (\alpha-1)w_1$$

by letting $\alpha - 3 + (\alpha-1)\beta = f_2$.

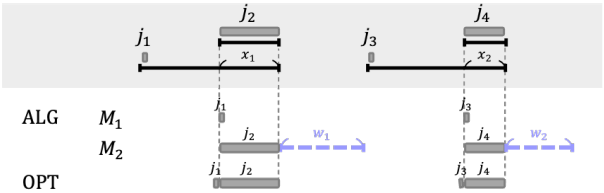


Figure 9 Case A with j_1 to j_4.

The adversary then releases j_4 exactly as it did for j_2 as shown in Fig. 9. Here, it is better for OPT to turn off at d_2 and turn on at a_4 since Adv selects a large $d_3 - a_3$. The CR is $\frac{4+x_1+w_1+x_2+w_2}{2+x_1+x_2}$. For the game to continue, we must have

$$\begin{aligned}
w_2 &\leq \alpha(2+x_1+x_2) - (4+x_1+w_1+x_2) \\
&= 2\alpha - 4 + (\alpha-1)x_1 + (\alpha-1)x_2 - w_1 \\
&\leq 2\alpha - 4 + (\alpha-1)\beta + (\alpha-1)(f_2 + (\alpha-1)w_1) - w_1 \\
&= 2\alpha - 4 + (\alpha-1)\beta + (\alpha-1)f_2 + (\alpha-1)^2 w_1 - w_1 \\
&= 2\alpha - 4 + (\alpha-1)(\beta+f_2) + (\alpha^2 - 2\alpha)w_1 \\
&= f_3 + (\alpha^2 - 2\alpha)w_1
\end{aligned}$$

by letting $f_3 = 2\alpha - 4 + (\alpha-1)(\beta+f_2)$.

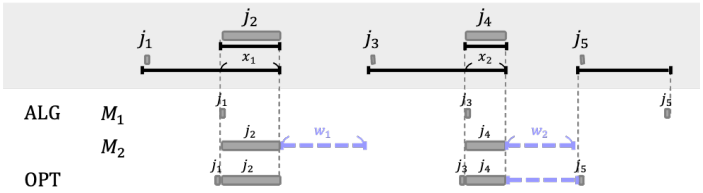


Figure 10 Case A ending with j_5.

Finally, the adversary releases a tiny request j_5 as shown in Fig. 10. For ALG, we impose only a turn-on cost which ALG at least spends. OPT can manage this request by continuing its idle state as before. Now the CR is at least

$$\begin{aligned}\frac{5+x_1+w_1+x_2+w_2}{2+x_1+x_2+w_2} &\geq \frac{5+\beta+w_1+f_2+(\alpha-1)w_1+f_3+(\alpha^2-2\alpha)w_1}{2+\beta+f_2+(\alpha-1)w_1+f_3+(\alpha^2-2\alpha)w_1}\\ &= \frac{5+\beta+f_2+f_3+(\alpha^2-\alpha)w_1}{2+\beta+f_2+f_3+(\alpha^2-\alpha-1)w_1}\\ &\geq \frac{5+\beta+f_2+f_3+(\alpha^2-\alpha)f_1}{2+\beta+f_2+f_3+(\alpha^2-\alpha-1)f_1} = C_A.\end{aligned}$$

The first inequality holds for the following reason: x_1, x_2 and w_2 appear both in the numerator and the denominator, the fraction becomes minimum when all x_1, x_2 and w_2 are maximum. For the second inequality, recall that our target CR, α, is greater than 2. So $\frac{(\alpha^2-\alpha)}{\alpha^2-\alpha-1} \leq 2$, which means the fraction becomes minimum when w_1 is maximum. Thus the CR is at least C_A for Case A.

Case B is similar and we can prove that the CR is at least

$$C_B = \frac{(\alpha^2-\alpha)g_1+4+\beta+g_2+g_3}{(\alpha^2-\alpha-1)g_1+2+g_2+g_3},$$

where $g_1 = \alpha-1-\beta$, $g_2 = \alpha-2-\beta$, and $g_3 = (\alpha-1)g_2+2\alpha-3-\beta$ (see [17] for more details).

For $\beta = 0.4745$ and $\alpha = 2.1068$, our numerical calculation shows that $C_A = 2.107447$ and $C_B = 2.106989$, and the theorem is proved. ◀

5 Concluding Remarks

Obvious future work is to narrow the gap. For the lower bound, one can easily notice that increasing the number of stages (currently, it is three) might help. This is correct and in fact one more stage can give us a strictly better bound. Unfortunately the degree of improvement is already pretty small and getting even smaller in further stages. For the upper bound, it is obviously important to consider more flexible structures for delaying requests. Our present algorithm executes all the pending requests when one request comes to its due time. It would be even more important to make our CR a function in ψ_σ. Our open question is that a smaller ψ_σ very likely implies a better CR.

References

1 Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms (TALG)*, 10(2):9, 2014.

2 Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration. *Journal of Computer and System Sciences*, 81(7):1194–1209, 2015.

3 Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2758–2769. SIAM, 2020.

4 Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1102–1113. SIAM, 2015.

5 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):3, 2007.

6 Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE transactions on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.

7 Jian-Jia Chen, Mong-Jen Kao, DT Lee, Ignaz Rutter, and Dorothea Wagner. Online dynamic power management with hard real-time guarantees. *Theoretical Computer Science*, 595:46–64, 2015.

8 Jian-Jia Chen and Tei-Wei Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. *ACM SIGPLAN Notices*, 41(7):153–162, 2006.

9 Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 289–294. IEEE, 2007.

10 Houssine Chetto and Maryline Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Information Processing Letters*, 30(4):177–184, 1989.

11 Erik D Demaine, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, Amin S Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. *Journal of Scheduling*, 16(2):151–160, 2013.

12 Sandy Irani and Anna R. Karlin. *Online Computation*, pages 521–564. PWS Publishing Co., USA, 1996.

13 Sandy Irani and Kirk R Pruhs. Algorithmic problems in power management. *ACM Sigact News*, 36(2):63–76, 2005.

14 Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):325–346, 2003.

15 Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)*, 3(4):41, 2007.

16 Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

17 Ya-Chun Liang, Kazuo Iwama, and Chung-Shou Liao. Improving the bounds of the online dynamic power management problem, 2022. `arXiv:2209.12021`.

18 Adam Wierman, Lachlan Leicester Henry Andrew, and Minghong Lin. *Speed scaling: An algorithmic perspective*, pages 385–406. CRC Press, 2012.

19 Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382. IEEE, 1995.

Integer Complexity and Mixed Binary-Ternary Representation

Kazuyuki Amano
Gunma University, Kiryu, Japan

Abstract

The integer complexity of a natural number n, denoted by $\|n\|$, is the smallest number of 1's needed to express n using an arbitrary combination of addition and multiplication (and parentheses). For example, $\|6\| = 5$ since the expression $6 = (1+1)\cdot(1+1+1)$ contains five 1's and there are no such expressions containing at most four 1's. The investigation of this cute complexity measure was originated by Mahler and Popken in the 1950s. It is easy to see that $\|n\|/\log_3 n \in [3, 3\log_2 3]$ ($\sim [3, 4.755]$) for every n, but the distribution of $\|n\|$ is largely unknown.

In this work, we focus on the restricted expressions obtained by applying Horner's schema to a *mixed binary-ternary representation* of a given number in which we can arrange base-two and base-three digits in an arbitrary order. Let $f(n)$ denote the minimum number of 1's needed to express n in this way. Apparently, $f(n) \geq \|n\|$ for every n. We extensively investigate on $f(n)$ via the combination of computer experiments and theoretical analysis and obtain the following set of results: (i) Computer experiments supporting the hypothesis that $f(n)/\log_3 n < 3.483$ on average and $f(n)/\log_3 n < 4.212$ for all n, (ii) For almost all natural numbers n, $3.120 < f(n)/\log_3 n < 3.587$, and (iii) There are infinitely many n's such that $f(n)/\log_3 n > 3.934$. Several new bounds on the original integer complexity are also presented in the paper.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Integer complexity, Lower bounds, Upper bounds, Horner's schema, Computer assisted proof

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.29

Supplementary Material *Dataset*: `https://gitlab.com/KazAmano/integer-complexity`
archived at `swh:1:dir:e76eb9f674cdd675ba9a9aaf112f11934e7d23f6`

Funding Supported in part by JSPS Kakenhi No. JP21K19758, JP18K11152 and JP18H04090.

Acknowledgements The author would like to thank the anonymous reviewers of ISAAC '22 for their many helpful comments and suggestions.

1 Introduction

The *integer complexity* of a natural number n, denoted by $\|n\|$, is the smallest number of 1's needed to express n using an arbitrary combination of addition and multiplication (and parentheses). For instance, $\|6\| = 5$ since $6 = (1+1)\cdot(1+1+1)$ is a shortest expression (in terms of the number of 1's) of 6 that contains five 1's. Similarly, $\|11\| = 8$ by the witness

$$\begin{aligned} 11 &= (1+1+1)\cdot(1+1+1)+1+1 \ (=3\cdot 3+2) \\ &= (1+1)(1+1+1+1+1)+1 = (1+1)((1+1)(1+1)+1)+1 \ (=2\cdot 5+1). \end{aligned}$$

The problem is to find a behavior of $\|n\|$ as well as an optimal expression of n. This cute problem was originated by Mahler and Popken [13] in the 1950s and was later popularized by Guy [9, 10].

Since the last operation is either addition or multiplication, the value of $\|n\|$ is given by a simple recursion. Namely, $\|1\| = 1$ and, for $n > 1$,

$$\|n\| = \min_{\substack{a,b<n\in\mathbb{N} \\ ab=n \text{ or } a+b=n}} \{\|a\| + \|b\|\}. \tag{1}$$

In spite of its simplicity, the analysis of $\|n\|$ seems extremely hard.

It is not hard to see that

$$3\log_3 n \le \|n\| \le 3\log_2 n \sim 4.755\log_3 n. \tag{2}$$

See e.g., [9]. The lower bound is attained by $n = 3^k$ whose shortest expression is

$$3^k = (1+1+1)\cdot(1+1+1)\cdot\cdots\cdot(1+1+1) \quad (k \text{ times}).$$

The optimality of this expression can easily be verified by noticing that $\log_3 m/m$ takes its maximum at $m = 3$ when m is a positive integer. The upper bound in Eq. (2) is obtained by applying Horner's method to the binary representation of n [9]. Namely, for a $(k+1)$-bit binary number $n = \sum_{i=0}^{k} d_i 2^i$, we can express n by

$$n = d_0 + (1+1)(d_1 + (1+1)(d_2 + (\cdots(1+1)(d_{k-1}+1+1)\cdots))). \tag{3}$$

This shows that every $(k+1)$-bit binary number has integer complexity at most $2k$ plus one less than the number of 1's in the binary representation of n. This implies $\|n\| \le 2k + \sum_{i=0}^{k-1} d_i \le 3k$, which gives the upper bound in Eq. (2).

Despite a long line of research (e.g., [1, 2, 3, 4, 5, 6, 7, 11, 14, 18]), many fundamental problems remain open. For example, we cannot answer the question asked by Selfridge (see [9]) over three decades ago that

Is there a number a for which $\|2^a\| < 2a$?

For a notational simplicity, we write $\|n\|_{\log} := \|n\|/\log_3 n$ (following [11]). We know from Eq. (2) that $\|n\|_{\log} \in [3, 3\log_2 3]$ ($\sim [3, 4.755]$) for every n. However, the distribution of $\|n\|_{\log}$ is largely unknown. We do not know whether there are infinitely many n's such that $\|n\|_{\log} \ge (3+\epsilon)$ for some $\epsilon > 0$. Also, to the best of our knowledge, a general upper bound of the form $\|n\|_{\log} \le (3\log_2 3 - \epsilon)$ for some $\epsilon > 0$ has not been published yet.

Since the best known upper bound is obtained via the binary representation and the best known lower bound is obtained via the ternary representation, it would be interesting to investigate the integer complexity via the *mixed binary-ternary representation*. This is the main focus of this work.

The mixed binary-ternary representation is a number system in which we can arrange the base-two and base-three digits in an arbitrary order. Such a system has been considered before e.g., in [19] to pursue a new way of attack to the notorious Collatz problem.

When n is represented by this system using k digits where the i-th base is $b_i \in \{2, 3\}$ and the i-th digit is $d_i \in \{0, 1, \ldots, b_i - 1\}$ (here the least significant digit is counted as zero-th), then by applying Horner's schema, we have the expression

$$n = d_0 + b_0(d_1 + b_1(d_2 + (\cdots b_{k-3}(d_{k-2} + b_{k-2}(d_{k-1})\cdots))), \tag{4}$$

which contains $\sum_{i=0}^{k-2}(d_i + b_i)$ plus d_{k-1} 1's, where the last d_{k-1} can be discarded if $d_{k-1} = 1$.

Let $f(n)$ denote the minimum number of 1's needed to express n in this way. Obviously, $f(n) \ge \|n\|$ for every natural number n. In fact, a quick inspection shows that $f(n) = \|n\|$ for every $n \le 34$ and the first discrepancy occurs at $n = 35$ as $12 = f(35) \gneq \|35\| = 11$. See the end of Section 2 for some more experimental data on the comparison between $f(n)$ and $\|n\|$.

The value of $f(n)$ is given by the following recursive formula.[1]

$$f(n) := \begin{cases} \min\{f(\lfloor n/2 \rfloor) + 2 + (n \bmod 2),\ f(\lfloor n/3 \rfloor) + 3 + (n \bmod 3)\} & (n \geq 6), \\ n & (n \leq 5). \end{cases} \tag{5}$$

In what follows, we write $f(n)/\log_3 n$ as $f(n)_{\log}$. Apparently, $f(n)_{\log} \in [3, 4.755]$ and the lower limit is sharp since $f(3^k) = 3k$. At a first glance, we have a hope that it is not hard to evaluate $\limsup_{n\to\infty} f(n)_{\log}$ whose value is expected to be well below 4.755. However, we could not find a way to prove even a slightly better than a trivial bound such as $f(n)_{\log} < (3\log_2 3 - \epsilon)$ for some $\epsilon > 0$.

In this work, we extensively investigate on the behavior of $f(n)$ through the combination of computer experiments and theoretical analysis and obtain the following set of results.

1. Computer experiments support the hypothesis that $f(n)_{\log} < 3.483$ on average and $f(n)_{\log} < 4.212$ for all n.
2. For almost all natural numbers n, $3.120 < f(n)_{\log} < 3.587$.
3. There are infinitely many n's such that $f(n)_{\log} > 3.934$.

Note that the lower bound in #2 shows that the mixed binary-ternary representation is not strong enough for proving an upper bound of the form $\|n\|_{\log} \leq (3 + \epsilon)$.

In addition, we give the following "new records" on the original integer complexity by extending the computational efforts along the line suggested by the previous work [7, 15].

1. For a set of numbers of natural density one, $\|n\|_{\log} < 3.556$. This improves the bound of $\|n\|_{\log} < 3.620$ due to Cordwell et al. [5].
2. For a set of numbers of logarithmic density one, $\|n\|_{\log} < 3.472$. This improves the bound of $\|n\|_{\log} < 3.520$ due to David Bevan (described in [15]).

The rest of the paper is organized as follows. In Section 2, we formally introduce the mixed binary-ternary representation and discuss its connection to the integer complexity. In Section 3, we show the results of our computational experiments as well as some conjectures inspired by them. In Section 4, we show the theoretical results on the upper and lower bounds on $f(n)$. In Section 5, we describe an improvement to the upper bound on $\|n\|$. Finally, we make some discussions to close the paper in Section 6.

2 Mixed Binary-Ternary Representation

For natural numbers $n, m \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2 \ldots, n\}$ and $[m, n]$ denotes the set $\{m, m+1, \ldots, n\}$.

Throughout this work, we consider a number system with mixed bases of two and three. The *mixed binary-ternary representation* of a natural number is consisting of two arrays of integers Digit and Base. For $i \geq 0$, $\mathsf{Base}[i] \in \{2, 3\}$ represents the base of the i-th digit (here the least significant digit is counted as zero-th) and $\mathsf{Digit}[i] \in \{0, 1, \ldots, \mathsf{Base}[i] - 1\}$ represents the i-th digit.

Given Digit and Base of length k, a number n is represented as

$$n = \sum_{i=0}^{k-1} \mathsf{Digit}[i] \prod_{j=0}^{i-1} \mathsf{Base}[j]. \tag{6}$$

[1] In Eq. (5), we apply the first case only when $n \geq 6$. Actually, the values of $f(n)$ will not be changed if we replace $n \geq 6$ by $n \geq 2$. The reason that we did not do so is as follows. The second term in the minimum should give a complexity corresponding to a representation such that the lowest digit is base-three. However, e.g., when $n = 5$, it wrongly gives $f(1) + 3 + 2 = 6$ because it ignores the fact that we can omit the leading 1 to express n.

For notational simplicity, we may write this as $n = (d_{k-1})_{b_{k-1}} \cdots (d_1)_{b_1}(d_0)_{b_0}$ for $d_i = \mathsf{Digit}[i]$ and $b_i = \mathsf{Base}[i]$. Note that the representation is not unique. For example, $25 = 1_2 1_2 0_2 0_2 1_2 = 1_2 1_3 0_2 1_3 = 2_3 0_2 0_3 1_2 = 2_3 2_3 1_3$ (and more).

Given a natural number n in the mixed binary-ternary representation, we can express n using $\{1, +, \cdot\}$ (and parentheses) as

$$n = d_0 + b_0(d_1 + b_1(d_2 + (\cdots b_{k-3}(d_{k-2} + b_{k-2}(d_{k-1}) \cdots))),$$

where 2 is replaced by $(1 + 1)$ and 3 is replaced by $(1 + 1 + 1)$. This expression contains $\sum_{i=0}^{k-2}(d_i + b_i)$ plus d_{k-1} 1's, where the last d_{k-1} can be discarded if $d_{k-1} = 1$.

For a natural number $n \in \mathbb{N}$, let $f(n)$ denote the minimum number of 1's in such an expression, where the minimum is taken over all orderings of the base-two and base-three digits.

Obviously, $f(n) \geq \|n\|$ for every $n \in \mathbb{N}$.

As was described in Introduction, $f(n)$ is given by the following recursive formula:

$$f(n) := \begin{cases} \min\{f(\lfloor n/2 \rfloor) + 2 + (n \bmod 2),\ f(\lfloor n/3 \rfloor) + 3 + (n \bmod 3)\} & (n \geq 6), \\ n & (n \leq 5). \end{cases}$$

A quick computer calculation shows that $f(n) = \|n\|$ for every $n \leq 34$. The first discrepancy occurs at $n = 35$ as $12 = f(35) \gneq \|35\| = 11$. For $n \leq 100$, there are only four numbers $n \in \{35, 70, 71, 95\}$ satisfying $f(n) \gneq \|n\|$. As expected, our computer experiment suggests that the fraction of such numbers increases as n increases. For example, for $n \leq 10^6$, 652,037 numbers satisfy $f(n) \gneq \|n\|$.

3 Experimental Results

In this section, we describe our experimental results of $f(n)$ as well as some conjectures inspired by them. If we have enough memory to store all $f(n)$'s, then we can obtain the table of $f(n)$ in linear time by dynamic programming based on Eq. (5). If not, we can calculate $f(n)$ by expanding the recursion given in Eq. (5). This is time-consuming when n is large, but uses less memory. We can also combine these two methods in a hybrid manner.

In this work, we calculated $f(n)$ for $n \leq 2^{42}$ ($\sim 4.4 \cdot 10^{12}$) using a computer. All our experiments are conducted on an Ubuntu server with 64GB memory and Ryzen 3960x CPU (24 cores, 3.8GHz). The computation of $f(n)$ took about one week using a single core of the aforementioned machine.

3.1 Worst Case Bounds

For $k \geq 1$, let $e_f(k)$ denote the smallest integer n such that $f(n) \geq k$. Our computation is enough to determine the values of $e_f(k)$ for $k \leq 110$. Due to the space constraint, we only show the values $e_f(k)$ for $90 \leq k \leq 110$ in Table 1. See Appendix A for the full list.

It is interesting to see that $e_f(k) + 1$ (or $e_f(k) + 2$ for some cases) tends to be factored into the product of small primes, although there are some exceptions.

We also show a graph that plots $k/\log_3 e_f(k)$. See Fig. 1. The graph suggests that $k/\log_3 e_f(k)$ tends to around 4.2 as k tends to ∞. In the range $k \leq 110$, $k/\log_3 e_f(k)$ attains its maximum at $k = 81$ with $81/\log_3(1504935935) \sim 4.21103$. This suggests the following conjecture, which would give a considerably better bound than the known bound of $\|n\|_{\log} < 4.755$.

▶ **Conjecture 1.** *For every natural number n, $\|n\|_{\log} \leq f(n)_{\log} < 4.212$.*

Table 1 The value of $e_f(k)$ for $90 \leq k \leq 110$.

k	$e_f(k)$	factorization of $e_f(k)+1$
90	18,059,231,231	$2^{17} \cdot 3^9 \cdot 7$
91	27,088,846,847	$2^{16} \cdot 3^{10} \cdot 7$
92	36,118,462,463	$2^{18} \cdot 3^9 \cdot 7$
93	54,177,693,691	$2^2 \cdot 683 \cdot 19830781$
94	54,177,693,695	$2^{17} \cdot 3^{10} \cdot 7$
95	72,236,924,927	$2^{19} \cdot 3^9 \cdot 7$
96	108,355,387,390	prime $(= 2^{18} \cdot 3^{10} \cdot 7 - 1)$
97	108,355,387,391	$2^{18} \cdot 3^{10} \cdot 7$
98	162,533,081,087	$2^{17} \cdot 3^{11} \cdot 7$
99	216,710,774,782	prime $(= 2^{19} \cdot 3^{10} \cdot 7 - 1)$
100	216,710,774,783	$2^{19} \cdot 3^{10} \cdot 7$
101	325,066,162,175	$2^{18} \cdot 3^{11} \cdot 7$
102	487,599,243,263	$2^{17} \cdot 3^{12} \cdot 7$
103	650,132,324,347	$2^2 \cdot 13 \cdot 29 \cdot 15443 \cdot 27917$
104	650,132,324,351	$2^{19} \cdot 3^{11} \cdot 7$
105	975,198,486,527	$2^{18} \cdot 3^{12} \cdot 7$
106	1,486,016,741,374	$5^3 \cdot 163 \cdot 72933337$ $(= 2^{23} \cdot 3^{11} - 1)$
107	1,486,016,741,375	$2^{23} \cdot 3^{11}$
108	1,950,396,973,055	$2^{19} \cdot 3^{12} \cdot 7^1$
109	3,715,041,853,439	$2^{22} \cdot 3^{11} \cdot 5$
110	3,900,793,946,111	$2^{20} \cdot 3^{12} \cdot 7$

It should be noted that Iraids et al. [11] computed the integer complexity of n for $n \leq 10^{12}$. Let $e(k)$ denote the smallest integer n such that $\|n\| \geq k$, i.e., a counterpart of $e_f(k)$. They determined $e(k)$ up to $k \leq 89$ (see the sequence A005520 in OEIS). By inspecting the results, they conjectured that $\lim_{k\to\infty} \frac{k}{\log_3 e(k)} \sim 3.37$. This suggests that there is a substantial gap between two measures $f(n)$ and $\|n\|$ in the worst case.

3.2 Average Case

Recall that we write $f(n)_{\log} := f(n)/\log_3 n$. For $k \geq 1$, let E_k denote the average of $f(n)_{\log}$ over all k-bit integers, i.e., $E_k := \mathbf{E}_{n\in[2^{k-1},2^k-1]}[f(n)_{\log}]$. Our computation reveals the exact value of E_k's for $k \leq 42$. See Table 2. Interestingly, the value of E_k hits the peak at $k = 36$ with $E_{36} \sim 3.48829$ and then starts to decrease.

In order to clarify this phenomenon, we tried to estimate the value of E_k's for larger values of k by random sampling. Namely, for each $43 \leq k \leq 100$, we compute $f(n)$ for randomly chosen 2^{25} k-bit integers to estimate E_k. Note that the computation of $f(n)$ itself is exact. See Fig. 2. The value of E_k seems to be steadily decreasing as k is increasing. The value of E_{100} is around 3.483.

We believe that $\lim_{k\to\infty} E_k$ exists. However, at this point, we do not have a reasonable guess on the value of this limit. In Section 4.3, we will show a non-trivial lower bound of $\lim_{k\to\infty} E_k > 3.120$ (if the limit exists).

▶ **Problem 2.** *Does* $\lim_{k\to\infty} E_k$ *exist? If it exists, find the value.*

For a comparison to the original integer complexity $\|n\|$, let $\tilde{E}_k$ denote a counterpart of E_k. A quick computation (based on Eq. (1)) suggest that $\tilde{E}_k$ seems to be decreasing as k is increasing. For example, $\tilde{E}_{20} \sim 3.3859$, $\tilde{E}_{25} \sim 3.3744$ and $\tilde{E}_{30} \sim 3.3644$ (see also [14] for an

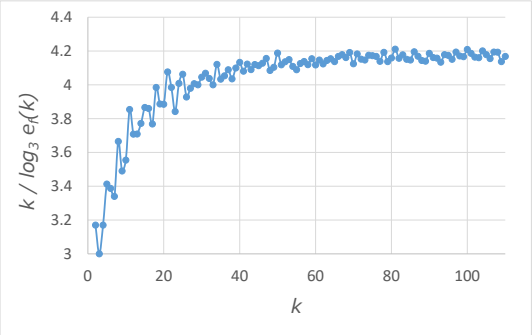


Figure 1 The plot of $k/\log_3 e_f(k)$.

Table 2 The average E_k of $f(n)_{\log}$ over all k-bit integers. The seventh digit from the decimal point is rounded.

k	E_k	k	E_k
20	3.482808	32	3.488126
21	3.483896	33	3.488197
22	3.484733	34	3.488248
23	3.485483	35	3.488275
24	3.486055	36	3.488290
25	3.486533	37	3.488284
26	3.486954	38	3.488270
27	3.487253	39	3.488243
28	3.487529	40	3.488208
29	3.487720	41	3.488164
30	3.487891	42	3.488114
31	3.488024		

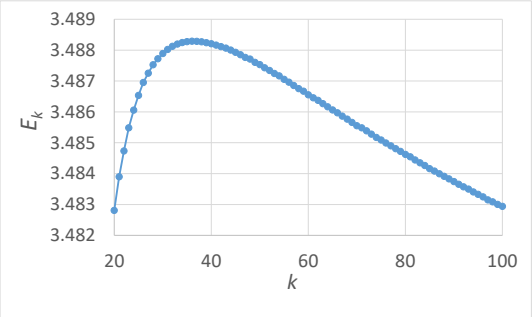


Figure 2 The plot of E_k. The value of E_k for $k \geq 43$ is an estimation by 2^{25} randomly chosen k-bit integers.

earlier experiment). We do not know whether $\lim_{k\to\infty} E_k$ is strictly larger than $\lim_{k\to\infty} \tilde{E}_k$, although the equality is unlikely to hold. Note that no lower bound on $\lim_{k\to\infty} \tilde{E}_k$ better than three is known.

4 Theoretical Results

In this section, we show the items #2 and #3 listed in Introduction. Note that some of the proofs are computer assisted.

4.1 Average Case Upper Bound

The first theoretical result is an upper bound on the average of $f(n)$. Note that the constant ~ 3.587 in the statement of the theorem is a bit worse than the observed average of ~ 3.483 described in the previous section.

▶ **Theorem 3.** *For almost all natural numbers n, $f(n)_{\log} \leq 3.587$.*

Here and hereafter, we say that a statement holds for almost all natural numbers n if it holds on a subset of $\mathbb{N}$ of natural density one.

Proof. The proof relies on the argument developed in [7] together with additional computational efforts.

Let $a, b \geq 0$ be two integers (whose values will be chosen later) and put $c = a + b$. Let $S_{a,b}$ be the set of all c-tuples consisting of a 2's and b 3's. Note that $|S_{a,b}| = \binom{c}{a}$. Given an integer $m \in [0, 2^a 3^b - 1]$ and $t = (t_0, t_1, \ldots, t_{c-1}) \in S_{a,b}$, let $d_t(m)$ denote the sum of the digits in the mixed binary-ternary representation in which the base of the i-th digit is t_i for $i = 0, 1, \ldots, c-1$. That is, $d_t(m) = \sum_{i=0}^{c-1} d_i$ where $m = (d_{c-1})_{t_{c-1}} \cdots (d_1)_{t_1} (d_0)_{t_0}$.

Let $E_{a,b}$ be the expectation of the minimum of $d_t(m)$ where m is chosen uniformly from $[0, 2^a 3^b - 1]$ and the minimum is taken over all $t \in S_{a,b}$, i.e.,

$$E_{a,b} = \frac{1}{2^a 3^b} \sum_{m \in [0, 2^a 3^b - 1]} \min_{t \in S_{a,b}} d_t(m).$$

We first claim that, for almost all natural numbers n, it holds that

$$f(n)_{\log} = \frac{f(n)}{\log_3 n} \leq \frac{2a + 3b + E_{a,b}}{\log_3(2^a 3^b)} + \epsilon,$$

for an arbitrarily small $\epsilon > 0$. Note that this claim was essentially shown in [7]. We include the proof here for the completeness.

The proof of the claim is as follows. Let k be a sufficiently large integer and let $n = r_k \cdots r_1 r_0$ in base $2^a 3^b$ such that $r_k \neq 0$. Notice that $n \geq (2^a 3^b)^k$.

By considering the expression based on Horner's expansion (Eq. (4)) to base $2^a 3^b$ representation of n, we see that

$$\frac{f(n)}{\log_3 n} \leq \frac{f(r_k)}{k \log_3(2^a 3^b)} + \frac{k(2a + 3b) + \sum_{i=0}^{k-1} \min_{t \in S_{a,b}} d_t(r_i)}{k \log_3(2^a 3^b)}. \tag{7}$$

The first term in Eq. (7) tends to 0 as k tends to ∞. By applying Chernoff bounds, we see that for almost all natural numbers n, the difference

$$\frac{1}{k} \sum_{i=0}^{k-1} \min_{t \in S_{a,b}} d_t(r_i) - \frac{1}{2^a 3^b} \sum_{m \in [0, 2^a 3^b - 1]} \min_{t \in S_{a,b}} d_t(m)$$

is arbitrarily small. By noticing that the second term is equal to $E_{a,b}$, we complete the proof of the claim.

Given a and b, we can calculate $E_{a,b}$ using a computer if a and b are not so large. We calculated $E_{a,b}$ for a and b such that $a+b \le 31$. The best bound is obtained when $a=17$ and $b=14$. We have $E_{17,14} = 7941127132545/(2^{17}\cdot 3^{14})$, which yields that

$$f(n)_{\log n} \le \frac{76+E_{17,14}}{\log_3(2^{17}\cdot 3^{14})} \sim 3.58601,$$

for almost all natural numbers n. This completes the proof of Theorem 3. ◀

The argument in the proof of Theorem 3 can be used for obtaining an upper bound on $\|n\|_{\log}$ using higher bases. We conducted the computation for several combinations of the exponents to the bases $\{2,3,5,7\}$. The best bound on $\|n\|_{\log}$ that we have obtained so far is when we consider the base $2^{15}3^{14}5^1$. We state this as a separate theorem.

▶ **Theorem 4.** *For almost all natural numbers n, $\|n\|_{\log} \le 3.556$.*

Proof. Let $a,b,c \ge 0$ be integers and define $E_{a,b,c}$ analogously to $E_{a,b}$, but the base is $2^a3^b5^c$ rather than 2^a3^b. By a similar reasoning to the proof of Theorem 3, we have

$$\|n\|_{\log n} \le \frac{2a+3b+5c+E_{a,b,c}}{\log_3(2^a3^b5^c)} + \epsilon,$$

for an arbitrary small $\epsilon > 0$.

A computer calculation shows $E_{15,14,1} = 9111121836916/(2^{15}\cdot 3^{14}\cdot 5)$. This implies that

$$\|n\|_{\log} \le \frac{77+E_{15,14,1}}{\log_3(2^{15}\cdot 3^{14}\cdot 5)} \sim 3.55517,$$

for almost all natural numbers n. ◀

Theorem 4 improves the best-known bound of $\|n\|_{\log} \le 3.6199$ due to Cordwell et al. [5], which was obtained by calculating $E_{11,9}$ in our terminology. (See also Arias de Reyna and van de Lune [7, Proposition 12] for an earlier bound of $\|n\|_{\log} < 3.635$ based on the calculation of $E_{9,8}$). In this work, we could calculate $E_{a,b}$ for much larger values of a and b than theirs. A key insight is that, for example, a 30-digit mixed binary-ternary representation in base $2^{15}3^{15}$ can be viewed as a concatenation of two 15-digit representations in base $2^{c_1}3^{15-c_1}$ and $2^{15-c_1}3^{c_1}$ for some $0 \le c_1 \le 15$. Then, a simple dynamic programming speeds up the calculation.

4.2 Average Case Lower Bound

The second theoretical result is a lower bound on the average of $f(n)$. The following theorem says that the expression based on the mixed binary-ternary representation is not rich enough to give an upper bound of $\|n\|_{\log} \le (3+\epsilon)$ for a small ϵ.

▶ **Theorem 5.** *For almost all natural numbers n, $f(n)_{\log} \ge 3.1201$.*

Proof. The proof is by counting argument. Suppose that k is a sufficiently large integer.

Let α be a real number such that $0 \le \alpha \le 1$ whose value will be chosen later. Consider a mixed binary-ternary representation of a k-bit integer in which αk digits are base two and $(1-\alpha)k\log_3 2$ digits are base three. Strictly speaking, these two numbers should be integers. This can be fulfilled by choosing α to be a multiple of $1/k$, and then set the number of base-three digits to $\lceil (1-\alpha)k\log_3 2\rceil$. However, since the effect is negligible, we omit the ceiling functions for clarity.

We will count the number of k-bit integers such that it has a representation satisfying the condition that the sum of the digits is at most βk. The value of β will be chosen appropriately in the later part of the proof.

Given α, the number of orderings of the base-two and base-three digits is given by

$$\binom{(\alpha + (1-\alpha)\log_3 2)\, k}{\alpha k}. \tag{8}$$

Once the bases are fixed to $(b_0, \ldots, b_{(\alpha+(1-\alpha)\log_3 2)k-1})$, the number of integers such that the sum of the digits is βk is at most

$$\binom{(\alpha + 2(1-\alpha)\log_3 2)\, k}{\beta k}. \tag{9}$$

This is because such an integer is the sum of βk numbers chosen from the multiset of size $(\alpha + 2(1-\alpha)\log_3 2)k$ given by

$$\left\{\prod_{j=0}^{i-1} b_j\right\}_i \cup \left\{\prod_{j=0}^{i-1} b_j\right\}_{i:b_i=3},$$

where the element $\prod_{j=0}^{i-1} b_j$ has multiplicity two in the set when $b_i = 3$.

We will use the following standard bound on the binomial coefficients:

$$\binom{n}{k} \leq 2^{nH(n/k)},$$

where $H(\cdot)$ denotes the binary entropy function.

By multiplying Eqs. (8) and (9), we have that the number of k-bit integers that has a mixed binary-ternary representation with αk base-two digits such that the sum of the digits is at most βk is upper bounded by $poly(k) \cdot 2^{\gamma k}$, where

$$\begin{aligned}\gamma \;=\; & (\alpha + (1-\alpha)\log_3 2) H\left(\frac{\alpha}{\alpha + (1-\alpha)\log_3 2}\right) \\ & + (\alpha + 2(1-\alpha)\log_3 2) H\left(\frac{\beta}{\alpha + 2(1-\alpha)\log_3 2}\right).\end{aligned}$$

For $0 \leq \alpha \leq 1$, let β_α denote the maximum value of β in the above expression such that $\gamma \leq 0.99999$ (say, γ is strictly smaller than 1).

Notice that, given α, the number of 1's in the expression of n obtained by applying Horner's schema to its mixed binary-ternary representation is given by

$$k(2\alpha + 3(1-\alpha)\log_3 2 + \beta) - O(1),$$

where βk is the sum of the digits.

Note that since $\alpha = \alpha'/k$ for some $\alpha \in [0, k]$, the number of possible α's is $O(k)$. Thus, for every sufficiently large k and for almost all k-bit binary integers n, we have

$$f(n)_{\log} \geq \frac{\min_{0\leq\alpha\leq 1}\{2\alpha + 3(1-\alpha)\log_3 2 + \beta_\alpha\}}{\log_3 2} - \epsilon$$

for an arbitrary small $\epsilon > 0$. Here we use the fact that $n < 2^k$.

By a simple numerical calculation, we can see that the right-hand side of the above is ~ 3.12017. The minimum is attained at $\alpha \sim 0.3536$ with $\beta_\alpha \sim 0.0379$. This completes the proof of Theorem 5. ◀

▶ Remark 6. The bound in Theorem 5 would slightly be improved if we refine the estimation in Eq. (9) by using trinomial coefficients instead of binary coefficients (by taking into account that the number $\prod_{j=0}^{i-1} b_j$ (for i with $b_i = 3$) appears twice in the ground set). However, the improvement seems to be tiny and hence we did not include a detailed calculation in this version.

4.3 Worst Case Lower Bound

The last result in this section is a lower bound on the worst case of $f(n)$.

▶ **Theorem 7.** *There are infinitely many numbers n such that $f(n)_{\log} \geq 3.934$.*

Proof. The proof is constructive. The key fact is that if $n \equiv -1 \pmod{2^a 3^b}$, then all of a lowest base-two digits as well as b lowest base-three digits have its maximal value, i.e., one or two depending on the base of the digit.

Let $n := \alpha 2^a 3^b - 1$ for a relatively small integer $\alpha > 0$. The values of a and b will be chosen later. For every ordering of the bases in a mixed binary-ternary representation of n, Horner's expression (shown in Eq. (4)) incurs cost three for each of the first a occurrences of the base-two digit, and similarly, incurs cost five for each of the first b occurrences of the base-three digit.

By recalling a general lower bound of $||n|| \geq 3 \log_3 n$, we can see that $f(n)$ is lower bounded by the minimum of $3a + 3\log(n/2^a)$ and $5b + 3\log_3(n/3^b)$.

Let $k := b/a$. If a and b are sufficiently large with respect to α, then

$$\begin{aligned} f(n)_{\log} = \frac{f(n)}{\log_3 n} &\geq \frac{\min\{3a + 3ak, 5ak + 3a\log_3 2\}}{a\log_3 2 + ak} - \epsilon \\ &= \frac{\min\{3 + 3k, 5k + 3\log_3 2\}}{\log_3 2 + k} - \epsilon \end{aligned}$$

for an arbitrary small constant $\epsilon > 0$.

A simple calculation shows that the last term attains a maximum of $(15 - 9\log_3 2)/(3 - \log_3 2) \sim 3.9347$ at $k = \frac{3}{2}(1 - \log_3 2) \sim 0.5536$. This completes the proof of the theorem. ◀

▶ Remark 8. In the proof of Theorem 7, we use a (pessimistic) lower bound of $\|n\|_{\log} \geq 3$ to estimate the length of an upper part of an expression. If we use an optimistic (but not proven) constant ~ 3.48 instead of 3, which is suggested by the experiments in Section 3.2, then a simple calculation shows that the bound in the theorem would become ~ 4.17. This seems reasonably consistent with the experiments in Section 3.1 (See Fig. 1). In addition, our proof suggests that the worst case occurs when $n = \alpha 2^a 3^b - 1$ for $b \sim 0.55a$ and small α. This is also consistent with the factorization data shown in Table 1.

5 Beyond Ternary

If our aim is to obtain an upper bound on $\|n\|_{\log}$ rather than $f(n)_{\log}$, there are no reasons to restrict the bases to two or three. We can consider a number system including higher bases.

In [15] (refining the work in [16]), Shriver formalized a method for obtaining an upper bound on $\|n\|_{\log}$ that is satisfied by a set of numbers $n \in \mathbb{N}$ of *logarithmic density one*. The logarithmic density of a set $S \subseteq \mathbb{N}$ of integers is defined by

$$\lim_{N\to\infty} \frac{1}{\log N} \sum_{n\in S, n\leq N} \frac{1}{n}$$

if the limit exists.

Below we describe an outline of their method using a greedy algorithm in base six as an illustrative example. See [15] for more details.

Consider the following algorithm that outputs an expression of a given number n.

1. If $n \leq 5$, output an optimal representation of n. Otherwise, move to step 2.
2. Choose a representation depending on $n \pmod 6$:
   - If $n \pmod 6 \equiv 0$, write as $3(n/3)$,
   - If $n \pmod 6 \equiv 1$, write as $3((n-1)/3)+1$,
   - If $n \pmod 6 \equiv 2$, write as $2(n/2)$,
   - If $n \pmod 6 \equiv 3$, write as $3(n/3)$,
   - If $n \pmod 6 \equiv 4$, write as $2(n/2)$,
   - If $n \pmod 6 \equiv 5$, write as $2((n-1)/2)+1$.

   Then apply step 2 to the result (the number in brackets) until one of the representations in step 1 can be applied.
3. Replace every 2 by (1+1) and every 3 by (1+1+1).

Following [15], we write this algorithm by $D := (3,3,2,3,2,2)$ which represents the divisors of each residue class in step 2.

Divide $\mathbb{N}$ into six residue classes $\{0,1,\ldots,5\}$. For example, in step 2, $6m+1$ is going to $2m$, that we rewrite as $6m'$, $6m'+2$ or $6m'+4$. We interpret this as that class 1 is going to class 0, 2, or 4 with equal probability. Then the transition of numbers in each class can be expressed by a six-state Markov model whose transition matrix is

$$M = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}.$$

Let π be the stationary distribution of M. An easy calculation shows that

$$\pi^T = \left(\frac{1}{13}, \frac{2}{13}, \frac{4}{13}, 0, \frac{3}{13}, \frac{3}{13}\right).$$

Let C be a 6-tuple whose i-th element represents the number of 1's used in a single step for the residue class i in step 2 of the algorithm. Namely, $C := (3,4,2,3,2,3)$.

Shriver [15] showed that, for an arbitrary small constant $\epsilon > 0$, the algorithm outputs an expression including at most $(\alpha+\epsilon)\log_3 n$ 1's for a set of numbers n with logarithmic density one, where

$$\alpha := \frac{\sum_i \pi(i) C[i]}{\sum_i \pi(i) \log_3 D[i]}.$$

An easy calculation shows that $\alpha \sim 3.6522$ for the above algorithm.

As expected, we can improve a bound by seeking an algorithm for a larger base or/and by optimizing the divisors of each residue class. The best-known bound obtained along this line is by Bevan (described in [15]) that $\|n\|_{\log} < 3.5197$ based on an algorithm in base 2310 $(=2\cdot 3\cdot 5\cdot 7\cdot 11)$.

In this work, we further extend their computational efforts. We found an algorithm in base $2^3\cdot 3^2\cdot 5\cdot 7\cdot 11 = 27,720$, which gives the following improved bound.

▶ **Theorem 9.** *The set of natural numbers n such that $\|n\|_{\log} \leq 3.4713$ has the logarithmic density one.*

The witness of Theorem 9 is the description of a $27,720$-tuple D, which is available at `https://gitlab.com/KazAmano/integer-complexity`[2]. Note that the problem for finding a good algorithm (or, equivalently, finding the divisors for each residue class in our case) is a non-convex optimization problem. We apply the gradient descent method in our computation. In each step, we need to calculate the stationary distribution of a 27,720-state Markov chain many times. This step is efficiently be implemented by using a power method.

6 Discussions

There remain many interesting problems to be explored. One of such is a question whether $f(2^k) = 2k$ for every $k \geq 1$, which has a similar spirit to the question whether $\|2^k\| = 2k$ asked by Selfridge (see [9]). This question has a close connection to a fundamental problem in number theory, i.e., the distribution of digits in the ternary representation of 2^n (discussed in e.g., [8, 12, 17]). Currently, no counterexamples are known. During this work, we verified that $f(2^k) = 2k$ for every $k \leq 200$ by computer calculations.

In [11], Iraids et al. have made an interesting discussion on the distribution of $\|n\|_{\log}$. It is known that $\|n\|_{\log} \in [3, 4755]$. Based on their computational results, they see that the interval $[3, 4755]$ shall be divided into three epochs depending on the density of $\|n\|_{\log}$. From 3 to some constant, say C_1, the values of $\|n\|_{\log}$ is "sparse". Then, from C_1 to some constant, say C_2, the set of values of $\|n\|_{\log}$ is "dense". Finally, from C_2 (to 4.755), the $\|n\|_{\log}$ is "absolutely sparse" meaning that there are only finitely many numbers n such that $\|n\|_{\log} > C_2$. By inspecting their experimental results, they conjectured that

$$3.1699 \sim \|2\|_{\log} \leq C_1 \leq C_2 \leq 3.37.$$

Based on the results of this work, we find that a similar analysis can be made for our complexity measure $f(n)_{\log}$. However, this time, there may be four epochs, namely, "sparse", "dense", "sparse", and "absolutely sparse" from 3 to the largest value of $f(n)_{\log}$. Here the "sparse" means that it contains infinitely many numbers but its density is zero.

Theorems 3 and 5 imply that the "dense" epoch is contained in the interval $[3.120, 3.587]$. Narrowing this interval is interesting future work. Also, Theorem 7 says that the point separating the third and fourth epochs is at least 3.934 (if it exists). Refining this bound as well as obtaining a general upper bound on $f(n)_{\log}$ are also interesting future work.

References

1 Harry J. Altman. Integer complexity: Representing numbers of bounded defect. *Theor. Comput. Sci.*, 652:64–85, 2016. `doi:10.1016/j.tcs.2016.09.005`.

2 Harry J. Altman. Integer complexity: Algorithms and computational results. *Integers*, 18:A45, 2018. URL: `http://math.colgate.edu/%7Eintegers/s45/s45.Abstract.html`.

3 Harry J. Altman and Joshua Zelinsky. Numbers with integer complexity close to the lower bound. *Integers*, 12A:A1, 2012. URL: `http://math.colgate.edu/%7Eintegers/a1self/a1self.Abstract.html`.

4 Juris Cernenoks, Janis Iraids, Martins Opmanis, Rihards Opmanis, and Karlis Podnieks. Integer complexity: Experimental and analytical results II. In Jeffrey O. Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 58–69. Springer, 2015. `doi:10.1007/978-3-319-19225-3_5`.

[2] Note that an additional condition that, for an underlying graph of M (which is uniquely determined by D), every vertex has a path to the vertex labeled 1, should be satisfied ([15, Proposition 4.5]). We checked that our D satisfies this condition computationally.

5 Katherine Cordwell, Alyssa Epstein, Anand Hemmandy, Steven J. Miller, Eyvindur A. Palsson, Aaditya Sharma, Stefan Steinerberger, and Yen Nhi Truong Vu. On algorithms to calculate integer complexity. *Integers*, 19:A12, 2019. URL: `http://math.colgate.edu/%7Eintegers/t12/t12.Abstract.html`.

6 J. Arias de Reyna. Complexity of natural numbers, 2021. (translation of Complejidad de los números naturales, Gac. R. Soc. Mat. Esp. 3, 230–250, 2000). `arXiv:2111.03345`.

7 J. Arias de Reyna and J. van de Lune. Algorithms for determining integer complexity, 2014. `arXiv:1404.2183`.

8 Taylor Dupuy and David Weirich. Bits of 3^n in binary, wieferich primes and a conjecture of Erdös. *J. Number Theory*, 158:268–280, 2016. `doi:10.1016/j.jnt.2015.05.022`.

9 Ricard K. Guy. What is the least number of ones needed to represent n using only $+$ and $\times$ (and parentheses)? *Amer. Math. Monthly*, 93:189–190, 1986.

10 Ricard K. Guy. *Unsolved Problems in Number Theory, Third Edition, Problem F26: Expressing numbers with just one*. Springer-Verlag, New York, 2004.

11 Jānis Iraids, Kaspars Balodis, Juris Čerņenoks, Mārtiņš Opmanis, Rihards Opmanis, and Kārlis Podnieks. Integer complexity: Experimental and analytical results. *Scientific Papers University of Latvia, Computer Science and Information Technologies*, 787:153–179, 2012. (also at `arxiv:1203.6462`).

12 Jeffrey C. Lagarias. Ternary expansions of powers of 2. *J. London Mathematical Society*, 79(3):562–588, 2009. `doi:doi.org/10.1112/jlms/jdn080`.

13 K. Mahler and J. Popken. On a maximum problem in arithmetic (dutch). *Nieuw Arch. Wiskunde*, 3(1):1–15, 1953.

14 Daniel A. Rawsthorne. How many 1's are needed? *Fib. Quart.*, 27:14–17, 1989.

15 Christopher E. Shriver. An application of markov chain analysis to integer complexity, 2015. `arXiv:1511.07842`.

16 Stefan Steinerberger. A short note on integer complexity. *Contributions to Discrete Mathematics*, 9:63–69, 2014.

17 Cameron L. Stewart. On the representation of an integer in two different bases. *Journal fur die reine und angewandte Mathematik*, 319:63–72, 1980.

18 Venecia Wang. A counterexample to the prime conjecture of expressing numbers using just ones. *J. Number Theory*, 133(2), 2013. `doi:10.1016/j.jnt.2012.08.003`.

19 Emre Yolcu, Scott Aaronson, and Marijn J. H. Heule. An automated approach to the collatz conjecture. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 468–484. Springer, 2021. `doi:10.1007/978-3-030-79876-5_27`.

A Appendix A

In the following, we show the full list of $e_f(k)$ for $k \leq 110$ in Tables 3, 4 and 5.

Table 3 The value of $e_f(k)$ for $k \leq 40$.

k	$e_f(k)$	factorization of $e_f(k)+1$
1	1	2
2	2	3
3	3	2^2
4	4	5
5	5	$2 \cdot 3$
6	7	2^3
7	10	11
8	11	$2^2 \cdot 3$
9	17	$2 \cdot 3^2$
10	22	23
11	23	$2^3 \cdot 3$
12	35	$2^2 \cdot 3^2$
13	47	$2^4 \cdot 3$
14	59	$2^2 \cdot 3 \cdot 5$
15	71	$2^3 \cdot 3^2$
16	95	$2^5 \cdot 3$
17	142	$11 \cdot 13$ $(= 2^4 \cdot 3^2 - 1)$
18	143	$2^4 \cdot 3^2$
19	215	$2^3 \cdot 3^3$
20	286	$7 \cdot 41$ $(= 2^5 \cdot 3^2 - 1)$
21	287	$2^5 \cdot 3^2$
22	431	$2^4 \cdot 3^3$
23	718	prime $(= 2^4 \cdot 3^2 \cdot 5 - 1)$
24	719	$2^4 \cdot 3^2 \cdot 5$
25	863	$7 \cdot 103$ $(= 2^5 \cdot 3^3)$
26	1,439	$2^5 \cdot 3^2 \cdot 5$
27	1,727	$2^6 \cdot 3^3$
28	2,159	$2^4 \cdot 3^3 \cdot 5$
29	2,879	$2^6 \cdot 3^2 \cdot 5$
30	3,455	$2^7 \cdot 3^3$
31	4,319	$2^5 \cdot 3^3 \cdot 5$
32	6,047	$2^5 \cdot 3^3 \cdot 7$
33	8,638	$53 \cdot 163$ $(= 2^6 \cdot 3^3 \cdot 5 - 1)$
34	8,639	$2^6 \cdot 3^3 \cdot 5$
35	13,823	$2^9 \cdot 3^3$
36	17,279	$2^7 \cdot 3^3 \cdot 5$
37	20,735	$2^8 \cdot 3^4$
38	31,103	$2^7 \cdot 3^5$
39	34,559	$2^8 \cdot 3^3 \cdot 5$
40	41,471	$2^9 \cdot 3^4$

Table 4 The value of $e_f(k)$ for $41 \leq k \leq 80$.

k	$e_f(k)$	factorization of $e_f(k)+1$
41	62,207	$2^8 \cdot 3^5$
42	72,575	$2^7 \cdot 3^4 \cdot 7$
43	103,679	$2^8 \cdot 3^4 \cdot 5$
44	124,415	$2^9 \cdot 3^5$
45	165,887	$2^{11} \cdot 3^4$
46	207,359	$2^9 \cdot 3^4 \cdot 5$
47	248,831	$2^{10} \cdot 3^5$
48	404,351	$2^7 \cdot 3^5 \cdot 13$
49	497,662	prime $(= 2^{11} \cdot 3^5 - 1)$
50	497,663	$2^{11} \cdot 3^5$
51	808,703	$2^8 \cdot 3^5 \cdot 13$
52	995,327	$2^{12} \cdot 3^5$
53	1,244,159	$2^{10} \cdot 3^5 \cdot 5$
54	1,866,239	$2^9 \cdot 3^6 \cdot 5$
55	2,612,735	$2^9 \cdot 3^6 \cdot 7$
56	2,985,983	$2^{12} \cdot 3^6$
57	3,732,479	$2^{10} \cdot 3^6 \cdot 5$
58	5,225,471	$2^{10} \cdot 3^6 \cdot 7$
59	5,971,967	$2^{13} \cdot 3^6$
60	8,957,951	$2^{12} \cdot 3^7$
61	10,450,943	$2^{11} \cdot 3^6 \cdot 7$
62	14,929,919	$2^{12} \cdot 3^6 \cdot 5$
63	17,915,903	$2^{13} \cdot 3^7$
64	22,394,879	$2^{11} \cdot 3^7 \cdot 5$
65	31,352,831	$2^{11} \cdot 3^7 \cdot 7$
66	35,831,807	$2^{14} \cdot 3^7$
67	44,789,759	$2^{12} \cdot 3^7 \cdot 5$
68	62,705,663	$2^{12} \cdot 3^7 \cdot 7$
69	71,663,615	$2^{15} \cdot 3^7$
70	125,411,326	prime $(= 2^{13} \cdot 3^7 \cdot 7 - 1)$
71	125,411,327	$2^{13} \cdot 3^7 \cdot 7$
72	188,116,991	$2^{12} \cdot 3^8 \cdot 7$
73	250,822,655	$2^{14} \cdot 3^7 \cdot 7$
74	286,654,463	$2^{17} \cdot 3^7$
75	376,233,983	$2^{13} \cdot 3^8 \cdot 7$
76	501,645,311	$2^{15} \cdot 3^7 \cdot 7$
77	752,467,966	$3257 \cdot 231031$ $(= 2^{14} \cdot 3^8 \cdot 7 - 1)$
78	752,467,967	$2^{14} \cdot 3^8 \cdot 7$
79	1,289,945,087	$2^{16} \cdot 3^9$
80	1,504,935,934	$5 \cdot 300987187$ $(= 2^{15} \cdot 3^8 \cdot 7 - 1)$

Table 5 The value of $e_f(k)$ for $81 \leq k \leq 110$.

k	$e_f(k)$	factorization of $e_f(k)+1$
81	$1,504,935,935$	$2^{15} \cdot 3^8 \cdot 7$
82	$2,579,890,175$	$2^{17} \cdot 3^9$
83	$3,009,871,871$	$2^{16} \cdot 3^8 \cdot 7$
84	$4,514,807,807$	$2^{15} \cdot 3^9 \cdot 7$
85	$6,019,743,742$	$13 \cdot 463057211$ $(= 2^{17} \cdot 3^8 \cdot 7 - 1)$
86	$6,019,743,743$	$2^{17} \cdot 3^8 \cdot 7$
87	$9,029,615,615$	$2^{16} \cdot 3^9 \cdot 7$
88	$13,544,423,423$	$2^{15} \cdot 3^{10} \cdot 7$
89	$18,059,231,227$	$2^2 \cdot 4514807807$
90	$18,059,231,231$	$2^{17} \cdot 3^9 \cdot 7$
91	$27,088,846,847$	$2^{16} \cdot 3^{10} \cdot 7$
92	$36,118,462,463$	$2^{18} \cdot 3^9 \cdot 7$
93	$54,177,693,691$	$2^2 \cdot 683 \cdot 19830781$
94	$54,177,693,695$	$2^{17} \cdot 3^{10} \cdot 7$
95	$72,236,924,927$	$2^{19} \cdot 3^9 \cdot 7$
96	$108,355,387,390$	prime $(= 2^{18} \cdot 3^{10} \cdot 7 - 1)$
97	$108,355,387,391$	$2^{18} \cdot 3^{10} \cdot 7$
98	$162,533,081,087$	$2^{17} \cdot 3^{11} \cdot 7$
99	$216,710,774,782$	prime $(= 2^{19} \cdot 3^{10} \cdot 7 - 1)$
100	$216,710,774,783$	$2^{19} \cdot 3^{10} \cdot 7$
101	$325,066,162,175$	$2^{18} \cdot 3^{11} \cdot 7$
102	$487,599,243,263$	$2^{17} \cdot 3^{12} \cdot 7$
103	$650,132,324,347$	$2^2 \cdot 13 \cdot 29 \cdot 15443 \cdot 27917$
104	$650,132,324,351$	$2^{19} \cdot 3^{11} \cdot 7$
105	$975,198,486,527$	$2^{18} \cdot 3^{12} \cdot 7$
106	$1,486,016,741,374$	$5^3 \cdot 163 \cdot 72933337$ $(= 2^{23} \cdot 3^{11} - 1)$
107	$1,486,016,741,375$	$2^{23} \cdot 3^{11}$
108	$1,950,396,973,055$	$2^{19} \cdot 3^{12} \cdot 7^1$
109	$3,715,041,853,439$	$2^{22} \cdot 3^{11} \cdot 5$
110	$3,900,793,946,111$	$2^{20} \cdot 3^{12} \cdot 7$

List Locally Surjective Homomorphisms in Hereditary Graph Classes

Pavel Dvořák
Tata Institute of Fundamental Research, Mumbai, India
Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Tomáš Masařík
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Jana Novotná
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Monika Krawczyk
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Paweł Rzążewski
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Aneta Żuk
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Abstract

A *locally surjective homomorphism* from a graph G to a graph H is an edge-preserving mapping from $V(G)$ to $V(H)$ that is surjective in the neighborhood of each vertex in G. In the *list locally surjective homomorphism* problem, denoted by LLSHom(H), the graph H is fixed and the instance consists of a graph G whose every vertex is equipped with a subset of $V(H)$, called list. We ask for the existence of a locally surjective homomorphism from G to H, where every vertex of G is mapped to a vertex from its list. In this paper, we study the complexity of the LLSHom(H) problem in F-free graphs, i.e., graphs that exclude a fixed graph F as an induced subgraph. We aim to understand for which pairs (H, F) the problem can be solved in subexponential time.

We show that for all graphs H, for which the problem is NP-hard in general graphs, it cannot be solved in subexponential time in F-free graphs for F being a bounded-degree forest, unless the ETH fails. The initial study reveals that a natural subfamily of bounded-degree forests F, that might lead to some tractability results, is the family $\mathcal{S}$ consisting of forests whose every component has at most three leaves. In this case, we exhibit the following dichotomy theorem: besides the cases that are polynomial-time solvable in general graphs, the graphs $H \in \{P_3, C_4\}$ are the only connected ones that allow for a subexponential-time algorithm in F-free graphs for every $F \in \mathcal{S}$ (unless the ETH fails).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Homomorphism, Hereditary graphs, Subexponential-time algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.30

Related Version *Full Version*: https://arxiv.org/abs/2202.12438 [9]

Funding *Pavel Dvořák*: Partially supported by EPSRC New Investigator Award EP/V010611/1 and by Czech Science Foundation GAČR grant #22-14872O.

Tomáš Masařík: Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 948057.
Jana Novotná: Supported by SVV-2020–260578 and GAUK 384321 of Charles University.
Paweł Rzążewski: Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.

1 Introduction

Graph coloring is arguably one of the best-studied problems in algorithmic graph theory. It is well-known that k-Coloring is polynomial-time solvable for $k \leq 2$ and NP-hard for every $k \geq 3$ [27]. Furthermore, assuming the Exponential-Time Hypothesis (ETH) [25, 26], the hard cases do not even admit algorithms working in subexponential time.

Coloring F-free graphs. A very natural direction of research is to investigate what restrictions put on the family of input graphs allow us to solve the problem more efficiently than in general graphs. In recent years, a very active topic has been to study the complexity of k-Coloring and related problems in graphs defined by one or more forbidden induced subgraphs. For a family $\mathcal{F}$ of graphs, we say that a graph G is $\mathcal{F}$-free if G does not contain any graph from $\mathcal{F}$ as an induced subgraph. If $\mathcal{F}$ consists of a single graph F, then we say F-free instead of $\{F\}$-free. Note that the class of $\mathcal{F}$-free graphs is *hereditary* i.e., closed under vertex deletion. On the other hand, every hereditary class of graphs can be equivalently defined as $\mathcal{F}$-free graphs for some unique minimal (possibly infinite) family $\mathcal{F}$ of graphs.

It is well-known that for every $k \geq 3$, the k-Coloring problem is NP-hard in F-free graphs, unless F is a *linear forest*, i.e., every connected component of F is a path. Indeed, for every constant g, k-Coloring is NP-hard in graphs of girth (i.e., the length of a shortest cycle) at least g [10]. Setting $g = |V(F)| + 1$, we immediately obtain hardness for every F that is not a forest. On the other hand, k-Coloring is NP-hard in line graphs, which are claw-free [23, 29]. The only forests that are claw-free are linear forests.

The complexity of k-Coloring in P_t-free graphs, where P_t is the path with t vertices, has recently attracted a lot of attention. For $t = 5$, the problem is polynomial-time solvable for every constant k [22]. If $k \geq 5$, then the problem is NP-hard already in P_6-free graphs [24]. The case $k = 4$ is also fully understood: it is polynomial-time solvable for $t \leq 6$ [38] and NP-hard for $t \geq 7$ [24]. The case of $k = 3$ is much more elusive. We know a polynomial-time algorithm for P_7-free graphs [1]. However, for $t \geq 8$, we know neither polynomial-time algorithm nor any hardness result. Some positive results are also known for the case that F is a disconnected linear forest [28, 6, 18].

Let us point out that almost all mentioned algorithmic results also hold for the more general *list* variant of the problem, where each vertex is given a list of admissible colors. The notable exception is List 4-Coloring, which is NP-hard already in P_6-free graphs [19]. Furthermore, all hardness results also imply the nonexistence of subexponential-time algorithms (assuming the ETH).

Some more general positive results can be obtained if we relax our notion of tractability. As observed by Groenland et al. [20], List 3-Coloring can be solved in subexponential time in P_t-free graphs, for every fixed t. This was recently improved by Pilipczuk, Pilipczuk, and Rzążewski [37] who showed a quasipolynomial-time algorithm for this problem. Note that this is strong evidence that the problem is not NP-hard.

Graph homomorphisms. Graph colorings can be seen as a special case of *graph homomorphisms*. A homomorphism from a graph G to a graph H (with possible loops) is a function $h : V(G) \to V(H)$, such that for every $uv \in E(G)$ it holds that $h(u)h(v) \in E(H)$. Note that

homomorphisms to K_k are precisely proper k-colorings. By the celebrated result of Hell and Nešetřil [21], determining whether an input graph G admits a homomorphism to a fixed graph H is polynomial-time solvable if H is bipartite or has a vertex with a loop, and NP-hard otherwise. A list variant of the graph homomorphism problem, denoted by LHOM(H), has also been considered. It turns out that the problem can be solved in polynomial time if H is a so-called *bi-arc graph*, and otherwise, the problem is NP-hard [12, 13, 14].

The complexity of variants of the graph homomorphism problem in hereditary graph classes was also studied. For example, Chudnovsky et al. [5] showed that LHOM(C_k) for $k \in \{5,7\} \cup [9,\infty)$ is polynomial-time solvable in P_9-free graphs. On the negative side, they showed that for every $k \geq 5$ the problem is NP-hard and cannot be solved in subexponential time (assuming the ETH) in F-free graphs, unless every component of $F \in \mathcal{S}$, where $\mathcal{S}$ consists of graphs whose every connected component is a path or a tree with three leaves (called a *subdivided claw*). This negative result was later extended by Piecyk and Rzążewski [36] who showed that if H is not a bi-arc graph (i.e., LHOM(H) is NP-hard in general graphs), then LHOM(H) is NP-hard and cannot be solved in subexponential time (assuming the ETH) in F-free graphs, unless $F \in \mathcal{S}$.

The case of forbidden path or subdivided claw was later investigated by Okrasa and Rzążewski [35]. They defined a class of *predacious graphs* and showed that if H is not predacious, then for every H, the LHOM(H) problem can be solved in quasipolynomial time in P_t-free graphs (for every t). Otherwise, for every H, there exists t for which LHOM(H) cannot be solved in subexponential time in P_t-free graphs unless the ETH fails. They also provided some partial results for the case of forbidden subdivided claws.

The complexity of variants of the graph homomorphism problem in other hereditary graph classes has also been considered [7, 15, 34].

Locally surjective graph homomorphisms. Graph homomorphisms are a very robust notion, which can be easily extended by putting some additional restrictions on the solution. In this paper, we focus on one such variant called *locally surjective homomorphisms*. A homomorphism h from G to H is locally surjective if it is surjective in the neighborhood of each vertex of G. In other words, if $h(v) = a \in V(H)$, then for every neighbor b of a in H (including a, if it has a loop) there is a neighbor v' of v in G, such that $h(v') = b$. The study of locally surjective homomorphisms originates in social sciences, where they can be used to model some social roles (the problem is called *role assignment* [11]). The problem of determining whether an input graph admits a locally surjective homomorphism to a fixed graph H is denoted by LSHOM(H). Fiala and Paulusma [17] provided the full complexity dichotomy for LSHOM(H). For simplicity, let us consider only connected graphs H, and let K_1° be the one-vertex graph with a loop. We denote $\mathcal{H}_{\text{poly}} := \{K_1, K_1^\circ, K_2\}$. Fiala and Paulusma [17] showed that LSHOM(H) is polynomial-time-solvable if $H \in \mathcal{H}_{\text{poly}}$, and otherwise it is NP-hard. Again, the hardness reduction excluded also subexponential time algorithms under the ETH.

Let us point out that LSHOM(P_3) is closely related to the well-known hypergraph 2-coloring problem [30] (or, equivalently, POSITIVE NAE SAT). In this problem, we ask whether the input hypergraph admits a 2-coloring of its vertices which makes no edge monochromatic. Consider a hypergraph $\mathbf{H}$ with vertices $\mathcal{V}$ and hyperedges $\mathcal{E}$, and let G be its *incidence graph*, i.e., the bipartite graph with vertex set $\mathcal{V} \cup \mathcal{E}$, where $v \in \mathcal{V}$ is adjacent to $e \in \mathcal{E}$ if and only if $v \in e$. Note that proper 2-colorings of $\mathbf{H}$ are precisely locally surjective homomorphisms of G to P_3 with consecutive vertices $1, 2, 3$, where $\mathcal{V}$ is mapped to $\{1, 3\}$, and $\mathcal{E}$ is mapped to $\{2\}$. As shown by Camby and Schaudt [3], 2-coloring of hypergraphs with P_7-free incidence graph is polynomial-time solvable.

The structural and computational aspects of locally surjective homomorphisms were studied by several authors [4, 16, 2]. However, up to the best of our knowledge, no systematic study of LSHom(H) in hereditary graph classes has been conducted.

Our contribution. In this paper, we consider the complexity of the *list* variant of LSHom(H), called LLSHom(H). First, we observe that if $H \in \mathcal{H}_{\text{poly}}$ (recall, these are the easy cases of LSHom(H)), then also LLSHom(H) can be solved in polynomial time in general graphs.

Then we focus on the complexity of the problem in F-free graphs. In particular, we are interested in determining the pairs (H, F), for which the problem can be solved in subexponential time. Similarly to the case of LHom(H), we split into two cases, depending whether $F \in \mathcal{S}$.

In the first case, we identify two more positive cases: we show that if $H \in \{P_3, C_4\}$, then the problem admits a subexponential-time algorithm for every $F \in \mathcal{S}$. The algorithm itself uses a win-win strategy: we combine branching on a high-degree vertex with a separator theorem that can be used if the maximum degree is bounded. A similar approach was used for various other problems [20, 31], however, the specifics of our problem require a slightly more complicated approach.

We also show that the above cases are the only positive ones for general $F \in \mathcal{S}$, which provides the following dichotomy theorem.

▶ **Theorem 1.** *Let $H \notin \mathcal{H}_{\text{poly}}$ be a fixed connected graph.*

1. *If $H \in \{P_3, C_4\}$, then for every $F \in \mathcal{S}$, the LLSHom(H) problem can be solved in time $2^{\mathcal{O}((n \log n)^{2/3})}$ in n-vertex F-free graphs.*
2. *Otherwise there is t, such that the LLSHom(H) problem cannot be solved in subexponential time in P_t-free graphs, unless the ETH fails.*

Further, we turn our attention to other forbidden graphs F. We show that whenever the problem is NP-hard for general graphs, i.e., for every $H \notin \mathcal{H}_{\text{poly}}$, then for every $g > 0$ there exists $d = d(H)$, such that LSHom(H) is NP-hard in graphs of degree at most d and girth at least g. This implies the following lower bound.[1]

▶ **Theorem 2 (♠).** *For every connected $H \notin \mathcal{H}_{\text{poly}}$, there exists $d \in \mathbb{N}$, such that the following holds. For every graph F that is not a forest of maximum degree at most d, the LLSHom(H) problem cannot be solved in time $2^{o(n)}$ in n-vertex F-free graphs of maximum degree d, unless the ETH fails.*

We conclude the paper by discussing the possibilities of improving our theorems in order to fully classify the complexity of LSHom(H) in F-free graphs.

2 Preliminaries

For a graph G and $v \in V(G)$, by $N_G(v)$ we denote the set of neighbors of v in G. For a set $X \subseteq V(G)$, by $N_G[X]$ we denote $X \cup \bigcup_{v \in X} N_G(v)$. If G is clear from the context, then we omit the subscripts. By $G[X]$, where $X \subseteq V(G)$, we denote the subgraph of G induced by the set X.

[1] Proofs of statements marked with (♠) are presented in the full version [9].

For graphs G and H, by $G \times H$ we denote their *direct product* (sometimes called categorical product or Kronecker product), i.e., the graph

$$V(G \times H) = V(G) \times V(H),$$
$$E(G \times H) = \big\{\{(u_1, v_1), (u_2, v_2)\} \mid u_1u_2 \in E(G) \wedge v_1v_2 \in E(H)\big\}.$$

For $t, a, b, c \geq 1$, by P_t we denote the t-vertex path, and by $S_{a,b,c}$ we denote the three-leaf tree with leaves at distance a, b, and c, respectively, from the unique vertex of degree 3, which we call *central*. Every such $S_{a,b,c}$ is called a *subdivided claw*. Recall that by $\mathcal{S}$, we denote the family of graphs whose every connected component is either a path or a subdivided claw.

Let h be a homomorphism from G to H. We say that a vertex $v \in V(G)$ is *happy* (in h) if $h(N_G(v)) = N_H(h(v))$. In other words, for every neighbor y of $h(v)$, some neighbor of v is colored y. We say that a homomorphism h is *locally surjective* if every vertex is happy in h. If h is a locally surjective homomorphism from G to H, then we denote it by $h : G \xrightarrow{s} H$.

For a fixed graph H (with possible loops) we consider the LLSHom(H) problem, whose instance is (G, L) where G is a graph and $L : V(G) \to 2^{V(H)}$ is a list function. We ask whether there exists a homomorphism $h : G \xrightarrow{s} H$, such that for every $v \in V(G)$ it holds that $h(v) \in L(v)$. If h is such a homomorphism, then we denote it by $h : (G, L) \xrightarrow{s} H$.

Observe that if G or H is disconnected, then each component of G must be mapped to some component of H. Thus, the problem can be easily reduced to the case that both G and H are connected. We will assume this from now on.

Recall that the non-list variant of our problem, i.e., LSHom(H), is polynomial time-solvable if $H \in \mathcal{H}_{\text{poly}} := \{K_1, K_2, K_1^\circ\}$ (where K_1° denotes the one-vertex graph with a loop), and NP-hard otherwise [17]. Let us point out that exactly the same dichotomy holds for LLSHom(H).

▶ **Corollary 3 (♠).** *If $H \in \mathcal{H}_{\text{poly}}$, then LLSHom($H$) is polynomial-time solvable, and otherwise it is NP-hard.*

2.1 Associated Bipartite Graphs and Associated Instances

Now let us show that in order to identify the hard cases of LLSHom(H) it is sufficient to consider the case that H is bipartite. A similar approach was used to solve LHom(H) [14, 32], but to the best of our knowledge, we are the first to observe that it also works for LLSHom(H).

Let H be a connected bipartite graph with bipartition classes X, Y, and consider an instance (G, L) of LLSHom(H), where G is connected. Note that if G is not bipartite, then (G, L) is clearly a no-instance. Thus, assume that G is bipartite with the bipartition classes A, B. We observe that in every homomorphism $h : G \to H$, either all vertices of A are mapped to X, and all vertices of B are mapped to Y, or all vertices of A are mapped to Y, and all vertices of B are mapped to X. Thus in order to solve (G, L), we can consider these two cases separately. More specifically, we need to solve two instances (G, L_1) and (G, L_2) of LLSHom(H), where

$$L_1(v) = \begin{cases} L(v) \cap X & \text{if } v \in A, \\ L(v) \cap Y & \text{if } v \in B, \end{cases} \qquad L_2(v) = \begin{cases} L(v) \cap Y & \text{if } v \in A, \\ L(v) \cap X & \text{if } v \in B. \end{cases}$$

This motivates the following definition, see also [32].

▶ **Definition 4.** *Let H be a connected bipartite graph with bipartition classes X, Y. We say that an instance (G, L) of LLSHom(H) is consistent, if*

1. *G is connected bipartite with the bipartition classes A, B,*
2. *$L(A) \subseteq X$ and $L(B) \subseteq Y$.*

For a graph $H = (V, E)$, by $H^* := H \times K_2$ we denote its *associated bipartite graph*. In other words, the vertex set of H^* is $\{v', v'' \ : \ v \in V\}$ and the edge set is $\{u'v'' \ : \ uv \in E\}$. We also define $V' := \{v' \ : \ v \in V\}$ and $V'' := \{v'' \ : \ v \in V\}$, i.e., V', V'' are the bipartition classes of H^*.

Note that if H is connected and nonbipartite, then H^* is connected. If H is bipartite, then H^* consists of two disjoint copies of H.

▶ **Lemma 5.** *Let H be a fixed connected nonbipartite graph. Let (G, L') be a consistent instance of LLSHom(H^*). For each $v \in V(G)$, define $L(v) := \{x : \{x', x''\} \cap L'(v) \neq \emptyset\}$. Then (G, L') is a* yes*-instance of LLSHom(H^*) if and only if (G, L) is a* yes*-instance of LLSHom(H).*

Proof. First consider $h^* : (G, L') \xrightarrow{s} H^*$. Define $h : V(G) \to V(H)$ as follows: $h(v) = x$ if and only if $h^*(v) \in \{x', x''\}$. Clearly, h is a homomorphism from G to H, and it respects lists L.

Let us show that h is locally surjective. Consider $v \in V(G)$ such that $h(v) = x$ and some $y \in N_H(x)$. Since $h(v) = x$, we know that $h^*(v) \in \{x', x''\}$ (the actual value depends on the bipartition class where v belongs). By symmetry, assume that $h^*(v) = x'$. Since h^* is locally surjective and $x'y'' \in E(H^*)$, there is $u \in N_G(v)$, such that $h^*(u) = y''$. Then, $h(u) = y$.

Now, consider $h : (G, L) \xrightarrow{s} H$. Let the bipartition classes of G be A, B, such that $L'(A) \subseteq V'$ and $L'(B) \subseteq V''$ (this holds since (G, L') is consistent). We define $h^* : V(G) \to V(H^*)$ as follows. Consider $v \in V(G)$ and let $h(v) = x$. If $v \in A$, then $h^*(v) = x'$ and if $v \in B$, then $h^*(v) = x''$. Again, it is straightforward to verify that h^* is a homomorphism from G to H^*, and it respects lists L', since (G, L') is consistent.

Now, let us argue that h^* is locally surjective. By symmetry, consider $v \in A$, such that $h(v) = x$. Then, $h^*(v) = x'$. Let $y'' \in N_H(x')$. Since h is locally surjective, there is $u \in N_G(v) \subseteq B$, such that $h(u) = y$. Thus, $h^*(u) = y''$. ◀

▶ **Corollary 6 (♠).** *Let $\mathcal{G}$ be a class of graphs and let H be a fixed connected nonbipartite graph. Suppose there is an algorithm A that solves every bipartite instance (G, L) of LLSHom(H), such that $G \in \mathcal{G}$, in time $f(|V(G)|)$. Then there is an algorithm that solves every instance (G, L') of LLSHom(H^*), where $G \in \mathcal{G}$, in time $f(|V(G)|) \cdot |V(G)|^{\mathcal{O}(1)}$.*

Note that Corollary 6 immediately implies the following.

▶ **Corollary 7.** *Let H be a fixed connected nonbipartite graph and let $\mathcal{G}$ be a class of graphs. If LLSHom(H^*) cannot be solved in time $2^{o(n)}$ for n-vertex instances in $\mathcal{G}$, then LLSHom(H) cannot be solved in time $2^{o(n)}$ for n-vertex instances in $\mathcal{G}$.*

3 Algorithm for F-free Graphs for $F \in \mathcal{S}$

An important tool used in our algorithm is the following structural result about $\{S_{t,t,t}, K_3\}$-free graphs (note that it only appears in the full version of [35]).

▶ **Theorem 8** (Okrasa, Rzążewski [33])**.** *Let $t \geq 2$ be an integer. Given an n-vertex $(K_3, S_{t,t,t})$-free graph G with maximum degree Δ, in time $2^{\mathcal{O}(t \cdot \Delta)} \cdot n$ we can find a tree decomposition of G with width at most $56t\Delta$.*

Equipped by this, we are ready to prove the following algorithmic result.

▶ **Theorem 9.** *Let $a, b, c \geq 1$ be fixed integers. The LLSHom(P_3) in n-vertex $S_{a,b,c}$-free graphs can be solved in time $2^{\mathcal{O}((n \log n)^{2/3})}$.*

Proof. Denote the consecutive vertices of P_3 by $1, 2, 3$. Let (G, L) be an instance of LLSHOM(P_3), where G has n vertices and is $S_{a,b,c}$-free. Let $t = \max(2, a, b, c)$ and note that G is $S_{t,t,t}$-free. Furthermore, if G is not bipartite, then (G, L) is clearly a no-instance. Thus, we can assume that G is bipartite (and, in particular, triangle-free).

Furthermore, recall that we can safely assume that the instance (G, L) is consistent. Let X and Y denote the bipartition classes of G, such that $L(X) \subseteq \{1, 3\}$ and $L(Y) = \{2\}$. Note that if $|Y| = 0$ or $|X| \leq 1$, then we are clearly dealing with a no-instance. Thus from now on, let us assume otherwise. In particular, it means that every vertex from X is happy in every list homomorphism from G to P_3. Consequently, our task boils down to choosing colors for vertices of X to make each vertex from Y happy.

Actually, we will design a recursive algorithm that solves a slightly more general problem, where we are additionally given a function $\sigma : Y \to 2^{\{1,3\}}$. We are looking for a list homomorphism $h : (G, L) \to P_3$, such that for every $y \in Y$ it holds that $\sigma(y) \subseteq h(N_G(y))$. Initially, we have $\sigma(y) = \{1, 3\}$ for every $y \in Y$. Thus, the returned homomorphism is indeed locally surjective. During the course of the algorithm, we will modify the sets σ to keep track of the colors seen by vertices in Y in the part of the graph that was removed.

Each recursive call starts with a preprocessing phase. First, we exhaustively apply the following steps. If there is some $x \in X$ with $L(x) = \emptyset$, then we immediately terminate the recursive call and report a no-instance. If there is some $y \in Y$ with $\sigma(y) = \emptyset$, then we can safely remove y from the graph. If there is some $x \in X$ with $|L(x)| = 1$, then we remove the element of $L(x)$ from the sets σ of all neighbors of x, and remove x from the graph.

If none of the above steps can be applied and G has an isolated vertex $y \in Y$ (note that $\sigma(y) \neq \emptyset$), then we terminate and report a no-instance. The preprocessing phase can clearly be performed in polynomial time.

Finally, if the graph obtained is disconnected, then we apply the following reasoning to every connected component independently. Let us still denote the instance by (G, L, σ), and assume that G is connected.

We consider two cases. First, suppose that there is $x \in X$ with $\deg x > (n \log n)^{1/3}$. We branch on choosing the color for x, i.e., we perform two recursive calls of the algorithm, in one branch setting $L(x) = \{1\}$, and in the other $L(x) = \{3\}$. Note that at least $\deg x/3 \geq (n \log n)^{1/3}/3$ neighbors of x have the same set σ. Consequently, in at least one branch, the sets σ will be reduced for at least $(n \log n)^{1/3}/3$ vertices during the preprocessing phase. In the other branch, we are guaranteed to have a little progress, too: the vertex x will be removed from the graph. Let us define the measure μ of the instance as $\mu := \sum_{x \in X} |L(x)| + \sum_{y \in Y} |\sigma(y)|$. Clearly $n \leq \mu \leq 2n$. Thus the complexity of this step is given by the following recursive inequality:

$$F(\mu) \leq F(\mu - 2) + F(\mu - (n \log n)^{1/3}/3) = \mu^{\mathcal{O}(\mu/(n \log n)^{1/3})} = 2^{\mathcal{O}((n \log n)^{2/3})}.$$

So now let us assume that for each $x \in X$ it holds that $\deg x < (n \log n)^{1/3}$. Let Y' be the set of vertices $y \in Y$ satisfying $\deg y \geq (n \log n)^{2/3}$. Observe that $|E(G)| \leq |X| \cdot (n \log n)^{1/3} \leq n^{4/3} \log^{1/3} n$. Consequently, $|Y'| \leq |E(G)|/(n \log n)^{2/3} \leq n^{2/3}$.

Consider the graph $G' := G - Y'$. As it is an induced subgraph of G, it is $(K_3, S_{t,t,t})$-free. Furthermore, the maximum degree of G' is at most $(n \log n)^{2/3}$. Consequently by Theorem 8, in time $2^{\mathcal{O}((n \log n)^{2/3})}$ we can find a tree decomposition of G' with width $\mathcal{O}((n \log n)^{2/3})$. Let us modify this tree decomposition by adding the set Y' to every bag – this way we obtain a tree decomposition of G with width $\mathcal{O}((n \log n)^{2/3} + n^{2/3}) = \mathcal{O}((n \log n)^{2/3})$.

Using fairly standard dynamic programming on a tree decomposition, we can solve our auxiliary problem on graphs given with a tree decomposition of width w in time $2^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)}$. Indeed, the state of the dynamic programming is the coloring of the vertices from X in

the current bag and the colors seen by the vertices from Y in the subgraph induced by the subtree rooted at the current bag (these colors are reflected in sets σ). Thus, the total number of states to consider is at most 3^w (two possibilities for a vertex from X and at most three for a vertex from Y).

Consequently, in the second case we obtain the running time $2^{\mathcal{O}((n \log n)^{2/3})} + 2^{\mathcal{O}((n \log n)^{2/3})} = 2^{\mathcal{O}((n \log n)^{2/3})}$.

Summing up, the overall complexity of the algorithm is $2^{\mathcal{O}((n \log n)^{2/3})}$. This completes the proof. ◀

Note that every P_t-free graph is also, e.g., $S_{t,1,1,}$-free, so Theorem 9 can also be applied to P_t-free graphs. Now, let us show a slight generalization of Theorem 9 to the case that we exclude a forest of paths and subdivided claws.

▶ **Theorem 10.** *For every $F \in \mathcal{S}$, the LLSHOM(P_3) problem in n-vertex F-free graphs can be solved in time $2^{\mathcal{O}((n \log n)^{2/3})}$.*

Sketch of proof. Let $F = F_1 + F_2 + \ldots + F_p$ for some $p \geq 1$, where each F_i is a subdivided claw.

We begin similarly to the algorithm from Theorem 9. Again, we are solving an auxiliary problem with instance (G, L, σ). First, we check if the instance graph is bipartite, and otherwise, we reject it. Let the bipartition classes of G be X and Y, and let $L(X) \subseteq \{1, 3\}$ and $L(Y) = \{2\}$. Then, we perform the preprocessing phase and the branching phase; note that in these phases, we do not assume anything about the forbidden induced graph. The recursion tree has $2^{\mathcal{O}((n \log n^{2/3})}$ leaves, each corresponds to an instance which is (K_3, F)-free and every vertex from X has maximum degree at most $(n \log n)^{1/3}$. Consider one such instance, for simplicity let us call it (G, L).

We continue as in the proof of Theorem 9 by selecting the set $Y' \subseteq Y$ of vertices of degree at least $(n \log n)^{2/3}$. Recall that $|Y'| \leq n^{2/3}$ and the graph $G' - Y'$ is of maximum degree at most $(n \log n)^{2/3}$.

Now for each $i = 1, \ldots, p-1$ we perform the following steps. Let (G', L') be an instance corresponding to a leaf of the recursion tree, with the set Y' removed. We check if G' contains F_i as an induced subgraph, this can be done in polynomial time by the exhaustive enumeration. If not, then G' is (K_3, F_i)-free and we can call the algorithm given by Theorem 8 and continue exactly as in the proof of Theorem 9.

Thus, let us suppose that there is $S \subseteq V(G')$, such that $G'[S] \simeq F_i$. We observe that $|N[S]| = \mathcal{O}((n \log n)^{2/3})$. We exhaustively guess the coloring of $N[S] \cap X$, this results in $2^{\mathcal{O}((n \log n)^{1/3})}$ branches. In each branch, we update the sets σ for the neighbors of colored vertices; in particular we reject if some vertex from $y \in S \cap Y$ does not see some color in $\sigma(y)$. Note that each instance is $(K_3, F_{i+1} + \ldots, +F_p)$-free.

After the last iteration, the instances corresponding to the leaves of the recursion tree are (K_3, F_p)-free, and thus we continue as in the proof of Theorem 9, i.e., use Theorem 8, restore the set Y', and solve the problem by dynamic programming.

The total number of leaves of the recursion tree is at most (here c_1, c_2 are constants)

$$\underbrace{2^{c_1 \cdot (n \log n^{2/3})}}_{\substack{\text{branching on} \\ \text{a high-degree vertex in } X}} \cdot \prod_{i=1}^{p-1} \underbrace{2^{c_2 \cdot (n \log n^{2/3})}}_{\substack{\text{branching on the neighborhood} \\ \text{of an induced copy of } F_i}} = 2^{\mathcal{O}((n \log n^{2/3})},$$

each of which corresponds to an instance that is (K_3, F_i)-free for some $i \in [p]$, and thus can be solved in time $2^{\mathcal{O}((n \log n^{2/3})}$ as in Theorem 9. ◀

Now, let us show that the algorithm from Theorem 9 can be used to solve LLSHOM(C_4) in P_t-free graphs. The proof of the following lemma is based on a similar argument used by Okrasa and Rzążewski [34] in the non-list case.

▶ **Lemma 11.** *Let (G, L) be a consistent instance of LLSHOM(C_4). Then the problem can be reduced in polynomial time to solving two consistent instances of LLSHOM(P_3).*

Proof. Let the consecutive vertices of C_4 be $1, 2, 3, 4$. Let the bipartition classes of G be X, Y, and let $L(X) \subseteq \{1, 3\}$ and $L(Y) \subseteq \{2, 4\}$.

Define L', L'' as follows:

$$L'(v) = \begin{cases} L(v) & \text{if } v \in X, \\ \{2\} & \text{if } v \in Y, \end{cases} \quad \text{and} \quad L''(v) = \begin{cases} \{1\} & \text{if } v \in X, \\ L(v) & \text{if } v \in Y. \end{cases}$$

We claim that (G, L) is a `yes`-instance of LLSHOM(C_4) if and only if (G, L') is a `yes`-instance of LLSHOM($C_4[1, 2, 3]$) and (G, L'') is a `yes`-instance of LLSHOM($C_4[4, 1, 2]$). Note that both graphs $C_4[1, 2, 3]$ and $C_4[4, 1, 2]$ are induced three-vertex paths.

First, suppose that there is some $h : (G, L) \xrightarrow{s} C_4$. Let us define $h', h'' : V(G) \to \{1, 2, 3, 4\}$ as follows:

$$h'(v) = \begin{cases} h(v) & \text{if } v \in X, \\ 2 & \text{if } v \in Y, \end{cases} \quad \text{and} \quad h''(v) = \begin{cases} 1 & \text{if } v \in X, \\ h(v) & \text{if } v \in Y. \end{cases}$$

Let us argue that $h' : (G, L') \xrightarrow{s} C_4[1, 2, 3]$ and $h'' : (G, L'') \xrightarrow{s} C_4[4, 1, 2]$. We prove only the first claim. The proof of the second one is analogous. First, h' is clearly a homomorphism. Furthermore, it satisfies the lists, as for $x \in X$ we have $h'(x) = h(x) \in L'(x) = L(x)$, and for $y \in Y$ we have $h'(y) = 2 \in \{2\} = L'(y)$. Finally, let us argue that h' is locally surjective. For each $x \in X$, there are some y, y' such that $h(y) = 2$ and $h(y') = 4$, as otherwise h is not locally surjective. Thus, $h'(y) = h'(y') = 2$, which makes x happy in h'. Similarly, for each $y \in Y$, there are some x, x' such that $h(x) = 1$ and $h(x') = 3$, as otherwise h is not locally surjective. Thus, $h'(x) = 1$ and $h'(x') = 3$, which makes y happy in h'.

Now, suppose there are $h' : (G, L') \xrightarrow{s} C_4[1, 2, 3]$ and $h'' : (G, L'') \xrightarrow{s} C_4[4, 1, 2]$. We define $h : V(G) \to \{1, 2, 3, 4\}$ as follows:

$$h(v) = \begin{cases} h'(v) & \text{if } v \in X, \\ h''(v) & \text{if } v \in Y. \end{cases}$$

Let us argue that $h : (G, L) \xrightarrow{s} C_4$.

First, note that h is a homomorphism, as for every edge $xy \in E(G)$, where $x \in X$ and $y \in Y$, we have $h(x) \in \{1, 3\}$ and $h(y) \in \{2, 4\}$.

Now, observe that h respects the lists L. Indeed, for $x \in X$ we have $h(x) = h'(x) \in L'(x) = L(x)$ and for $y \in Y$ we have $h(y) = h''(y) \in L''(y) = L(y)$.

Finally, let us argue that h is locally surjective. Consider some $x \in X$; the argument for vertices in Y is symmetric. Note that $h''(x) = 1$. Since h'' is locally surjective, x has two neighbors y, y', such that $h''(y) = 2$ and $h''(y') = 4$. Consequently, $h(y) = 2$ and $h(y') = 4$, which makes x happy in h. This completes the proof. ◀

Combining Lemma 11 with Theorem 10, we immediately obtain the following.

▶ **Theorem 12.** *For every $F \in \mathcal{S}$, the LLSHOM(C_4) in n-vertex F-free graphs can be solved in time $2^{\mathcal{O}((n \log n)^{2/3})}$.*

4 Hardness for F-free Graphs for $F \in \mathcal{S}$

In this section, we will prove the hardness part of Theorem 1, i.e., if H is connected and $H \notin \mathcal{H}_{\text{poly}} \cup \{P_3, C_4\}$ then there is t such that LLSHOM(H) cannot be solved in subexponential time in P_t-free graphs. It easily follows that such H contains at least one of $K_2^\circ, K_2^{\circ\circ}, K_3, P_4, K_{1,3}$ as an induced subgraph, where K_2° and $K_2^{\circ\circ}$ are graphs consisting of an edge with a loop at one or both of its endpoint, respectively. First, we will prove the theorem for several base cases for $H \in \{K_{1,3}, P_4, K_2^{\circ\circ}\}$.

▶ **Theorem 13 (♠).** *For $H \in \{K_{1,3}, P_4, K_2^{\circ\circ}\}$ the LLSHOM(H) problem cannot be solved in time $2^{o(n)}$ in n-vertex P_{14}-free graphs, unless the ETH fails. Moreover, the problem is hard even for instances where all the lists are of size at most 2, and each vertex with a list of size exactly two has a neighbor with a list of size exactly one.*

Then, we will generalize the result for H containing an induced subgraph $H' \in \{K_{1,3}, P_4, K_2^{\circ\circ}\}$. The last cases when H contains K_2° or K_3 as an induced subgraph will follow from the base cases and Corollary 7 as for such H the graph H^* contains P_4 as an induced subgraph.

4.1 Proof of the Hardness Part of Theorem 1

First, we show a lemma that helps us extend the hardness reductions to all graphs in the second part of Theorem 1.

▶ **Lemma 14.** *Let H be a graph without isolated vertices and let $u, v \in V(H)$. There exists a graph $Z := H \times H$ with lists L and $z \in V(Z)$ with $L(z) = \{u, v\}$ such that there are at least two homomorphism $h_u, h_v : (Z, L) \xrightarrow{s} H$ such that $h_u(z) = u$ and $h_v(z) = v$.*

Proof. First, we define z as $(u, v) \in V(Z)$ and set $L(z)$ appropriately. We do not restrict other lists of $V(Z)$. For vertices $(a, b) \in V(Z)$ we define $h_u((a, b)) := a$ and $h_v((a, b)) := b$. We verify that h_u is indeed a locally surjective list homomorphism. Take an edge $(a, b)(c, d) \in E(Z)$. We infer that $ac \in E(H)$. For $(a, b) \in V(Z)$ there exist $d \in N_H(b)$ as no vertex is isolated. Vertex (a, b) is happy as for each $c \in N_H(a)$ we have an edge $(a, b)(c, d) \in E(Z)$ and so vertex (a, b) is happy. The proof for h_v is analogous. ◀

Now, we are ready to prove the hardness part of Theorem 1. In particular, we will show the following theorem.

▶ **Theorem 15.** *Let $H \notin \mathcal{H}_{\text{poly}} \cup \{P_3, C_4\}$ be a connected graph. Let q be the number of vertices in the longest induced path in $H \times H$. There exists $t \leq 14 + 2q$ such that LLSHOM(H) cannot be solved in time $2^{o(n)}$ in n-vertex P_t-free graphs, unless the ETH fails.*

Proof. Let $H \notin \mathcal{H}_{\text{poly}} \cup \{P_3, C_4\}$ be connected. As we stated above, the graph H contains at least one of $K_2^\circ, K_2^{\circ\circ}, K_3, P_4, K_{1,3}$ as an induced subgraph H'. Theorem 13 proved the cases when $H \in \{K_{1,3}, P_4, K_2^{\circ\circ}\}$.

First, suppose that $H' \in \{K_{1,3}, P_4, K_2^{\circ\circ}\}$. We show how to adjust the hardness construction for H' to H. Recall that by Theorem 13 we can assume that the list of each vertex is of size at most two, and moreover, if v has a list of size exactly two, then v has a neighbor with a list of size one. Now, we describe how to modify an instance (G', L') of LLSHOM(H') (called *original one*) into an equivalent instance (G, L) of LLSHOM(H).

Let w be a vertex of G'. First, consider the case that $|L(w)| = 1$ where $\{c_w\} = L(w)$. Let $X_w := (V(H) \setminus V(H')) \cap N_H(c_w)$. In other words, set X_w represents the neighbors of c_w that are only in H but not in H'. We add $|X_w|$ disjoint copies of graph H into G with lists $L(y) = \{y\}$ for $y \in V(H)$. For each $c \in X_w$ we connect c in the c-th copy of H with w. We call c the *contact vertex* of the respective *additional gadget*.

Now, consider the case that $|L(w)| = 2$. Let $\{a, b\} = L(w)$. As observed there is a vertex $q_w \in N_{G'}(w)$ such that $|L(q)| = 1$ and let $\{c_q\} = L(q)$. Let $X^{ab}_w := (V(H) \setminus V(H')) \cap N_H(a) \cap N_H(b)$. Let $X^a_w := (V(H) \setminus V(H')) \cap N_H(a) \setminus X^{ab}_w$. Let $X^b_w := (V(H) \setminus V(H')) \cap N_H(b) \setminus X^{ab}_w$. Finally, let $X_w := X^b_w \cup X^a_w$. In other words, set X^{ab}_w represents common neighbors of a and b that are only in H and not in H'. Similarly, set X^a_w is composed of the neighbors of a which are not the neighbors of b, and the symmetrical is true for set X^b_w.

We add $|X^{ab}_w|$ disjoint copies of graph H into G' with lists $L(y) = \{y\}$ for $y \in V(H)$. We call those copies of H *additional gadgets*. For each $c \in X^{ab}_w$, we connect c in the c-th copy of H with w. We call c the *contact vertex*. We use Lemma 14 and we add $|X_w|$ copies of the graph Z_w into G'. We connect w with a special vertex $z \in V(Z_w)$. For each $c \in X_w$, we define $L(z) := \{c, c_q\}$ in the c-th copy of Z_w. Again, we say that c is the *contact vertex* and a copy of Z_w is a *non-trivial additional gadget*. Consult Figure 1 for an overview of the construction.

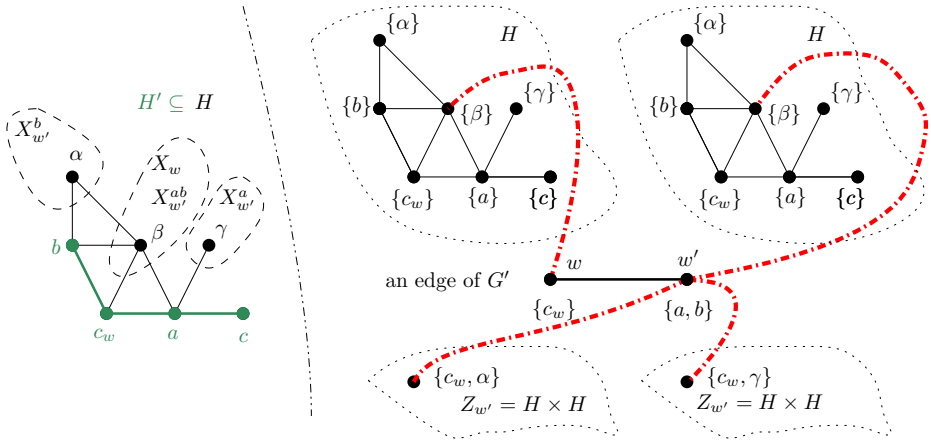


Figure 1 An example of constuction of (G, L) shown on one edge w, w' of (G', L'). The added gadgets are attached using red dash-dotted edges.

It is easy to verify that the additional gadgets always allow us to make the original vertices of the construction happy regarding the vertices outside H' as we added one copy of an additional gadget per vertex v of H' per each color ($c \in N_H(v) \setminus V(H')$) v need to see in its neighborhood in construction of G. The above is the *first property*. On the other hand, the additional gadgets never allow the contact vertices to be mapped to any vertex of H except for the ones that are already seen in the neighborhood within the original construction (which happens only in the case of non-trivial additional gadgets), we call it the *second property*. By the construction, this exception happens only when the original vertex v had a list of size two (in G') $L(v) = \{c_1, c_2\}$ and the color $c_1 \in H'$ had a private neighbor q (with respect to the other color) outside of H', i.e. $q \in N_H(c_1) \setminus N_H(c_1) \setminus V(H')$. In this case, the non-trivial additional gadget may allow mapping its contact vertex to the color of $u \in N_{G'}(v)$ which has the list of size exactly one (such a vertex always exists by the assumptions on the hardness construction of G'). Moreover, observe that regardless of what color is assigned to a vertex v in the `yes`-instance of G', there is always a valid (respecting homomorphism) color to map vertex $u \in V(G) \setminus V(G')$. We call it the *third property*. Note that we do not need to argue about vertices within the additional gadgets as the mapping for them always exists, and it makes all their vertices happy regardless of the mapping of G' by Lemma 14 for non-trivial additional gadgets (and by trivial reasons for the rest).

Whenever we have a yes-instance (G', L') of LLSHom(H'), we obtain a yes-instance (G, L) of LLSHom(H) as a conclusion of the first and third properties allowing all vertices (of G) to be happy and mapped correctly. Conversely, whenever we have a yes-instance (G, L) of LLSHom(H), we obtain a yes-instance (G', L') of LLSHom(H'). Indeed, if we restrict to the vertices of G' only (those are always mapped to H'), they must be already happy without any help from vertices in $V(G) \setminus V(G')$ by the second property (the restricted mapping is a homomorphism trivially). Therefore, we conclude that the newly constructed instance (G, L) is equivalent to the original one, i.e., (G', L').

Observe that the length of the longest induced path in G is the length of the longest path in G' plus twice the length of the longest path in $H \times H$, which we denoted as q.

It remains to show what to do if $H' \in \{K_2^\circ, K_3\}$ is an induced subgraph of H. As H is non-bipartite, we create a connected bipartite graph H^*. Now, observe that if $H' = K_2^\circ$ then H^* contains P_4. Further, if $H' = K_3$ then H^* contains C_6 and so P_4. As in the case of H^* containing a P_4, we already proved hardness, the hardness for H follows by Corollary 7. ◀

5 Concluding Remarks

Let us conclude the paper with discussing some potential ways to strengthen our results. First of all, we believe that the cases covered by the algorithmic statement in Theorem 1 are actually polynomial-time solvable. Furthermore, we think the hardness counterpart of Theorem 1, as well as Theorem 2, hold even in the non-list setting, i.e., for LSHom(H).

Next, recall that in the hardness part of Theorem 1, the length t of the forbidden induced path depends on H. One might wonder if it is possible to find t, such that for every $H \notin \mathcal{H}_{\text{poly}} \cup \{P_3, C_4\}$, the LLSHom$(H)$ problem is hard in P_t-free graphs.

Suppose that such a t exists and consider $H = P_t$ with consecutive vertices $1, \ldots, t$. Without loss of generality, we may assume that $t \geq 4$. Consider a locally surjective homomorphism h from G to P_t. Note that h is in particular surjective, so there exists a vertex v_1 mapped to 1. By local surjectivity of h, there must be a neighbor v_2 of v_1 mapped to 2, a neighbor v_3 of v_2 mapped to 3, and so on. Note that $v_1, v_2, \ldots, v_t$ is a path in G. Furthermore, this path is induced, as otherwise h is not a homomorphism. Consequently, every yes-instance of LSHom(P_t) (and thus of LLSHom(P_t)) contains an induced t-vertex path. This means that LLSHom(P_t) is polynomial-time solvable (and actually trivial) in P_t-free graphs. On the other hand, $P_t \notin \mathcal{H}_{\text{poly}} \cup \{P_3, C_4\}$, so by Theorem 1 (2.) there exists some t', for which the problem is hard in $P_{t'}$-free graphs.

Moreover, recall that in Theorem 2 the degree bound on F depends on H. Again, one might wonder if this is necessary. However, every yes-instance of LSHom(H) must contain a vertex of degree $\Delta(H)$, as some vertex v of G must be mapped to a maximum-degree vertex a of H, and all vertices from $N_H(a)$ must appear on the set $N_G(v)$. Consequently, we cannot hope for a universal upper bound on the degree of G in the proof of Theorem 2.

The above two examples show that obtaining the full characterization of pairs (H, F), for which LLSHom(H) admits a subexponential-time algorithm in F-free graphs, would be a tedious task. One can probably start with some small graphs F. Let us point out that if $F = P_4$, then LLSHom(H) is polynomial-time solvable for every H. Indeed, P_4-free graphs, also known as cographs, have bounded *cliquewidth* and the result follows from the celebrated meta-theorem for bounded-cliquewidth graphs by Courcelle, Makowsky, and Rotics [8].

References

1 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Combinatorica*, 38(4):779–801, 2018. doi:10.1007/s00493-017-3553-8.

2 Laurent Bulteau, Konrad K. Dabrowski, Noleen Köhler, Sebastian Ordyniak, and Daniël Paulusma. An algorithmic framework for locally constrained homomorphisms. *CoRR*, abs/2201.11731, 2022. doi:10.48550/arXiv.2201.11731.

3 Eglantine Camby and Oliver Schaudt. A new characterization of P_k-free graphs. *Algorithmica*, 75(1):205–217, 2016. doi:10.1007/s00453-015-9989-6.

4 Steven Chaplick, Jiří Fiala, Pim van 't Hof, Daniël Paulusma, and Marek Tesař. Locally constrained homomorphisms on graphs of bounded treewidth and bounded degree. *Theoretical Computer Science*, 590:86–95, 2015. doi:10.1016/j.tcs.2015.01.028.

5 Maria Chudnovsky, Shenwei Huang, Pawel Rzążewski, Sophie Spirkl, and Mingxian Zhong. Complexity of C_k-coloring in hereditary classes of graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 31:1–31:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.31.

6 Maria Chudnovsky, Shenwei Huang, Sophie Spirkl, and Mingxian Zhong. List 3-coloring graphs with no induced $P_6 + P_3$. *Algorithmica*, 83(1):216–251, 2021. doi:10.1007/s00453-020-00754-y.

7 Maria Chudnovsky, Jason King, Michał Pilipczuk, Paweł Rzążewski, and Sophie Spirkl. Finding large h-colorable subgraphs in hereditary graph classes. *SIAM Journal on Discrete Mathematics*, 35(4):2357–2386, 2021. doi:10.1137/20M1367660.

8 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

9 Pavel Dvořák, Monika Krawczyk, Tomáš Masařík, Jana Novotná, Paweł Rzążewski, and Aneta Żuk. List locally surjective homomorphisms in hereditary graph classes, 2022. doi:10.48550/arXiv.2202.12438.

10 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability and Computing*, 7(4):375–386, 1998. doi:10.1017/S0963548398003678.

11 Martin G. Everett and Steve Borgatti. Role colouring a graph. *Mathematical Social Sciences*, 21(2):183–188, 1991. doi:10.1016/0165-4896(91)90080-B.

12 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.

13 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.

14 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

15 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms of graphs with bounded degrees. *Discrete Mathematics*, 307(3-5):386–392, 2007. doi:10.1016/j.disc.2005.09.030.

16 Jiří Fiala and Jan Kratochvíl. Locally constrained graph homomorphisms - structure, complexity, and applications. *Computer Science Review*, 2(2):97–111, 2008. doi:10.1016/j.cosrev.2008.06.001.

17 Jiří Fiala and Daniël Paulusma. A complete complexity classification of the role assignment problem. *Journal of Computer and System Sciences*, 349(1):67–81, 2005. doi:10.1016/j.tcs.2005.09.029.

18 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of coloring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84(4):331–363, 2017. doi:10.1002/jgt.22028.

19 Petr A. Golovach, Daniël Paulusma, and Jian Song. Closing complexity gaps for coloring problems on H-free graphs. *Information and Computation*, 237:204–214, 2014. doi:10.1016/j.ic.2014.02.004.

20 Carla Groenland, Karolina Okrasa, Paweł Rzążewski, Alex Scott, Paul Seymour, and Sophie Spirkl. H-colouring P_t-free graphs in subexponential time. *Discrete Applied Mathematics*, 267:184–189, 2019. doi:10.1016/j.dam.2019.04.010.

21 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

22 Chính T. Hoàng, Marcin Kaminski, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57(1):74–81, 2010. doi:10.1007/s00453-008-9197-8.

23 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. doi:10.1137/0210055.

24 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *European Journal of Combinatorics*, 51:336–346, 2016. doi:10.1016/j.ejc.2015.06.005.

25 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

27 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

28 Tereza Klimošová, Josef Malík, Tomáš Masařík, Jana Novotná, Daniël Paulusma, and Veronika Slívová. Colouring $P_r + P_s$-free graphs. *Algorithmica*, 82(7):1833–1858, 2020. doi:10.1007/s00453-020-00675-w.

29 Daniel Leven and Zvi Galil. NP-completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4(1):35–44, 1983. doi:10.1016/0196-6774(83)90032-9.

30 László Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, volume 37 of *Utilitas Math.*, pages 3–12, 1973.

31 Jana Novotná, Karolina Okrasa, Michał Pilipczuk, Paweł Rzążewski, Erik Jan van Leeuwen, and Bartosz Walczak. Subexponential-time algorithms for finding large induced sparse subgraphs. *Algorithmica*, 83(8):2634–2650, 2021. doi:10.1007/s00453-020-00745-z.

32 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

33 Karolina Okrasa and Paweł Rzążewski. Complexity of the list homomorphism problem in hereditary graph classes. *CoRR*, abs/2010.03393, 2020. arXiv:2010.03393.

34 Karolina Okrasa and Paweł Rzążewski. Subexponential algorithms for variants of the homomorphism problem in string graphs. *Journal of Computer and System Sciences*, 109:126–144, 2020. doi:10.1016/j.jcss.2019.12.004.

35 Karolina Okrasa and Paweł Rzążewski. Complexity of the list homomorphism problem in hereditary graph classes. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 54:1–54:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.54.

36 Marta Piecyk and Paweł Rzążewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.56.

37 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for Independent Set in P_t-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

38 Sophie Spirkl, Maria Chudnovsky, and Mingxian Zhong. Four-coloring P_6-free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1239–1256. SIAM, 2019. doi:10.1137/1.9781611975482.76.

Locally Checkable Problems Parameterized by Clique-Width

Narmina Baghirova
Department of Informatics, University of Fribourg, Switzerland

Carolina Lucía Gonzalez[1]
Instituto de Investigación en Ciencias de la Computación (ICC),
CONICET-University of Buenos Aires, Argentina

Bernard Ries
Department of Informatics, University of Fribourg, Switzerland

David Schindl
Department of Informatics, University of Fribourg, Switzerland

Abstract

We continue the study initiated by Bonomo-Braberman and Gonzalez in 2020 on r-locally checkable problems. We propose a dynamic programming algorithm that takes as input a graph with an associated clique-width expression and solves a 1-locally checkable problem under certain restrictions. We show that it runs in polynomial time in graphs of bounded clique-width, when the number of colors of the locally checkable problem is fixed. Furthermore, we present a first extension of our framework to global properties by taking into account the sizes of the color classes, and consequently enlarge the set of problems solvable in polynomial time with our approach in graphs of bounded clique-width. As examples, we apply this setting to show that, when parameterized by clique-width, the $[k]-$Roman domination problem is FPT, and the k-community problem, Max PDS and other variants are XP.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Combinatorial algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases locally checkable problem, clique-width, dynamic programming, coloring

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.31

Related Version *Full Version*: `https://arxiv.org/abs/2203.02992`

Funding *Carolina Lucía Gonzalez*: This research was conducted during two research stays in the University of Fribourg financed by Universidad de Buenos Aires and University of Fribourg. CONICET PIP 11220200100084CO and UBACyT 20020170100495BA.

1 Introduction

Many graph problems can be stated as a sort of partitioning, or equivalently, as a sort of coloring problem. Furthermore, most decision problems on graphs from the literature belong to the class NP, and their certificate verification algorithms often consist in checking some *local* property for each vertex, i.e. involving itself and its neighborhood only, plus possibly some *global* property concerning, for instance, the sizes or the connectivity of some subsets of vertices. One could therefore try to cover a broad variety of these problems under a same umbrella, and hence develop efficient algorithms to solve them at once. With this objective

[1] Corresponding author.

in mind, several definitions of subsets of partitioning problems, where each vertex has to satisfy a local property, as well as extensions of these sets of problems including some global property, have been proposed and shown to be solvable in polynomial time in various graph classes. In particular, in [6], the authors defined so-called *r-locally checkable problems*. Each of these problems has an associated set of colors and a *check function*, that is, a function that takes as input a vertex v of the graph and a coloring of the r-neighborhood of v (i.e. the set of vertices at distance at most r from v) and outputs True or False. A *proper coloring* of the input graph G is defined as a coloring c of the vertices such that, for every vertex v, the check function applied to v and the restriction of c to the r-neighborhood of v outputs True. They also consider a set of weights with a total order, and associate a weight to each pair of vertex and possible color. The weight of a coloring c is then naturally obtained by combining the weights of the pairs $(v, c(v))$. Then, an r-locally checkable problem consists in finding the minimum weight of a proper coloring of the input graph G. Examples of r-locally checkable problems include k-Coloring, Maximum Independent Set and Minimum Dominating Set [6].

Since many r-locally checkable problems are hard on general graphs, it is of interest to determine under which conditions (on the check function and the set of colors) we can efficiently solve them for a given class of graphs. In [6], the authors showed that, under mild conditions, r-locally checkable problems can be solved in polynomial time in graphs of bounded tree-width. In this paper, we will focus on 1-locally checkable problems with an associated *color-counting* check function, defined as follows.

▶ **Definition 1.** *Let G be a graph and $\textsc{Colors} = \{a_1, \ldots, a_q\}$ be a set of colors. A check function f is* color-counting *if it only depends on the vertex v, the color it receives and, for each color $a \in \textsc{Colors}$, the number of neighbors of v of color a.*

In other words, a check function f is color-counting *if there exists a function f' such that*

$$f(v, c) = f'(v, c(v), n_1, \ldots, n_q)$$

for every vertex $v \in V(G)$ and every coloring c of the closed neighborhood of v, where $n_j = |\{u \in N_G(v) : c(u) = a_j\}|$ for all $j \in \{1, \ldots, q\}$.

Since we are only going to work with color-counting check functions in this paper, we will directly refer to them as $check(v, a, n_1, \ldots, n_q)$.

In [17], the authors analyzed the restrictions on 1-locally checkable problems with respect to mim-width. They define *d-stable* check functions, which are a subset of the color-counting check functions, and we will use them to improve our complexity results.

▶ **Definition 2** ([17])**.** *Let $d \in \mathbb{N}$. Let G be a graph, $\textsc{Colors}$ be a set of q colors and check be a color-counting check function. We say that check is* d-stable *if for all $v \in V(G)$, $a \in \textsc{Colors}$ and non-negative integers $n_1, \ldots, n_q$ we have*

$$check(v, a, n_1, \ldots, n_q) = check(v, a, \min(d, n_1), \ldots, \min(d, n_q)).$$

We present a dynamic programming algorithm, which is XP parameterized by clique-width, for 1-locally checkable problems with a constant number of colors and a color-counting check function. Moreover, this algorithm is FPT when the check function is also d-stable, for any constant d. In a second step, we extend our framework in such a way that we are able to ensure that the size of a given color class belongs to some predefined set of integers. By including this global property for as many colors classes as necessary, the application of our

framework allows to obtain first XP algorithms parameterized by clique-width for problems such as k-COMMUNITY, MAX PDS and (GLOBAL) $[k]$-ROMAN DOMINATION, as well as some variants of them. A generalization of this framework to r-locally checkable problems, for any fixed r, would be quite natural, and the authors of this paper are currently working on it.

The set of locally checkable problems considered here is a subset of the one considered in [6], but notice that our assumptions above are not too restrictive. Indeed, if we do not impose these assumptions then, as explained in [6], one obtains locally checkable problems that are NP-hard on complete graphs (which have clique-width at most 2) and thus, it is unlikely to find XP algorithms parameterized by clique-width for this more general class of locally checkable problems.

As mentioned above, several definitions of subsets of partitioning problems have been defined in the literature and shown to be solvable in polynomial time in various graph classes. We cite here some of the corresponding publications that are the most closely related to our work.

In [7, 20, 27], the authors studied a large class of vertex partitioning problems called *locally checkable vertex partitioning* (LCVP) problems. In these problems, a $q \times q$ matrix D is given, where each entry is a finite or cofinite set of integers. A partition of the set of vertices $V_1, \ldots, V_q$ is sought, such that for each $i, j \in \{1, \ldots, q\}$, we have $|N_G(v) \cap V_j| \in D[i, j]$ for all $v \in V_i$. Empty partition classes are allowed. In [27], Telle and Proskurowski solved these problems in polynomial time on graphs of bounded treewidth. This result was generalized in [7], where Bui-Xuan, Telle and Vatshelle gave an algorithm that solves LCVP problems given a decomposition tree of the input graph. In the same paper, they proved that this algorithm is FPT parameterized by boolean-width, and later in [20], Jaffke et al. showed that the same algorithm is XP parameterized by mim-width, when a suitable decomposition tree is given. As shown in [17], every LCVP problem can be modeled as a 1-locally checkable problem with a d-stable check function (where d is as defined in [7]):

$$check(v, a, n_1, \ldots, n_q) = (\forall j \in \{1, \ldots, q\},\, n_j \in D[a, j])\,.$$

While many locally checkable problems are expressible as LCVP problems, there are still some relevant problems that do not admit such a characterization, but do belong to the set of problems we analyze in this paper. Examples include $[k]-$ROMAN DOMINATION and BALANCED k-COMMUNITY, see Section 6.

In [16], Gerber and Kobler studied a variation of LCVP, with two modifications. On one hand they restrict the entries of D to sets of consecutive integers, and on the other hand, they associate to each vertex v a set $\rho(v) \subseteq \{1, \ldots, q\}$ such that $v \in V_i \Rightarrow i \in \rho(v)$. They show that the problems in this framework are XP when parameterized by clique-width. Notice that these problems are also covered by our framework.

In [10], Courcelle, Makowsky and Rotics presented an algorithm which, given as input a graph with an associated clique-width expression, solves problems expressible in a certain variation of Monadic Second-Order Logic, called MSO_1. On graphs of clique-width at most k, the running time of their algorithm is linear in the size of the input graph. However, as pointed out in [14], the multiplicative constant grows extremely fast with k.

Following a similar research line, in their recent article [5], Bergougnoux, Dreier and Jaffke defined an extension of existential MSO_1, which they call *distance neighborhood logic with acyclicity and connectivity constraints* (A&C DN logic). They provided an algorithm that solves problems expressible in this logic, given a suitable branch decomposition of the input graph. The complexity of the algorithm is expressed in terms of the *d-neighborhood* equivalence relation (see [7]), allowing them to state their main result parameterized by

mim-width (XP), tree-width, rank-width or clique-width (FPT), with a single-exponential dependence. As shown in [17], all locally checkable problems with constant number of colors and d-stable check functions, for some constant d, can be expressed in A&C DN logic. However, locally checkable problems with a color-counting check function that is not d-stable for any constant d, and possibly extended with global properties, such as BALANCED k-COMMUNITY, cannot be directly expressed by an A&C DN logic formula of fixed length.

This paper is structured as follows. In Section 2, we give some definitions and notations. In Section 3, we formally present our framework, while in Section 4, we describe the dynamic programming algorithm, prove its correctness and analyse its complexity. Section 5 deals with the extension of our results of the previous section to include the global size property. Finally, in Section 6, we apply our results to some selected problems.

2 Preliminaries

2.1 Algebraic definitions

Let $f\colon D \to C$ be a function and let $S \subseteq D$. We denote by $f|_S$ the function f restricted to the domain S, that is, the function $f|_S\colon S \to C$ is defined as $f|_S(x) = f(x)$ for all $x \in S$. Let D' be a set such that $D \cap D' = \emptyset$, and let $g\colon D' \to C$. We denote by $f \cup g$ the function $h\colon D \cup D' \to C$ such that $h(x) = f(x)$ if $x \in D$, and $h(x) = g(x)$ if $x \in D'$. Note that, since D and D' are disjoint, $f \cup g$ is well defined.

We denote by $[\![a, b]\!]$, with $a, b \in \mathbb{Z}$ and $a \leq b$, the set of all integer numbers greater than or equal to a and less than or equal to b, that is $\{a, a+1, \ldots, b\}$. Furthermore, we use BOOL to denote the set $\{\text{TRUE}, \text{FALSE}\}$.

2.2 Graph theoretical definitions

Throughout this paper, we consider simple, finite and undirected graphs. For graph theoretical notions not defined here, the reader is referred to [28].

The notion of clique-width of a graph G, denoted by $cw(G)$, was first introduced in [11]. It is defined as the minimum number of labels needed to construct G using the following 4 operations:

- creation of a new vertex v with label i (denoted by $i(v)$);
- disjoint union of two labeled graphs G_1 and G_2 (denoted by $G_1 \oplus G_2$);
- join between two labels i and j, $i \neq j$, i.e. adding an edge between every vertex with label i and every vertex with label j (denoted by $\eta_{i,j}$);
- renaming of label i to label j, i.e. every vertex with label i gets label j (denoted by $\rho_{i \to j}$).

Given a graph class $\mathcal{G}$, the clique-width of $\mathcal{G}$ is $cw(\mathcal{G}) = \sup\{cw(G) \mid G \in \mathcal{G}\}$. We say that $\mathcal{G}$ is *of bounded clique-width* if $cw(\mathcal{G}) < \infty$.

A *clique-width expression* is simply a well-formed expression of operations each corresponding to one of the four operations mentioned above. For a clique-width expression e, we denote by G_e the graph constructed by e. If the number of distinct labels used in a clique-width expression e is at most k, then we say it is a *clique-width k-expression*. It was shown in [12] that any graph G admitting a clique-width k-expression also admits an *irredundant clique-width k-expression*, i.e., such that whenever we execute a join operation $\eta_{i,j}$, there are no already existing edges between vertices with label i and vertices with label j.

Consider a clique-width expression e and the corresponding graph G_e. An *expression tree* of G_e is a rooted binary tree T_e defined as follows:

- The nodes of T_e are of four types corresponding to operations $i(\cdot)$, $\oplus$, η and ρ.
- The leaves of T_e correspond to the creation operation $i(\cdot)$.
- A disjoint union node $\oplus$ corresponds to the disjoint union of the graphs associated with its two children.
- A join node $\eta_{i,j}$ corresponds to the graph associated with its unique child in which we make all vertices of label i adjacent to all vertices of label j.
- A relabeling node $\rho_{i\to j}$ corresponds to the graph associated with its unique child in which we change label i to label j.
- The graph G_e corresponds to the graph associated with the root of T_e.

Notice that for every node $t \in V(T_e)$, the subtree of T_e rooted at t defines a clique-width expression e_t the corresponding graph of which, denoted by G_{e_t}, is a subgraph of G_e. We say that e' is a *subexpression* of e if e' is the expression determined by the subtree of T_e rooted at some node $t \in V(T_e)$. Consider any vertex v in G_{e_t} for some $t \in V(T_e)$. Then all neighbors of v in G_e which are not yet neighbors of v in G_{e_t}, i.e. the edges between v and these vertices are only defined by the ancestor operations of t in T_e, are said to be *upcoming neighbors of v with respect to e_t*. Notice that for any two vertices in G_{e_t} having the same label, their sets of upcoming neighbors with respect to e_t are identical.

Let e be a clique-width k-expression, G_e be its corresponding graph and let T_e be an expression tree of G_e. We define the function $\ell_e : V(G) \to [\![1, k]\!]$ such that $\ell_e(v)$ is the final label of v, i.e. the label of v after the operation corresponding to the root of T_e. We also define $\overline{\ell(e)}$ as the set of labels i such that there exists no $v \in V(G_e)$ such that $\ell_e(v) = i$.

In the remaining of our paper, we will only consider irredundant clique-width k-expressions where in any relabeling operation $\rho_{i\to j}(e)$ we have $j \notin \overline{\ell(e)}$. Notice that under these assumptions the total number of operations in such a clique-width expression of a graph G is in $O(|V(G)| + |E(G)|)$.

2.3 Finite-state automata

A *deterministic finite-state automaton* is a five-tuple $(Q, \Sigma, \delta, q_0, F)$ that consists of

- Q: a finite set of *states*,
- Σ: a finite set of *input symbols* (often called the *alphabet*),
- $\delta\colon Q \times \Sigma \to Q$: a *transition function*,
- $q_0 \in Q$: an *initial* (or *start*) *state*, and
- $F \subseteq Q$: a set of *final* (or *accepting*) *states*.

We say that an automaton $M = (Q, \Sigma, \delta, q_0, F)$ *accepts* a string $c_1 \dots c_n$, with $n \geq 1$, if and only if $c_i \in \Sigma$ for all $1 \leq i \leq n$ and $\delta(\dots \delta(\delta(q_0, c_1), c_2) \dots, c_n) \in F$.

For more about automata theory, we refer the reader to [19].

2.4 Weight sets

Let $(\textsc{Weights}, \preceq)$ be a totally ordered set with a maximum element (called $\textsc{Error}$), together with the minimum operation of the order $\preceq$ (called min) and a closed binary operation on $\textsc{Weights}$ (called $\circledast$) that is commutative and associative, has a neutral element and an absorbing element that is equal to $\textsc{Error}$, and the following property is satisfied: $s_1 \preceq s_2 \Rightarrow s_1 \circledast s_3 \preceq s_2 \circledast s_3$ for all $s_1, s_2, s_3 \in \textsc{Weights}$. In such a case, we say that $(\textsc{Weights}, \preceq, \circledast)$ is a *weight set*.

A classic example of a weight set is $(\mathbb{N} \cup \{+\infty\}, \leq, +)$. Notice that the maximum element is $+\infty$ in this case. We could also consider the reversed order of natural weights: $(\mathbb{N} \cup \{-\infty\}, \geq, +)$, where the maximum element is now $-\infty$. Another simple example worth mentioning is $(\{0, 1\}, \leq, \max)$.

3 Color-counting 1-locally checkable problems

Suppose we are given:

- a simple undirected graph G,
- a set $\textsc{Colors} = \{a_1, \ldots, a_q\}$,
- for every $v \in V(G)$, a nonempty set $L_v \subseteq \textsc{Colors}$ of possible colors for v,
- a weight set $(\textsc{Weights}, \preceq, \circledast)$,
- for every $v \in V(G)$ and for every $a \in L_v$, a weight $\mathrm{w}_{v,a} \in \textsc{Weights} - \{\textsc{Error}\}$ of assigning color a to vertex v, and
- a color-counting check function *check*.

We say that a coloring $c : V(G) \to \textsc{Colors}$ is *valid* if $c(v) \in L_v$ for all $v \in V(G)$. The weight of a valid coloring c is $\mathrm{w}(c) = \circledast_{v \in V} \mathrm{w}_{v,c(v)}$. Furthermore, we say that c is a *proper* coloring of G if it is a valid coloring of G and $check(v, c(v), n_1, \ldots, n_q)$ is true for every $v \in V(G)$, where $n_j = |\{u \in N_G(v) : c(u) = a_j\}|$ for all $j \in [\![1, q]\!]$.

A *color-counting 1-locally checkable problem* consists in finding the minimum weight of a proper coloring of the input graph G.

4 Algorithm

Consider a color-counting 1-locally checkable problem Π and let G be the input graph and e_G a clique-width k-expression of G. Let $\mathcal{N} \in [\![1, |V(G)|]\!]$ be an integer such that $check(v, a, n_1, \ldots, n_q) = check(v, a, \min(\mathcal{N}, n_1), \ldots, \min(\mathcal{N}, n_q))$ for all $v \in V(G)$, $a \in \textsc{Colors}$ and non-negative integers $n_1, \ldots, n_q$.

In this section, we present an algorithm which computes the minimum weight of a proper coloring of G by using the expression e_G as well as the notion of (C, N)-coloring defined hereafter.

▶ **Definition 3** ((C, N)-coloring). *Let e be a subexpression of e_G, and let C and N be two matrices in $[\![0, \mathcal{N}]\!]^{k \times q}$. A valid coloring c of G_e is called a (C, N)-coloring of G_e if the following two conditions hold:*

(C1) $\min(\mathcal{N}, |\{v \in V(G_e) : c(v) = a \wedge \ell_e(v) = i\}|) = C[i, a]$ *for all* $i \in [\![1, k]\!]$ *and all* $a \in \textsc{Colors}$;

(C2) *for every vertex* $v \in V(G_e)$ *we have* $check(v, c(v), n_1, \ldots, n_q) = \textsc{True}$, *where* $n_j = \min(\mathcal{N}, N[\ell_e(v), a_j] + |\{u \in N_{G_e}(v) : c(u) = a_j\}|)$ *for every* $j \in [\![1, q]\!]$.

The minimum weight among all possible (C, N)-colorings of G_e is denoted by $\lambda(e, C, N)$, i.e. $\lambda(e, C, N) = \min\{\mathrm{w}(c) : c \text{ is a } (C, N)\text{-coloring of } G_e\}$. Notice that if no such coloring exists then $\lambda(e, C, N) = \textsc{Error}$.

The following lemma explains the link between proper colorings and (C, N)-colorings.

▶ **Lemma 4.** *Let Π be a color-counting 1-locally checkable problem with input graph G and let e_G be a clique-width k-expression of G. Then the minimum weight of a proper coloring of G equals the minimum among all $\lambda(e_G, C, N_0)$, where $N_0 \in [\![0, \mathcal{N}]\!]^{k \times q}$ is the matrix whose elements are all 0 and $C \in [\![0, \mathcal{N}]\!]^{k \times q}$ is any matrix such that $C[i, a] = 0$ for every $i \in \overline{\ell(e_G)}$ and every $a \in \textsc{Colors}$.*

Proof. We will show that for every proper coloring c of G there exists a matrix $C \in [\![0, \mathcal{N}]\!]^{k \times q}$ such that $C[i, a] = 0$ for every $i \in \overline{\ell(e_G)}$ and every $a \in \textsc{Colors}$, and such that $\mathrm{w}(c) \geq \lambda(e_G, C, N_0)$. On the other hand, we will then show that for every matrix $C \in [\![0, \mathcal{N}]\!]^{k \times q}$ such that $C[i, a] = 0$ for every $i \in \overline{\ell(e_G)}$ and every $a \in \textsc{Colors}$, and such that $\lambda(e_G, C, N_0) \neq \textsc{Error}$, there exists a proper coloring c of G such that $\mathrm{w}(c) = \lambda(e_G, C, N_0)$.

Suppose we have a proper coloring c of G. Let $C \in [\![0, \mathcal{N}]\!]^{k \times q}$ be the matrix such that $C[i,a] = \min(\mathcal{N}, |\{v \in V(G) : c(v) = a \wedge \ell_e(v) = i\}|)$ for all $i \in [\![1,k]\!]$ and all $a \in \textsc{Colors}$. Clearly, $C[i,a] = 0$ for every $i \in \overline{\ell(e_G)}$ and every $a \in \textsc{Colors}$. Also, for every $v \in V(G)$ we have that $\mathit{check}(v, c(v), n_1, \ldots, n_q)$ is true, where $n_j = \min(\mathcal{N}, N_0[\ell_e(v), a_j] + |\{u \in N_G(v) : c(u) = a_j\}|)$ for every $j \in [\![1,q]\!]$, because $N_0[\ell_e(v), a_j] = 0$ for every $j \in [\![1,q]\!]$ and c is a proper coloring of G. Therefore, c is a (C, N_0)-coloring of G and so $\mathrm{w}(c) \geq \lambda(e_G, C, N_0)$.

Now suppose we have a matrix $C \in [\![0, \mathcal{N}]\!]^{k \times q}$ such that $C[i,a] = 0$ for every $i \in \overline{\ell(e_G)}$ and every $a \in \textsc{Colors}$, and such that $\lambda(e_G, C, N_0) \neq \textsc{Error}$. Let c be a (C, N_0)-coloring of G of minimum weight (notice that at least one such c exists, since $\lambda(e_G, C, N_0) \neq \textsc{Error}$). We will prove that c is a proper coloring of G. By definition of a (C, N_0)-coloring, c is a valid coloring, so it only remains to prove that $\mathit{check}(v, c|_{N_G[v]})$ is true for every $v \in V(G)$. We know that for every $v \in V(G)$, we have that $\mathit{check}(v, c(v), n_1, \ldots, n_q)$ is true, where $n_j = \min(\mathcal{N}, N_0[\ell_e(v), a_j] + |\{u \in N_G(v) : c(u) = a_j\}|)$ for every $j \in [\![1,q]\!]$. Since $N_0[\ell_e(v), a_j] = 0$ for every $v \in V(G)$ and $j \in [\![1,q]\!]$, we have that $\mathit{check}(v, c|_{N_G[v]}) = \mathit{check}(v, c(v), n_1, \ldots, n_q) = \textsc{True}$, where $n_j = \min(\mathcal{N}, |\{u \in N_G(v) : c(u) = a_j\}|)$ for all $j \in [\![1,q]\!]$. ◀

So Lemma 4 tells us that in order to solve a color-counting 1-locally checkable problem Π, i.e. in order to find a minimum weight of a proper coloring, it is sufficient to find the minimum weight among all (C, N_0)-colorings of the input graph G, where C and N_0 are as described above. Our algorithm is based exactly on this idea, i.e. it determines the minimum among all $\lambda(e_G, C, N_0)$. This is achieved by traversing the binary rooted tree T_{e_G} in a bottom-up fashion and determining in a recursive way the values $\lambda(e, C, N)$, where e is a subexpression of e_G and $C, N \in [\![0, \mathcal{N}]\!]^{k \times q}$. Throughout this recursion, the matrices C and N will intuitively behave in the following way: if we have a proper coloring c of G such that $c|_{V(G_e)}$ is a (C, N)-coloring of G_e, then

- $C[i,a]$ represents the minimum between $\mathcal{N}$ and the number of vertices v in G_e such that $\ell_e(v) = i$ and $c(v) = a$, and
- $N[i,a]$ represents the minimum between $\mathcal{N}$ and the number of vertices $u \in V(G)$ with $c(u) = a$ that are upcoming neighbors with respect to e of every vertex v with $\ell_e(v) = i$.

For the next four lemmas, we will assume that we are given matrices C and N in $[\![0, \mathcal{N}]\!]^{k \times q}$. We will describe the recursive computation of $\lambda(e, C, N)$ by distinguishing four cases depending on the kind of clique-width operation at the root of the tree T_e.

▶ **Lemma 5** (Creating new vertex: $i(v)$)**.** *If there exists $a \in L_v$ such that $C[i,a] = 1$ and $C[j,b] = 0$ for all the other entries $[j,b]$ in C, and if $\mathit{check}(v, a, N[i,a_1], \ldots, N[i,a_q])$ is true, then $\lambda(i(v), C, N) = \mathrm{w}_{v,a}$. Otherwise, $\lambda(i(v), C, N) = \textsc{Error}$.*

Proof. First notice that $G_{i(v)}$ is the graph consisting of a single vertex v with label i. Therefore, if C and N have the above properties (i.e. there exists $a \in L_v$ such that $C[i,a] = 1$ and $C[j,b] = 0$ for all the other entries $[j,b]$ in C, and $\mathit{check}(v, a, N[i,a_1], \ldots, N[i,a_q])$ is true), there is exactly one (C, N)-coloring c of $G_{i(v)}$, defined by $c(v) = a$. Indeed, since $a \in L_v$, it follows that c is a valid coloring. Moreover, conditions (C1) and (C2) are trivially satisfied. Then, $\lambda(i(v), C, N) = \mathrm{w}(c) = \mathrm{w}_{v,a}$.

On the other hand, if C does not have exactly one nonzero entry, or if it is not in row i, or if it is in a column $a \notin L_v$, or if this entry is not equal to 1, then no valid coloring satisfying condition (C1) exists. If for this unique possible choice of color a, $\mathit{check}(v, a, N[i,a_1], \ldots, N[i,a_q])$ is false, then no (C, N)-coloring of $G_{i(v)}$ exists either. Therefore $\lambda(i(v), C, N) = \textsc{Error}$. ◀

▶ **Lemma 6** (Disjoint union: $e_1 \oplus e_2$). *Let N_1 and N_2 be two matrices in $[\![0,\mathcal{N}]\!]^{k\times q}$ such that:*

- *$N_1[i,a] = 0$ for every label $i \in \overline{\ell(e_1)}$ and every color $a \in$ Colors;*
- *$N_1[i,a] = N[i,a]$ for every label $i \notin \overline{\ell(e_1)}$ and every color $a \in$ Colors; and*
- *N_2 is defined analogously with respect to e_2.*

Then

$$\begin{aligned}\lambda(e_1 \oplus e_2, C, N) = \min\{&\lambda(e_1, C_1, N_1) \circledast \lambda(e_2, C_2, N_2) :\\ &\text{(a) } C_1, C_2 \in [\![0,\mathcal{N}]\!]^{k\times q}\\ &\text{(b) } C_1[i,a] = 0 \text{ for all } i \in \overline{\ell(e_1)}, a \in \textsc{Colors};\\ &\text{(c) } C_2[i,a] = 0 \text{ for all } i \in \overline{\ell(e_2)}, a \in \textsc{Colors};\\ &\text{(d) } C[i,a] = \min(\mathcal{N}, C_1[i,a] + C_2[i,a]) \text{ for all } i \in [\![1,k]\!], a \in \textsc{Colors}\}.\end{aligned}$$

Proof. Let $\alpha = \min\{\lambda(e_1, C_1, N_1) \circledast \lambda(e_2, C_2, N_2) : (a), (b), (c), (d) \text{ are satisfied}\}$. We will first prove that $\lambda(e_1 \oplus e_2, C, N) \geq \alpha$. If $\lambda(e_1 \oplus e_2, C, N) = \textsc{Error}$, then we are done. So assume that $\lambda(e_1 \oplus e_2, C, N) \neq \textsc{Error}$ and let c be a (C,N)-coloring of $G_{e_1\oplus e_2}$ whose weight equals $\lambda(e_1 \oplus e_2, C, N)$. We need to show that there exist C_1 and C_2 in $[\![0,\mathcal{N}]\!]^{k\times q}$ satisfying $(b), (c), (d)$ and such that the weight of c is at least $\lambda(e_1, C_1, N_1) \circledast \lambda(e_2, C_2, N_2)$. Let $c_1 = c|_{V(G_{e_1})}$ and $c_2 = c|_{V(G_{e_2})}$. Then, we define $C_1[i,a] = \min(\mathcal{N}, |\{v \in V(G_{e_1}) : c_1(v) = a \wedge \ell_{e_1}(v) = i\}|)$ for any label $i \in [\![1,k]\!]$ and color $a \in \textsc{Colors}$, and similarly for C_2 with respect to e_2. Consequently, conditions (b) and (c) are satisfied: if $i \in \overline{\ell(e_1)}$, then $\{v \in V(G_{e_1}) : \ell_{e_1}(v) = i\} = \emptyset$, thus $C_1[i,a] = 0$, and similarly for C_2. Condition (d) is also satisfied because c is a (C,N)-coloring of $G_{e_1\oplus e_2}$, $V(G_{e_1})$ and $V(G_{e_2})$ are disjoint, $\ell_{e_1}(v) = \ell_{e_1\oplus e_2}(v)$ for every $v \in V(G_{e_1})$ and $\ell_{e_2}(v) = \ell_{e_1\oplus e_2}(v)$ for every $v \in V(G_{e_2})$. We now show that c_1 is a (C_1, N_1)-coloring of G_{e_1}. Condition (C1) is trivially satisfied by the definition of C_1. To show that condition (C2) is satisfied, we will show that $\mathit{check}(v, c_1(v), n'_1, \ldots, n'_q)$ is true for every vertex $v \in V(G_{e_1})$, where $n'_j = \min(\mathcal{N}, N_1[\ell_{e_1}(v), a_j] + |\{u \in N_{G_{e_1}}(v) : c_1(u) = a_j\}|)$ for every $j \in [\![1,q]\!]$. Since c is a (C,N)-coloring of $G_{e_1\oplus e_2}$, we have that $\mathit{check}(v, c(v), n_1, \ldots, n_q)$ is true for every $v \in V(G_{e_1\oplus e_2})$, where $n_j = \min(\mathcal{N}, N[\ell_{e_1\oplus e_2}(v), a_j] + |\{u \in N_{G_{e_1\oplus e_2}}(v) : c(u) = a_j\}|)$ for every $j \in [\![1,q]\!]$. By definition of N_1 and since $\ell_{e_1}(v) = \ell_{e_1\oplus e_2}(v)$ for every $v \in V(G_{e_1})$, we have $N_1[\ell_{e_1}(v), a_j] = N[\ell_{e_1\oplus e_2}(v), a_j]$ for every $v \in V(G_{e_1})$ and every $j \in [\![1,q]\!]$. Also, $N_{G_{e_1}}(v) = N_{G_{e_1\oplus e_2}}(v)$ for every $v \in V(G_{e_1})$ and $c_1 = c|_{V(G_{e_1})}$, so $|\{u \in N_{G_{e_1}}(v) : c_1(u) = a_j\}| = |\{u \in N_{G_{e_1\oplus e_2}}(v) : c(u) = a_j\}|$ for every $v \in V(G_{e_1})$ and every $j \in [\![1,q]\!]$. Therefore $n'_j = n_j$ for every $j \in [\![1,q]\!]$ and hence, $\mathit{check}(v, c_1(v), n'_1, \ldots, n'_q)$ is true for every $v \in V(G_{e_1})$. Using similar arguments, we can show that c_2 is a (C_2, N_2)-coloring of G_{e_2}. Moreover, $\mathrm{w}(c) = \mathrm{w}(c_1) \circledast \mathrm{w}(c_2)$ by definition of c_1 and c_2, and consequently, $\lambda(e_1 \oplus e_2, C, N) = \mathrm{w}(c) = \mathrm{w}(c_1) \circledast \mathrm{w}(c_2) \geq \lambda(e_1, C_1, N_1) \circledast \lambda(e_2, C_2, N_2) \geq \alpha$.

Let us show now that $\lambda(e_1 \oplus e_2, C, N) \leq \alpha$. Consider two matrices C_1 and C_2 in $[\![0,\mathcal{N}]\!]^{k\times q}$ satisfying $(b), (c), (d)$. If $\lambda(e_1, C_1, N_1) = \textsc{Error}$ or $\lambda(e_2, C_2, N_2) = \textsc{Error}$, then we are done. Otherwise, we are going to construct a (C,N)-coloring c of $G_{e_1\oplus e_2}$ such that $\mathrm{w}(c) = \lambda(e_1, C_1, N_1) \circledast \lambda(e_2, C_2, N_2)$. Let c_1 be a (C_1, N_1)-coloring of G_{e_1} whose weight is $\lambda(e_1, C_1, N_1)$ and c_2 be a (C_2, N_2)-coloring of G_{e_2} whose weight is $\lambda(e_2, C_2, N_2)$. Let $c = c_1 \cup c_2$ (note that c is well defined because $V(G_{e_1})$ and $V(G_{e_2})$ are disjoint sets). Clearly, c is a valid coloring and $\mathrm{w}(c) = \mathrm{w}(c_1) \circledast \mathrm{w}(c_2)$. We now show that c is a (C,N)-coloring of $G_{e_1\oplus e_2}$. We start with condition (C1). Consider $i \in [\![1,k]\!]$ and $a \in \textsc{Colors}$. Since c_1 is a (C_1, N_1)-coloring of G_{e_1} and c_2 is a (C_2, N_2)-coloring of G_{e_2}, we have $C_1[i,a] = \min(\mathcal{N}, |\{v \in V(G_{e_1}) : c_1(v) = a \wedge \ell_{e_1}(v) = i\}|)$ and $C_2[i,a] = \min(\mathcal{N}, |\{v \in V(G_{e_2}) : c_2(v) = a \wedge \ell_{e_2}(v) = i\}|)$.

Then,

$$\begin{aligned}
C[i,a] &= \min(\mathcal{N}, C_1[i,a] + C_2[i,a]) \\
&= \min(\mathcal{N}, \min(\mathcal{N}, |\{v \in V(G_{e_1}) : c_1(v) = a \wedge \ell_{e_1}(v) = i\}|) + \\
&\quad \min(\mathcal{N}, |\{v \in V(G_{e_2}) : c_2(v) = a \wedge \ell_{e_2}(v) = i\}|)) \\
&= \min(\mathcal{N}, |\{v \in V(G_{e_1}) : c_1(v) = a \wedge \ell_{e_1}(v) = i\}| + \\
&\quad |\{v \in V(G_{e_2}) : c_2(v) = a \wedge \ell_{e_2}(v) = i\}|) \\
&= \min(\mathcal{N}, |\{v \in V(G_{e_1 \oplus e_2}) : c(v) = a \wedge \ell_{e_1 \oplus e_2}(v) = i\}|).
\end{aligned}$$

The last equality is simply due to the definition of c and to the facts that $V(G_{e_1 \oplus e_2}) = V(G_{e_1}) \cup V(G_{e_2})$ and for every $v \in G_{e_1}$ we have that $\ell_{e_1}(v) = \ell_{e_1 \oplus e_2}(v)$ and for every $v \in G_{e_2}$ we have that $\ell_{e_2}(v) = \ell_{e_1 \oplus e_2}(v)$. Let us focus on (C2) now. Consider $v \in G_{e_1 \oplus e_2}$. Assume, without loss of generality, that $v \in V(G_{e_1})$. We will prove that $check(v, c(v), n_1, \ldots, n_q)$ is true, where $n_j = \min(\mathcal{N}, N[\ell_{e_1 \oplus e_2}(v), a_j] + |\{u \in N_{G_{e_1 \oplus e_2}}(v) : c(u) = a_j\}|)$ for every $j \in [\![1, q]\!]$. Since c_1 is a (C_1, N_1)-coloring of G_{e_1}, we know that $check(v, c_1(v), n'_1, \ldots, n'_q)$ is true, where $n'_j = \min(\mathcal{N}, N_1[\ell_{e_1}(v), a_j] + |\{u \in N_{G_{e_1}}(v) : c_1(u) = a_j\}|)$ for every $j \in [\![1, q]\!]$. Using similar arguments as above, we obtain again that $n'_j = n_j$ for every $j \in [\![1, q]\!]$. Due to the definition of c, we then conclude that $check(v, c(v), n_1, \ldots, n_q)$ is true for every $v \in V(G)$, and so (C2) is satisfied. Thus, c is a (C, N)-coloring of $G_{e_1 \oplus e_2}$. ◀

▶ **Lemma 7** (Join: $\eta_{i,j}(e)$)**.** *Let $N_e \in [\![0, \mathcal{N}]\!]^{k \times q}$ be such that*

- $N_e[i,a] = \min(\mathcal{N}, N[i,a] + C[j,a])$ *for every* $a \in$ Colors*;*
- $N_e[j,a] = \min(\mathcal{N}, N[j,a] + C[i,a])$ *for every* $a \in$ Colors*;*
- $N_e[h,a] = N[h,a]$ *for every* $h \in [\![1,k]\!] \setminus \{i,j\}$ *and every* $a \in$ Colors*.*

Then, $\lambda(\eta_{i,j}(e), C, N) = \lambda(e, C, N_e)$.

Proof. First of all, since the labelings of the vertices do not change between $G_{\eta_{i,j}(e)}$ and G_e, we simply use ℓ to denote both labelings ℓ_e and $\ell_{\eta_{i,j}(e)}$. Also, since $V(G_{\eta_{i,j}(e)}) = V(G_e)$, we simply denote this set by V.

Let $c\colon V \to$ Colors be a valid coloring. We are going to show that c is a (C, N)-coloring of $G_{\eta_{i,j}(e)}$ if and only if c is a (C, N_e)-coloring of G_e. In order to prove this, it suffices to show that, for every color $a \in$ Colors and every vertex $v \in V$, we have $\min(\mathcal{N}, N[\ell(v), a] + |\{u \in N_{G_{\eta_{i,j}(e)}}(v) : c(u) = a\}|) = \min(\mathcal{N}, N_e[\ell(v), a] + |\{u \in N_{G_e}(v) : c(u) = a\}|)$. If $\ell(v) \neq i, j$ then $N_e[\ell(v), a_i] = N[\ell(v), a_i]$ by definition of N_e and clearly $N_{G_e}(v) = N_{G_{\eta_{i,j}(e)}}(v)$. On the other hand, if $\ell(v) = i$ (if $\ell(v) = j$ the proof is analogous) then

$$\begin{aligned}
&\min(\mathcal{N}, N_e[i,a] + |\{u \in N_{G_e}(v) : c(u) = a\}|) \\
&= \min(\mathcal{N}, \min(\mathcal{N}, N[i,a] + C[j,a]) + |\{u \in N_{G_e}(v) : c(u) = a\}|) \\
&= \min(\mathcal{N}, N[i,a] + C[j,a] + |\{u \in N_{G_e}(v) : c(u) = a\}|) \\
&= \min(\mathcal{N}, N[i,a] + \min(\mathcal{N}, |\{u \in V : c(u) = a \wedge \ell(u) = j\}|) + |\{u \in N_{G_e}(v) : c(u) = a\}|) \\
&= \min(\mathcal{N}, N[i,a] + |\{u \in V : c(u) = a \wedge \ell(u) = j\}| + |\{u \in N_{G_e}(v) : c(u) = a\}|) \\
&= \min(\mathcal{N}, N[i,a] + |\{u \in N_{G_{\eta_{i,j}(e)}}(v) : c(u) = a \wedge \ell(u) = j\}| + |\{u \in N_{G_e}(v) : c(u) = a\}|) \\
&= \min(\mathcal{N}, N[i,a] + |\{u \in N_{G_{\eta_{i,j}(e)}}(v) : c(u) = a\}|).
\end{aligned}$$

The last two equalities follow from the fact that every vertex with label j is a neighbor of v in $G_{\eta_{i,j}(e)}$ but a non-neighbor of v in G_e (recall that we are working with irredundant clique-width expressions). ◀

▶ **Lemma 8** (Relabeling: $\rho_{i\to j}(e)$)**.** *Let N_e be such that*

- $N_e[i,a] = N[j,a]$ *for every* $a \in$ Colors*;*
- $N_e[h,a] = N[h,a]$ *for every* $h \in [\![1,k]\!] \setminus \{i\}$ *and every* $a \in$ Colors*.*

If $C[i,a] = 0$ for all $a \in$ Colors, then

$$\begin{aligned}\lambda(\rho_{i\to j}(e), C, N) = \min\{\lambda(e, C_e, N_e) : \\ &\text{(a) } C_e \in [\![0,\mathcal{N}]\!]^{k\times q} \\ &\text{(b) } C[j,a] = \min(\mathcal{N}, C_e[i,a] + C_e[j,a]) \text{ for all } a \in \text{Colors}; \\ &\text{(c) } C_e[h,a] = C[h,a] \text{ for all } h \in [\![1,k]\!] \setminus \{i,j\}, a \in \text{Colors}\}.\end{aligned}$$

Otherwise, $\lambda(\rho_{i\to j}(e), C, N) =$ Error.

Proof. If $C[i,a] \neq 0$ for some $a \in$ Colors, then, by definition, there exists no (C,N)-coloring of $G_{\rho_{i\to j}(e)}$ and $\lambda(\rho_{i\to j}(e), C, N) =$ Error. So we may assume now that $C[i,a] = 0$ for all $a \in$ Colors. Let $\alpha = \min\{\lambda(e, C_e, N_e) : (a), (b), (c) \text{ are satisfied}\}$. We will first prove that $\lambda(\rho_{i\to j}(e), C, N) \geq \alpha$. Let c be a (C,N)-coloring of $G_{\rho_{i\to j}(e)}$ whose weight equals $\lambda(\rho_{i\to j}(e), C, N)$. We will show that there exists a matrix C_e in $[\![0,\mathcal{N}]\!]^{k\times q}$ satisfying $(b), (c)$ and such that c is a (C_e, N_e)-coloring of G_e. Let

$$C_e[h,a] = \min(\mathcal{N}, |\{v \in V(G_e) : c(v) = a \wedge \ell_e(v) = h\}|)$$

for every $h \in [\![1,k]\!]$ and every $a \in$ Colors. Since c is a (C,N)-coloring of $G_{\rho_{i\to j}(e)}$, then $C[h,a] = \min(\mathcal{N}, |\{v \in V(G_{\rho_{i\to j}(e)}) : c(v) = a \wedge \ell_{\rho_{i\to j}(e)}(v) = h\})$ for every $h \in [\![1,k]\!]$ and $a \in$ Colors. Therefore, (c) is trivially satisfied, since $\ell_{\rho_{i\to j}(e)}(v) = \ell_e(v)$ for any vertex v whose label is neither i nor j in G_e. Furthermore the following inequalities hold,

$$\begin{aligned}\min(\mathcal{N}, C_e[i,a] + C_e[j,a]) =& \min(\mathcal{N}, \min(\mathcal{N}, |\{v \in V(G_e) : c(v) = a \wedge \ell_e(v) = i\}|) \\ &+ \min(\mathcal{N}, |\{v \in V(G_e) : c(v) = a \wedge \ell_e(v) = j\}|)) \\ =& \min(\mathcal{N}, |\{v \in V(G_e) : c(v) = a \wedge \ell_e(v) = i\}| \\ &+ |\{v \in V(G_e) : c(v) = a \wedge \ell_e(v) = j\}|) \\ =& \min(\mathcal{N}, |\{v \in V(G_e) : c(v) = a \wedge (\ell_e(v) = i \vee \ell_e(v) = j)\}|) \\ =& \min(\mathcal{N}, |\{v \in V(G_{\rho_{i\to j}(e)}) : c(v) = a \wedge \ell_{\rho_{i\to j}(e)}(v) = j\}|) \\ =& C[j,a]\end{aligned}$$

and thus, condition (b) is satisfied as well. We next show that c is a (C_e, N_e)-coloring of G_e. We know that c is a valid coloring because c is a (C,N)-coloring of $G_{\rho_{i\to j}(e)}$. Also, (C1) is trivially satisfied from our definition of C_e. For (C2), we have to show that $check(v, c(v), n'_1, \ldots, n'_q)$ is true for every v in G_e, where $n'_b = \min(\mathcal{N}, N_e[\ell_e(v), a_b] + |\{u \in N_{G_e}(v) : c(u) = a_b\}|)$ for every $v \in V(G_e)$ and every $b \in [\![1,q]\!]$. Since c is a (C,N)-coloring of $G_{\rho_{i\to j}(e)}$, we know that $check(v, c(v), n_1, \ldots, n_q)$ is true for every $v \in V(G_{\rho_{i\to j}(e)})$, where $n_b = \min(\mathcal{N}, N[\ell_{\rho_{i\to j}(e)}(v), a_b] + |\{u \in N_{G_{\rho_{i\to j}(e)}}(v) : c(u) = a_b\}|)$ for every $b \in [\![1,q]\!]$. By definition of N_e and since $N_{G_e}(v) = N_{G_{\rho_{i\to j}(e)}}(v)$ for all vertices v, we have $n_b = n'_b$ for all $b \in [\![1,q]\!]$, and therefore $check(v, c(v), n'_1, \ldots, n'_q)$ is true for every $v \in G_e$ and $b \in [\![1,q]\!]$.

We now show that $\lambda(\rho_{i\to j}(e), C, N) \leq \alpha$. Let C_e be a matrix in $[\![0,\mathcal{N}]\!]^{k\times q}$ satisfying $(b), (c)$, and let c be a (C_e, N_e)-coloring of G_e of weight $\lambda(e, C_e, N_e)$. We are going to show that c is also a (C,N)-coloring of $G_{\rho_{i\to j}(e)}$. Condition (C1) is trivially satisfied for every label h in $[\![1,k]\!] \setminus \{i,j\}$ and every color a. For label i, condition (C1) is also true since $C[i,a] = 0$ by assumption and there are no vertices with label i in $V(G_{\rho_{i\to j}(e)})$. We can show in a

Algorithm 1 Main algorithm.

```
1  for every subexpression e of e_G, traversing them in a bottom-up fashion, do
2  |  forall matrices C, N ∈ [[0, N]]^{k×q} do
3  |  |  Compute λ(e, C, N) using Lemmas 5, 6, 7 or 8, according to the type of the
   |  |   operation at the root of T_e.
4  |  |  Store the result in memory for future uses.
5  |  end
6  end
7  Let N_0 be the matrix in [[0, N]]^{k×q} such that all its elements are 0.
8  Let m ← Error
9  forall C ∈ [[0, N]]^{k×q} such that C[i, a] = 0 for every i ∈ ℓ(e_G), a ∈ Colors do
10 |  m ← min(m, λ(e_G, C, N_0))
11 end
12 return m
```

similar way as above that $C[j,a] = \min(\mathcal{N}, |\{v \in V(G_{\rho_{i\to j}(e)}) : c(v) = a \wedge \ell_{\rho_{i\to j}(e)}(v) = j\}|)$ for all $a \in$ Colors. We need to verify now (C2), i.e., for every vertex v, we have that $check(v, c(v), n_1, \ldots, n_q)$ is true, where $n_b = \min(\mathcal{N}, N[\ell_{\rho_{i\to j}(e)}(v), a_b] + |\{u \in N_{G_{\rho_{i\to j}(e)}}(v) : c(u) = a_b\}|)$. As before, this is a consequence of the fact that c is a (C_e, N_e)-coloring of G_e, and that for every vertex v and color a_b, we have $N_e[\ell_e(v), a_b] = N[\ell_{\rho_{i\to j}(e)}(v), a_b]$ by definition and $|\{u \in N_{G_e}(v) : c(u) = a_b\}|) = |\{u \in N_{G_{\rho_{i\to j}(e)}}(v) : c(u) = a_b\}|$. ◀

Our algorithm, which takes the same input as a locally checkable problem, plus the number $\mathcal{N}$ and an irredundant clique-width k-expression e_G of the input graph G together with its binary rooted tree T_{e_G}, and outputs the minimum weight of a proper coloring of G, is presented in Algorithm 1. As explained above, we proceed in a bottom-up fashion, i.e. we start with the leaf nodes of T_{e_G}, then continue with their parents and so on, and compute each time $\lambda(e, C, N)$ for the corresponding subexpression e (i.e. for the subexpression e corresponding to the node of T_{e_G} that we are currently analyzing) and all possible choices of C and N using the recurrences in Lemmas 5, 6, 7 and 8 (see lines 1-3). Since we are storing the results (see line 4), the number of times we need to compute some value $\lambda(\cdot,\cdot,\cdot)$ is given by the number of subexpressions of e_G times the possible choices for the matrices C and N. Since we have $O(|V(G)| + |E(G)|)$ subexpressions in the given clique-width expression (see Section 2), and since there exist $(\mathcal{N}+1)^{kq}$ possible matrices C, respectively possible matrices N, we obtain that line 3 of our algorithm is called at most $O((|V(G)| + |E(G)|)(\mathcal{N}+1)^{2kq})$ times. In lines 7-11, we then determine the minimum among all $\lambda(e_G, C, N_0)$, where N_0 is the matrix whose elements are all 0, and $C \in [\![0, \mathcal{N}]\!]^{k\times q}$ is any matrix such that $C[i,a] = 0$ for every $i \in \overline{\ell(e_G)}$ and every $a \in$ Colors. This can be done in time $O((\mathcal{N}+1)^{kq})$.

It remains to determine the complexity of computing some value $\lambda(\cdot,\cdot,\cdot)$. This clearly depends on the operation we consider. Thus, we distinguish 4 cases:

- **Creating new vertex:** We need to go through the entries of C, which can be done in time $O(kq)$. Let us denote by $t_{check}(|V(G)|, q, \mathcal{N})$ the complexity of evaluating the check function. Hence, we obtain a complexity of $O(kq + t_{check}(|V(G)|, q, \mathcal{N}))$ for this operation.
- **Disjoint union:** We first need to determine N_1 and N_2, which takes $O(kq)$ time, and then we need to find the minimum weight by going through all possible choices of C_1 and C_2, which can be done in time $O((\mathcal{N}+1)^{2kq})$. This gives us an overall complexity of $O((\mathcal{N}+1)^{2kq})$ for determining $\lambda(\cdot,\cdot,\cdot)$ for the disjoint union operation.

- **Join:** We simply need to determine the matrix N_e, which can be done in $O(kq)$ time.
- **Relabeling:** First, we need to determine the matrix N_e, which takes $O(kq)$ time, and then we need to find the minimum weight by considering possible choices of C_e with all rows fixed except two, which clearly takes $O((\mathcal{N}+1)^{2q})$. Thus, overall the complexity of determining $\lambda(\cdot,\cdot,\cdot)$ for the relabeling operation is $O(kq+(\mathcal{N}+1)^{2q})$.

Now the complexity of computing any $\lambda(e,C,N)$ is bounded by the sum of the complexities of the four cases, for which we obtain $O(t_{check}(|V(G)|,q,\mathcal{N})+(\mathcal{N}+1)^{2kq})$. Thus, we obtain the following complexity:

$$O((|V(G)|+|E(G)|)(\mathcal{N}+1)^{2kq}(t_{check}(|V(G)|,q,\mathcal{N})+(\mathcal{N}+1)^{2kq})).$$

▶ Remark 9. We can modify the algorithm in order to also obtain the coloring function as an output. This does not affect the complexity.

Let us highlight the following main consequences of the previous analysis. Notice that, by the results in [23], we do not need a clique-width expression as input.

▶ **Corollary 10.** *Consider a color-counting 1-locally checkable problem* Π *with constant number of colors and a check function computable in polynomial time. Then* Π *is XP parameterized by clique-width.*

▶ **Corollary 11.** *Let* $d\in\mathbb{N}$. *If* Π *is a* d*-stable 1-locally checkable problem where the number of colors is* $O(\log|V(G)|)$ *and the check function can be computed in polynomial time, then* Π *is XP parameterized by clique-width.*

▶ **Corollary 12.** *Let* $d\in\mathbb{N}$. *If* Π *is a* d*-stable 1-locally checkable problem where the number of colors is constant and the check function can be computed in constant time, then* Π *is FPT parameterized by clique-width. Moreover, if an irredundant clique-width* k*-expression is given as input, then it is linear FPT parameterized by* k.

Notice that various well-known graph theoretical problems, such as k-Coloring, Maximum Independent Set, as well as $[k]-$Roman domination (see Section 6), are indeed d-stable 1-locally checkable problems, for some constant d, with constant number of colors.

We would also like to mention that Lemmas 4, 5, 6, 7 and 8 still hold if in this section we replace every occurrence of "$min(\mathcal{N},\cdot)$" by "$\cdot \bmod (\mathcal{N}+1)$". Further, the complexity of Algorithm 1 remains the same. By doing so, we obtain FPT algorithms parameterized by clique-width for problems such as parity domination [15, 18] and finding the minimum size of a non-$z(\bmod k)$ dominating set [8]. The next corollary formally states this observation.

▶ **Corollary 13.** *Let* $m\in\mathbb{N}$. *Consider a color-counting 1-locally checkable problem* Π *where the number of colors is constant, and the check function can be computed in constant time and is such that* $check(v,a,n_1,\ldots,n_q)=check(v,a,n_1 \bmod m,\ldots,n_q \bmod m)$ *for all* $v\in V(G)$, $a\in$ Colors *and non-negative integers* $n_1,\ldots,n_q$. *Then* Π *is FPT parameterized by clique-width. Moreover, if an irredundant clique-width* k*-expression is given as input, then it is linear FPT parameterized by* k.

5 Global size property

In this section, we extend the results of Section 3 by considering color-counting 1-locally checkable problems in which it is also required that the number of vertices that receive a given color $a\in$ Colors belongs to a predefined set σ_a of non-negative integers.

Let $(Q, \{1\}, \delta, q_0, F)$ be a deterministic finite-state automaton which accepts a string of t consecutive 1's if and only if $t \in \sigma_a$. Note that for all finite sets of non-negative integers, there exists such an automaton (for example, let m be the maximum element of the set, then we set $Q = \{s_0, \ldots, s_{m+1}\}$, $q_0 = s_0$, $F = \{s_i : i \in \sigma\}$, $\delta(s_i, 1) = s_{i+1}$ for all $0 \leq i \leq m$ and $\delta(s_{m+1}, 1) = s_{m+1}$). Let us define the notation $\delta^0(s_i) = s_i$ and $\delta^n(s_i) = \delta(\delta^{n-1}(s_i), 1)$ for every state $s_i \in Q$ and positive integer n.

We will now proceed in a similar way as in Section 4 but considering additional parameters. Let us first introduce the relevant notion of $(C, N, p_1, \ldots, p_m)$-*colorings*, which will be defined recursively. This notion can be used to extend the results of the aforementioned section using different global properties. Intuitively, if $C, N \in [\![0, \mathcal{N}]\!]^{k \times q}$ and $p_1, \ldots, p_m$ are parameters such that $(C, N, p_1, \ldots, p_m)$-colorings of G_e are defined, then, for additional parameters $p_{m+1}, \ldots, p_{m+m'}$, we define a $(C, N, p_1, \ldots, p_m, p_{m+1}, \ldots, p_{m+m'})$-*coloring of* G_e as a $(C, N, p_1, \ldots, p_m)$-coloring of G_e such that parameters $p_{m+1}, \ldots, p_{m+m'}$ satisfy some predefined property. In the case of the particular global property mentioned at the beginning of this section, we will only consider two additional parameters. The first such parameter is a state $s_a \in Q$ and the second parameter is a function $f_a \colon Q \to$ Bool. We then define a $(C, N, p_1, \ldots, p_m, s_a, f_a)$-*coloring* c of G_e as a $(C, N, p_1, \ldots, p_m)$-coloring of G_e such that $f_a(\delta^n(s_a)) =$ True, where $n = |\{v \in V(G_e) : c(v) = a\}|$. Also, in the same spirit as before, we will denote by $\lambda(e, C, N, p_1, \ldots, p_m, s_a, f_a)$ the minimum weight among all $(C, N, p_1, \ldots, p_m, s_a, f_a)$-colorings of G_e. If we want to fix the size of $\mathcal{R}$ color classes, say $a_1, \ldots, a_{\mathcal{R}}$, it suffices to associate an automaton M_i and the corresponding parameters s_{a_i} and f_{a_i} with each color class a_i, for $i \in [\![1, \mathcal{R}]\!]$.

By providing a lemma explaining how to solve a color-counting 1-locally checkable problem with given global properties by using $(C, N, p_1, \ldots, p_m, s_a, f_a)$-colorings, and then again distinguishing the four clique-width operations, as in the previous section, we can prove that, when the number of colors is constant, this new algorithm is also XP parameterized by clique-width. Due to space restrictions, their statements are omitted here, but presented in Appendix A.

6 Applications

In this section, we provide some examples of problems whose complexity status in graphs of bounded clique-width was unknown, and for each of which the application of our framework yields a first polynomial-time algorithm in this class of graphs.

6.1 (Global) $[k]$-Roman domination

The $[k]$-Roman domination problem was first defined in [1] as a generalization of Roman and double Roman domination [9, 4]. Let $k \geq 1$ be an integer. A $[k]$-*Roman dominating function* on a graph G is a function $f \colon V(G) \to [\![0, k+1]\!]$ having the property that if $f(v) < k$ then $\sum_{u \in N_G[v]} f(u) \geq |AN_G^f(v)| + k$, where $AN_G^f(v) = \{u \in N_G(v) : f(u) \geq 1\}$ (this set is called the *active neighborhood of* v). The *weight* of a $[k]$-Roman dominating function f is $\sum_{v \in V(G)} f(v)$, and the minimum weight of a $[k]$-Roman dominating function on G is the $[k]$-*Roman domination number* of G, denoted by $\gamma_{[kR]}(G)$. The problem consists in computing the $[k]$-Roman domination number of a given graph.

In [6], this problem was shown to be solvable in linear time in graphs of bounded treewidth. In their model, the number of colors is a constant and the check function is actually $(k+1)$-stable. We can express it in the following way:

- $\textsc{Colors} = [\![0, k+1]\!]$ and $L_v = [\![0, k+1]\!]$ for all $v \in V(G)$;
- $(\textsc{Weights}, \preceq, \circledast) = (\mathbb{N} \cup \{+\infty\}, \leq, +)$ and $\mathrm{w}_{v,a} = a$ for all $v \in V(G), a \in L_v$;
- $check(v, a, n_0, \ldots, n_{k+1}) = \left(a + \sum_{j=0}^{k+1} j n_j \geq k + \sum_{j=1}^{k+1} n_j\right)$.

Then, by Corollary 12, this problem is FPT parameterized by clique-width (and linear FPT when a suitable clique-width expression is given).

In [25], the authors introduced a variant of this problem, called GLOBAL ROMAN DOMINATION. This problem was later extended to GLOBAL DOUBLE ROMAN DOMINATION [26] and GLOBAL TRIPLE ROMAN DOMINATION [21]. The definition of these problems can be naturally generalized as follows. A *global* $[k]$*-Roman dominating function* on a graph G is a $[k]$-Roman dominating function in both G and $\overline{G}$. The GLOBAL $[k]$-ROMAN DOMINATION problem consists in computing the minimum weight of a global $[k]$-Roman dominating function of a graph.

In order to show that this problem is XP parameterized by clique-width, we first define an auxiliary problem.

SPECIFIED SIZE GLOBAL $[k]$-ROMAN DOMINATION
Instance: A graph G and $k+2$ non-negative integers $s_0, \ldots, s_{k+1}$ such that $\sum_{i=0}^{k+1} s_i = |V(G)|$.
Question: Does G admit a global $[k]$-Roman dominating function f such that, for all $i \in [\![0, k+1]\!]$, s_i equals the number of vertices $v \in V(G)$ with $f(v) = i$?

This last problem can be modeled as a color-counting 1-locally checkable problem with global properties:

- $\textsc{Colors} = [\![0, k+1]\!]$ and $L_v = [\![0, k+1]\!]$ for all $v \in V(G)$;
- $(\textsc{Weights}, \preceq, \circledast) = (\mathbb{N} \cup \{+\infty\}, \leq, +)$ and $\mathrm{w}_{v,a} = a$ for all $v \in V(G), a \in L_v$;
- $check(v, a, n_0, \ldots, n_{k+1}) = (a + \sum_{j=1}^{k+1}(j-1)n_j \geq k) \wedge (\sum_{j=1}^{k+1}(j-1)(s_j - n_j) \geq k)$;
- for all $a \in \textsc{Colors}$, we ask for the size of the color class of a to belong to $\{s_a\}$.

Finally, to solve GLOBAL $[k]$-ROMAN DOMINATION on graphs of bounded clique-width, we successively iterate over the feasible combinations of values $s_0, \ldots, s_{k+1}$ such that $\sum_{i=0}^{k+1} s_i = |V(G)|$ and $s_i \geq 0$ for all $i \in [\![0, k+1]\!]$. Notice that the number of such combinations is no more than $(|V(G)|+1)^{k+2}$. For each combination, we solve SPECIFIED SIZE GLOBAL $[k]$-ROMAN DOMINATION, and we retain the solution of minimum weight.

6.2 k-community, Max PDS and other variants

The notion of *community structure* was first introduced in [22], as a partition $\{C_1, \ldots, C_k\}$, with $k \geq 2$, of the set of vertices of a graph into so called *communities*, such that for each $i \in [\![1, k]\!]$ we have $|C_i| \geq 2$ and, for each vertex $v \in C_i$ and each community $C_j \neq C_i$, $\frac{|N_G(v) \cap C_i|}{|C_i|-1} \geq \frac{|N_G(v) \cap C_j|}{|C_j|}$. Finding a community structure in any graph G can be done in polynomial time (see [22]). However, the number of communities k in the obtained community structure can be any value between 2 and $\frac{|V(G)|}{2}$, and the algorithm does not apply when we want to impose the number of communities. The 2-COMMUNITY problem was introduced in [2] as the problem of deciding whether a given connected graph has a *2-community structure*, i.e. a community structure with 2 communities. This can be naturally generalized to the k-COMMUNITY problem, for any fixed k, as the problem of deciding whether a given connected graph has a community structure with k communities. The complexity status of 2-COMMUNITY is still unknown, and only a few graph classes are known to admit polynomial time algorithms for this problem (for instance, graphs of maximum degree 3 and graphs of minimum degree $|V(G)| - 3$ [2]).

We show here that k-COMMUNITY is XP parameterized by clique-width. Our approach is similar to the one for GLOBAL $[k]-$ROMAN DOMINATION, in the sense that we define a variant of the problem where we require a certain size of each community, to which we reduce k-COMMUNITY.

SPECIFIED SIZE k-COMMUNITY
Instance: A graph G and k integers $s_1, \ldots, s_k \geq 2$, such that $\sum_{i=1}^{k} s_i = |V(G)|$.
Question: Does G admit a k-community structure $\{C_1, \ldots, C_k\}$ such that $|C_i| = s_i$ for all $i \in [\![1, k]\!]$?

The SPECIFIED SIZE k-COMMUNITY problem can be modeled as a color-counting 1-locally checkable problem with global properties. Notice that since it is a decision problem, we only need two values for the weight set.

- COLORS $= [\![1, k]\!]$ and $L_v = [\![1, k]\!]$ for all $v \in V(G)$;
- (WEIGHTS, $\preceq, \circledast$) $= (\{0, 1\}, \leq, \max)$ and $\mathrm{w}_{v,a} = 0$ for all $v \in V(G), a \in L_v$;
- $check(v, a, n_1, \ldots, n_k) = \left(\forall b \in [\![1, k]\!], \frac{n_a}{s_a - 1} \geq \frac{n_b}{s_b}\right)$;
- for all $a \in$ COLORS, we ask for the size of the color class of a to belong to $\{s_a\}$.

Then, k-COMMUNITY can be solved by successively iterating over the feasible combinations of values $s_1, \ldots, s_k$ such that $\sum_{i=1}^{k} s_i = |V(G)|$ and $s_i \geq 2$ for all $i \in [\![1, k]\!]$, and for each of the combinations solving SPECIFIED SIZE k-COMMUNITY.

Note that BALANCED k-COMMUNITY, i.e. the problem of finding a k-community structure with all parts having the same size, is equivalent to SPECIFIED SIZE k-COMMUNITY with $s_i = s_j$, for all $i, j \in [\![1, k]\!]$. Hence, it is also XP parameterized by clique-width. In [13], it was shown that this problem is NP-complete in general, and in [2] it was pointed out to be polynomially solvable in graphs of bounded treewidth. It is not difficult to see that the problem WEAK k-COMMUNITY, defined in [2], can also be solved by slightly modifying the above check function.

A closely related problem is the MAXIMUM PROPORTIONALLY DENSE SUBGRAPH (MAX PDS) problem, originally defined in [3]. Let G be a graph and $S \subset V(G)$, such that $2 \leq |S| < |V(G)|$. We say that the induced subgraph $G[S]$ is a *proportionally dense subgraph (PDS)* if for every $v \in S$, we have $\frac{|N_G(v) \cap S|}{|S|-1} \geq \frac{|N_G(v) \cap \overline{S}|}{|\overline{S}|}$. Then, the MAX PDS problem consists in finding a proportionally dense subgraph in G of maximum size. The authors of [3] showed that the MAX PDS problem is NP-hard, even when restricted to split graphs or bipartite graphs, and that it can be solved in linear time in cubic Hamiltonian graphs.

By proceeding in a similar way as before, where in the associated auxiliary problem we have only two colors, S and $\overline{\text{S}}$, and the check function is given by $check(v, a, n_0, n_1) = \left(a = \text{S} \Rightarrow \frac{n_{\text{S}}}{s_{\text{S}} - 1} \geq \frac{n_{\overline{\text{S}}}}{s_{\overline{\text{S}}}}\right)$, we can show that MAX PDS is XP parameterized by clique-width.

Another variation defined in [3] is the PDS EXTENSION problem, which asks whether there exists a proportionally dense subgraph $G[S]$ such that $U \subset S$, for some $U \subset V(G)$ given as an input. It was shown in [3] that the PDS EXTENSION problem is NP-complete, and no polynomial time algorithms were known for any graph class. We can show that this problem is also XP parameterized by clique-width, by proceeding almost exactly as explained above, where the only change is that we now set $L_v = \{\text{S}\}$ for all $v \in U$.

Given a graph G and a real number $\gamma \in (0, 1]$, a *degree-based γ-quasi-clique* is defined as a subset $S \subseteq V(G)$ such that the degree of any vertex in $G[S]$ is at least $\gamma(|S| - 1)$, that is, $\frac{|N_G(v) \cap S|}{|S|-1} \geq \gamma$. The MAXIMUM DEGREE-BASED γ-QUASI-CLIQUE problem consists in finding a degree-based γ-quasi-clique of maximum cardinality in a graph. In [24], it was shown that this problem is NP-hard for any fixed γ. Using the same techniques as for MAX-PDS (only slightly modifying the check function), we obtain that MAXIMUM DEGREE-BASED γ-QUASI-CLIQUE is XP parameterized by clique-width.

References

1 H. Abdollahzadeh Ahangar, M.P. Álvarez, M. Chellali, S.M. Sheikholeslami, and J.C. Valenzuela-Tripodoro. Triple roman domination in graphs. *Applied Mathematics and Computation*, 391:125444, 2021. doi:10.1016/j.amc.2020.125444.

2 Cristina Bazgan, Janka Chlebikova, and Thomas Pontoizeau. Structural and algorithmic properties of 2-community structures. *Algorithmica*, 80:1890–1908, 2018. doi:10.1007/s00453-017-0283-7.

3 Cristina Bazgan, Janka Chlebíková, Clément Dallard, and Thomas Pontoizeau. Proportionally dense subgraph of maximum size: Complexity and approximation. *Discrete Applied Mathematics*, 270:25–36, 2019. doi:10.1016/j.dam.2019.07.010.

4 Robert A. Beeler, Teresa W. Haynes, and Stephen T. Hedetniemi. Double Roman domination. *Discrete Applied Mathematics*, 211:23–29, 2016. doi:10.1016/j.dam.2016.03.017.

5 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. A logic-based algorithmic meta-theorem for mim-width, 2022. doi:10.48550/arXiv.2202.13335.

6 Flavia Bonomo-Braberman and Carolina Lucía Gonzalez. A new approach on locally checkable problems. *Discrete Applied Mathematics*, 314:53–80, 2022. doi:10.1016/j.dam.2022.01.019.

7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.

8 Yair Caro and Michael S. Jacobson. On non-z(mod k) dominating sets. *Discussiones Mathematicae Graph Theory*, 23(1):189–199, 2003. doi:10.7151/dmgt.1195.

9 Ernie J Cockayne, Paul A Dreyer, Sandra M Hedetniemi, and Stephen T Hedetniemi. Roman domination in graphs. *Discrete Mathematics*, 278(1):11–22, 2004. doi:10.1016/j.disc.2003.06.004.

10 B. Courcelle, J. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Systems*, 33:125–150, 2000. doi:10.1007/s002249910009.

11 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

12 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

13 Vladimir Estivill-Castro and Mahdi Parsa. Hardness and tractability of detecting connected communities. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '16, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2843043.2843053.

14 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1):3–31, 2004. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS). doi:10.1016/j.apal.2004.01.007.

15 Elisabeth Gassner and Johannes Hatzl. A parity domination problem in graphs with bounded treewidth and distance-hereditary graphs. *Computing*, 82:171–187, 2008. doi:10.1007/s00607-008-0005-8.

16 Michael U. Gerber and Daniel Kobler. Algorithms for vertex-partitioning problems on graphs with fixed clique-width. *Theoretical Computer Science*, 299:719–734, 2003.

17 Carolina Lucía Gonzalez and Felix Mann. On d-stable locally checkable problems on bounded mim-width graphs, 2022. doi:10.48550/arXiv.2203.15724.

18 T.W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs*. CRC Press, 1st edition, 1998. doi:10.1201/9781482246582.

19 John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1990.

20 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Generalized distance domination problems and their complexity on graphs of bounded mim-width, 2018. doi:10.48550/arXiv.1803.03514.

21 F. Nahani Pour, H. Abdollahzadeh Ahangar, M. Chellali, and S.M. Sheikholeslami. Global triple roman dominating function. *Discrete Applied Mathematics*, 314:228–237, 2022.

22 Martin Olsen. A general view on computing communities. *Mathematical Social Sciences*, 66(3):331–336, 2013. doi:10.1016/j.mathsocsci.2013.07.002.

23 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

24 Grigory Pastukhov, Alexander Veremyev, Vladimir Boginski, and Oleg A. Prokopyev. On maximum degree-based -quasi-clique problem: Complexity and exact approaches. *Networks*, 71(2):136–152, 2018. doi:10.1002/net.21791.

25 P. Roushini Leely Pushpam and S. Padmapriea. Global roman domination in graphs. *Discrete Applied Mathematics*, 200:176–185, 2016.

26 Zehui Shao, S. M. Sheikholeslami, S. Nazari-Moghaddam, and Shaohui Wang. Global double roman domination in graphs. *Journal of Discrete Mathematical Sciences and Cryptography*, 22(1):31–44, 2019.

27 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

28 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2001.

A Examples and omitted lemmas

We include here examples of color-counting 1-locally checkable problems, and the lemmas omitted from Section 5. Due to space constraints, only some of the proofs are presented.

A.1 Examples of color-counting 1-locally checkable problems

▶ **Example 14.** Consider the k-Coloring problem. This problem can be seen as a color-counting 1-locally checkable problem with the following characteristics:

- $\textsc{Colors} = [\![1, k]\!]$ and $L_v = [\![1, k]\!]$ for all $v \in V(G)$;
- $(\textsc{Weights}, \preceq, \circledast) = (\{0, 1\}, \leq, \max)$ and $\mathrm{w}_{v,a} = 0$ for all $v \in V(G), a \in L_v$;
- $\mathit{check}(v, a, n_1, \ldots, n_k) = (n_a = 0)$.

▶ **Example 15.** The Maximum Independent Set problem can also be modeled as a color-counting 1-locally checkable problem:

- $\textsc{Colors} = \{0, 1\}$ and $L_v = \{0, 1\}$ for all $v \in V(G)$;
- $(\textsc{Weights}, \preceq, \circledast) = (\mathbb{N} \cup \{-\infty\}, \geq, +)$ and $\mathrm{w}_{v,a} = a$ for all $v \in V(G), a \in L_v$;
- $\mathit{check}(v, a, n_0, n_1) = (a = 0 \vee n_1 = 0)$.

▶ **Example 16.** The Minimum Odd Dominating Set problem can as well be modeled as a color-counting 1-locally checkable problem, as follows:

- $\textsc{Colors} = \{0, 1\}$ and $L_v = \{0, 1\}$ for all $v \in V(G)$;
- $(\textsc{Weights}, \preceq, \circledast) = (\mathbb{N} \cup \{+\infty\}, \leq, +)$ and $\mathrm{w}_{v,a} = a$ for all $v \in V(G), a \in L_v$;
- $\mathit{check}(v, a, n_0, n_1) = (a + n_1 \equiv 1 \mod 2)$.

A.2 Omitted lemmas of Section 5

In what follows, we will write $\widehat{p}$ instead of $p_1, \ldots, p_m$ to make the notation less cumbersome. Note that $\widehat{p}$ is empty when $m = 0$.

▶ **Lemma 17.** *Let Π be color-counting 1-locally checkable problem with a set of global properties Γ, input graph G with a clique-width k-expression e_G of G, and let $a \in$ Colors. Let $\sigma \subseteq \mathbb{N}$ and let $(Q, \{1\}, \delta, q_0, F)$ be a deterministic finite-state automaton that accepts a string of n consecutive 1's if and only if $n \in \sigma$. Let $\in_F \colon Q \to$ Bool be the function such that $\in_F (q) = (q \in F)$.*

Assume that the minimum weight of a proper coloring of G satisfying Γ equals

$$\min\{\lambda(e_G, C, N, \widehat{p}) : P(C, N, \widehat{p}) = \textsc{True}\}$$

for some property P. Furthermore, assume that

(1) *for every proper coloring c of G satisfying Γ there exist $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that c is a $(C, N, \widehat{p})$-coloring of G;*

(2) *for every $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that $\lambda(e_G, C, N, \widehat{p}) \neq$ Error, every $(C, N, \widehat{p})$-coloring of G is a proper coloring of G satisfying Γ.*

Then, the minimum weight of a proper coloring c of G satisfying Γ and such that $|\{v \in V(G) : c(v) = a\}| \in \sigma$ equals

$$\min\{\lambda(e_G, C, N, \widehat{p}, q_0, \in_F) : P(C, N, \widehat{p}) = \textsc{True}\}.$$

Moreover,

(a) *for every proper coloring c of G satisfying Γ and such that $|\{v \in V(G) : c(v) = a\}| \in \sigma$, there exist $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that c is $(C, N, \widehat{p}, q_0, \in_F)$-coloring of G,*

(b) *for every $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that $\lambda(e_G, C, N, \widehat{p}, q_0, \in_F) \neq$ Error, every $(C, N, \widehat{p}, q_0, \in_F)$-coloring c of G is a proper coloring of G satisfying Γ and such that $|\{v \in V(G) : c(v) = a\}| \in \sigma$.*

Proof. We will first show that for every proper coloring c of G satisfying Γ and such that $|\{v \in V(G) : c(v) = a\}| \in \sigma$, there exist $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that $\mathrm{w}(c) \geq \lambda(e_G, C, N, \widehat{p}, q_0, \in_F)$. Suppose we have such a proper coloring c of G satisfying Γ and such that $|\{v \in V(G) : c(v) = a\}| \in \sigma$. By assumption (1), we know that there exist $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and c is a $(C, N, \widehat{p})$-coloring of G. Since we are assuming that $|\{v \in V(G) : c(v) = a\}| \in \sigma$, and since the automaton accepts a string of t consecutive 1's if and only if $t \in \sigma$, it follows that $\in_F (\delta^n(q_0)) =$ True, where $n = |\{v \in V(G) : c(v) = a\}|$. Thus, c is a $(C, N, \widehat{p}, q_0, \in_F)$-coloring of G and $\mathrm{w}(c) \geq \lambda(C, N, \widehat{p}, q_0, \in_F)$.

On the other hand, we will now show that for every $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that $\lambda(e_G, C, N, \widehat{p}, q_0, \in_F) \neq$ Error, there exists a proper coloring c of G satisfying Γ and such that $|\{v \in V(G) : c(v) = a\}| \in \sigma$ with $\mathrm{w}(c) = \lambda(e_G, C, N, \widehat{p}, q_0, \in_F)$. So suppose we have $C, N, \widehat{p}$ such that $P(C, N, \widehat{p}) =$ True and such that $\lambda(C, N, \widehat{p}, q_0, \in_F) \neq$ Error. By the latter assumption and by the definition of $\lambda(C, N, \widehat{p}, q_0, \in_F)$, we get that $\lambda(C, N, \widehat{p}) \neq$ Error. Let c be a $(C, N, \widehat{p}, q_0, \in_F)$-coloring of G (notice that at least one such c exists). By definition, c is a $(C, N, \widehat{p})$-coloring of G. Then we conclude by assumption (2) that c is a proper coloring of G satisfying Γ. Finally, since c is a $(C, N, \widehat{p}, q_0, \in_F)$-coloring of G, we have that $\in_F (\delta^n(q_0)) =$ True, where $n = |\{v \in V(G) : c(v) = a\}|$, which implies that $|\{v \in V(G) : c(v) = a\}| \in \sigma$. If we consider in particular c of minimum weight, then $\mathrm{w}(c) = \lambda(e_G, C, N, \widehat{p}, q_0, \in_F)$.

Notice that (a) and (b) are implicitly shown by the above. ◀

▶ **Lemma 18** (Creating new vertex: $i(v)$)**.** *If $C[i,a] = 1$ and $f_a(\delta(s_a,1)) = \textsc{True}$, or if $C[i,a] = 0$ and $f_a(s_a) = \textsc{True}$, then*

$$\lambda(i(v), C, N, \widehat{p}, s_a, f_a) = \lambda(i(v), C, N, \widehat{p}).$$

Otherwise, $\lambda(i(v), C, N, \widehat{p}, s_a, f_a) = \textsc{Error}$.

▶ **Lemma 19** (Disjoint union: $e_1 \oplus e_2$)**.** *Assume that*

$$\lambda(e_1 \oplus e_2, C, N, \widehat{p}) = \min\{\lambda(e_1, C_1, N_1, \widehat{p_1}) \circledast \lambda(e_2, C_2, N_2, \widehat{p_2}) : P(C, N, \widehat{p}, C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}) = \textsc{True}\}$$

for some property P. Moreover, assume that

(1) *for every $(C, N, \widehat{p})$-coloring c of $G_{e_1 \oplus e_2}$ there exist $C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}$ such that $P(C, N, \widehat{p}, C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}) = \textsc{True}$ and such that $c|_{V(G_{e_1})}$ is a $(C_1, N_1, \widehat{p_1})$-coloring of G_{e_1} and $c|_{V(G_{e_2})}$ is a $(C_2, N_2, \widehat{p_2})$-coloring of G_{e_2};*

(2) *for all $C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}$ such that $P(C, N, \widehat{p}, C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}) = \textsc{True}$, if c_1 is a $(C_1, N_1, \widehat{p_1})$-coloring of G_{e_1} and c_2 is a $(C_2, N_2, \widehat{p_2})$-coloring of G_{e_2}, then $c = c_1 \cup c_2$ is a $(C, N, \widehat{p})$-coloring of $G_{e_1 \oplus e_2}$.*

Then,

$$\lambda(e_1 \oplus e_2, C, N, \widehat{p}, s_a, f_a) = \min\{\lambda(e_1, C_1, N_1, \widehat{p_1}, s_a, eq_q) \circledast \lambda(e_2, C_2, N_2, \widehat{p_2}, q, f_a) : q \in Q \text{ and } P(C, N, \widehat{p}, C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}) = \textsc{True}\}.$$

Moreover,

(a) *for every $(C, N, \widehat{p}, s_a, f_a)$-coloring c of $G_{e_1 \oplus e_2}$ there exist q, $C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}$ such that $q \in Q$, $P(C, N, \widehat{p}, C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}) = \textsc{True}$, and $c|_{V(G_{e_1})}$ is a $(C_1, N_1, \widehat{p_1}, s_a, eq_q)$-coloring of G_{e_1} and $c|_{V(G_{e_2})}$ is a $(C_2, N_2, \widehat{p_2}, q, f_a)$-coloring of G_{e_2};*

(b) *for all q, $C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}$ such that $q \in Q$ and $P(C, N, \widehat{p}, C_1, N_1, \widehat{p_1}, C_2, N_2, \widehat{p_2}) = \textsc{True}$, if c_1 is a $(C_1, N_1, \widehat{p_1}, s_a, eq_q)$-coloring of G_{e_1} and c_2 is a $(C_2, N_2, \widehat{p_2}, q, f_a)$-coloring of G_{e_2} then $c = c_1 \cup c_2$ is a $(C, N, \widehat{p}, s_a, f_a)$-coloring of $G_{e_1 \oplus e_2}$.*

▶ **Lemma 20** (Join: $\eta_{i,j}(e)$)**.** *Assume that there exist $C', N', \widehat{p'}$ such that c is a $(C, N, \widehat{p})$-coloring of $G_{\eta_{i,j}(e)}$ if and only if c is a $(C', N', \widehat{p'})$-coloring of G_e.*

Then, c is a $(C, N, \widehat{p}, s_a, f_a)$-coloring of $G_{\eta_{i,j}(e)}$ if and only if c is a $(C', N', \widehat{p'}, s_a, f_a)$-coloring of G_e. In particular,

$$\lambda(\eta_{i,j}(e), C, N, \widehat{p}, s_a, f_a) = \lambda(e, C', N', \widehat{p'}, s_a, f_a).$$

▶ **Lemma 21** (Relabeling: $\rho_{i \to j}(e)$)**.** *Assume that*

$$\lambda(\rho_{i \to j}(e), C, N, \widehat{p}) = \min\{\lambda(e, C_e, N_e, \widehat{p_e}) : P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}\}$$

for some property P. Moreover, assume that

(1) *for every $(C, N, \widehat{p})$-coloring c of $G_{\rho_{i \to j}(e)}$, there exist parameters $C_e, N_e, \widehat{p_e}$ such that $P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}$ and c is a $(C_e, N_e, \widehat{p_e})$-coloring of G_e;*

(2) *for all parameters $C_e, N_e, \widehat{p_e}$ such that $P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}$, if c is a $(C_e, N_e, \widehat{p_e})$-coloring of G_e, then c is also a $(C, N, \widehat{p})$-coloring c of $G_{\rho_{i \to j}(e)}$.*

Then,

$$\lambda(\rho_{i\to j}(e), C, N, \widehat{p}, s_a, f_a) = \min\{\lambda(e, C_e, N_e, \widehat{p_e}, s_a, f_a) : P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}\}.$$

Moreover,

(a) *for every $(C, N, \widehat{p}, s_a, f_a)$-coloring c of $G_{\rho_{i\to j}(e)}$, there exist parameters $C_e, N_e, \widehat{p_e}$ such that $P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}$ and c is a $(C_e, N_e, \widehat{p_e}, s_a, f_a)$-coloring of G_e;*

(b) *for all parameters $C_e, N_e, \widehat{p_e}$ such that $P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}$, if c is a $(C_e, N_e, \widehat{p_e}, s_a, f_a)$-coloring of G_e, then c is also a $(C, N, \widehat{p}, s_a, f_a)$-coloring c of $G_{\rho_{i\to j}(e)}$.*

Proof. Let $\alpha = \min\{\lambda(e, C_e, N_e, \widehat{p_e}, s_a, f_a) : P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}\}$. We will first prove that $\lambda(\rho_{i\to j}(e), C, N, \widehat{p}, s_a, f_a) \geq \alpha$. Let c be a $(C, N, \widehat{p}, s_a, f_a)$-coloring of $G_{\rho_{i\to j}(e)}$. We show that there exist parameters $C_e, N_e, \widehat{p_e}$ such that $P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}$ and c is a $(C_e, N_e, \widehat{p_e}, s_a, f_a)$-coloring of G_e. By definition, c is a $(C, N, \widehat{p})$-coloring of $G_{\rho_{i\to j}(e)}$ such that $f_a(\delta^n(s_a)) = \textsc{True}$, where $n = |\{v \in V(G_{\rho_{i\to j}(e)}) : c(v) = a\}|$. Therefore, by assumption (1), there exist parameters $C_e, N_e, \widehat{p_e}$ such that $P(C, N, \widehat{p}, C_e, N_e, \widehat{p_e}) = \textsc{True}$ and c is a $(C_e, N_e, \widehat{p_e})$-coloring of G_e. Furthermore, since $n_e = |\{v \in V(G_e) : c(v) = a\}| = |\{v \in V(G_{\rho_{i\to j}(e)}) : c(v) = a\}| = n$, it immediately follows that $f_a(\delta^{n_e}(s_a)) = \textsc{True}$. Thus, c is a $(C_e, N_e, \widehat{p_e}, s_a, f_a)$-coloring of G_e. If we consider in particular a $(C, N, \widehat{p}, s_a, f_a)$-coloring c of $G_{\rho_{i\to j}(e)}$ of minimum weight, then $\lambda(\rho_{i\to j}(e), C, N, \widehat{p}, s_a, f_a) = \mathrm{w}(c) \geq \lambda(e, C_e, N_e, \widehat{p_e}, s_a, f_a) \geq \alpha$.

Let us now show that $\lambda(\rho_{i\to j}(e), C, N, \widehat{p}, s_a, f_a) \leq \alpha$. Let $C_e, N_e, \widehat{p_e}$ be such that $P(C, N, \widehat{p}, C_e,\ N_e, \widehat{p_e}) = \textsc{True}$. Let c be a $(C_e, N_e, \widehat{p_e}, s_a, f_a)$-coloring of G_e. By definition, c is a $(C_e, N_e, \widehat{p_e})$-coloring of G_e such that $f_a(\delta^{n_e}(s_a)) = \textsc{True}$, where $n_e = |\{v \in V(G_e) : c(v) = a\}|$. Then, by assumption (2), c is also a $(C, N, \widehat{p})$-coloring c of $G_{\rho_{i\to j}(e)}$. Furthermore, since $n = |\{v \in V(G_{\rho_{i\to j}(e)}) : c(v) = a\}| = |\{v \in V(G_e) : c(v) = a\}| = n_e$, it immediately follows that $f_a(\delta^n(s_a)) = \textsc{True}$. Therefore, c is a $(C, N, \widehat{p}, s_a, f_a)$-coloring c of $G_{\rho_{i\to j}(e)}$. If we consider in particular a $(C_e, N_e, \widehat{p_e}, s_a, f_a)$-coloring c of G_e for which α is obtained, then $\alpha = \mathrm{w}(c) \geq \lambda(\rho_{i\to j}(e), C, N, \widehat{p}, s_a, f_a)$.

Notice that (a) and (b) are implicitly shown by the above. ◀

A.2.1 Complexity of the modified algorithm

First, assume that for any possible parameter f_a and state q, $\delta(s, 1)$ and $f_a(q)$ can be computed in constant time. Also, assume that we want to fix the size of $\mathcal{R}$ color classes $a_1, \ldots, a_{\mathcal{R}}$. Let $\mathcal{S}$ be the size of the largest set of states among the $\mathcal{R}$ considered automata. Then, we add a term $\mathcal{R}$ to the complexity corresponding to the operation of creating a new labeled vertex, and we multiply by a term $\mathcal{S}^{\mathcal{R}}$ the complexity corresponding to the disjoint union operation. Moreover, whenever we go through all the possible $\lambda(e, C, N, \widehat{p}, s_{a_1}, f_{a_1}, \ldots, s_{a_{\mathcal{R}}}, f_{a_{\mathcal{R}}})$, we multiply the complexity by a factor $(\mathcal{S}(\mathcal{S}+1))^{\mathcal{R}}$. Hence, since $\mathcal{R}$ is at most the number of colors and $\mathcal{S} \leq |V(G)|$, we conclude that the new algorithm is also XP parameterized by clique-width when the number of colors is constant.

Lower Bounds on Retroactive Data Structures

Lily Chung
Massachusetts Institute of Technology, Cambridge, MA, USA

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA

Dylan Hendrickson
Massachusetts Institute of Technology, Cambridge, MA, USA

Jayson Lynch
Cheriton School of Computer Science, University of Waterloo, Canada

Abstract

We prove essentially optimal fine-grained lower bounds on the gap between a data structure and a partially retroactive version of the same data structure. Precisely, assuming any one of three standard conjectures, we describe a problem that has a data structure where operations run in $O(T(n,m))$ time per operation, but any partially retroactive version of that data structure requires $T(n,m) \cdot m^{1-o(1)}$ worst-case time per operation, where n is the size of the data structure at any time and m is the number of operations. Any data structure with operations running in $O(T(n,m))$ time per operation can be converted (via the "rollback method") into a partially retroactive data structure running in $O(T(n,m) \cdot m)$ time per operation, so our lower bound is tight up to an $m^{o(1)}$ factor common in fine-grained complexity.

2012 ACM Subject Classification Theory of computation → Cell probe models and lower bounds

Keywords and phrases Retroactivity, time travel, rollback, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.32

Related Version *arXiv Version*: `https://arXiv.org/abs/2211.14664`

Acknowledgements This work was initiated during open problem solving in the MIT class on Advanced Data Structures (6.851) in Spring 2021. We thank the other participants of that class – in particular, Joshua Ani, Josh Brunner, and Naveen Venkat – for related discussions and providing an inspiring atmosphere. We also thank Michael Coulombe for helpful pointers regarding time travel.

1 Introduction

A trope in popular science fiction is the idea of traveling back in time, making a change to the world, then traveling forward in time (usually to the original time) to observe the effected changes. This style of time travel (as opposed to stable time loops or multiverses) was one of the inspirations for retroactive data structures. Such sequences of events occur in many pieces of media including in the movies *Back To The Future* (1985), *Timecop* (1994), *12 Monkeys* (1995), *Star Trek: First Contact* (1996), *The Butterfly Effect* (2004), *Looper* (2012), and *Avengers: Endgame* (2019); TV shows *Doctor Who* (1963/2005–), *Heroes* (2006–), *Marvel's Agents of S.H.I.E.L.D.* (2013–), *DC's Legends of Tomorrow* (2016–), and *The Umbrella Academy* (2019–); anime/manga *Doraemon* (1969/1973–) and *Steins;Gate* (2009/2011–); short story *A Sound of Thunder* (Ray Bradbury, 1952); books *The Hitchhiker's Guide to the Galaxy* (Douglas Adams, 1979) and *Pastwatch: The Redemption of Christopher Columbus* (Orson Scott Card, 1996); and video games *Chrono Trigger* (1995) and *The Legend of Zelda: Ocarina of Time* (1998).

Is this kind of "jump back, change, jump forward" or "Back To The Future" time travel realistic? It is inconsistent with known models of physics, but it is nonetheless interesting to prove inconsistency with other assumptions about the physical world. Here we give evidence of such inconsistency under a ***fine-grained physical Church–Turing thesis***:

the universe's physics while traveling forward in time are implemented by a computer with similar power (up to polylogarithmic or $n^{o(1)}$ factors) to standard theoretical computers such as the word RAM, possibly randomized or quantum, or possibly a parallel system of such computers. (The word RAM is the standard in data structures and most of theoretical computer science, and it can efficiently simulate e.g. an $O(1)$-head d-dimensional Turing machine for any d, making it a good choice of universal model, at least for lower bounds.) Under this assumption, is time travel *computationally* realistic?

We can turn such questions about computational realism into problems about data structures, by thinking of the state of the universe as a data structure. For example, jumping back in time (for viewing only) is essentially asking for a *partially persistent* universe computation, while jumping back in time and making a change to branch a new universe is essentially asking for a *fully persistent* universe computation, both of which are efficiently possible for essentially any universe computation (with logarithmic overhead in general, and constant overhead assuming bounded in-degree which may be reasonable given geometric constraints) [7]. Returning to "Back to the Future" time travel, making a (blind) change in the past and instantly seeing the cascaded effects on the present is essentially asking for a *partially retroactive* universe computation, while observing and changing the past and immediately updating the future is essentially asking for a *fully retroactive* universe computation.[1]

1.1 Our Results

In this paper, we give new evidence that data structures cannot be made efficiently retroactive, even partially retroactive, which complements a prior separation between partial and full retroactivity [4]. Specifically, under any of three standard assumptions in fine-grained complexity, we prove near-optimality of the trivial "rollback" method for retroactive changes to the past [5, Theorem 1]: rewind time by undoing operations to when the change should occur, make the desired change, and then replay the previously undone operations (by maintaining the current timeline of changes and their inverses). In the worst case (and assuming no amortization in the input data structure), this method occurs a multiplicative overhead of $\Theta(m)$ where m is the number of operations in the timeline. We prove an essentially matching worst-case lower bound of $\Omega(m^{1-o(1)})$ multiplicative overhead under any one of the following assumptions, which are all common in fine-grained complexity [14, 4] and were the same assumptions made by the prior retroactive separation results [4]:

1. CircuitSAT on an n-input $2^{o(n)}$-gate circuit requires $2^{n-o(n)}$ time.
   This assumption is weaker than the standard Strong Exponential Time Hypothesis (SETH) [13] which makes a similar statement about formulas (which cannot re-use computations like circuits can).
2. $(\min, +)$ matrix–vector product with an $n \times n$ integer matrix and an online sequence of n integer vectors of length n requires $n^{3-o(1)}$ time.
   This assumption is weaker than the standard APSP assumption that All-Pairs Shortest Paths in an n-vertex graph with integer weights (equivalent to *offline* $(\min, +)$ matrix–vector product) requires $n^{3-o(1)}$ time [12].
3. 3SUM (given n integers, do any three sum to zero?) requires $n^{2-o(1)}$ time [9].
   This assumption is arguably the beginning of fine-grained complexity.

Our lower bounds hold even if the retroactive data structure is amortized, and apply to randomized algorithms if the assumptions do.

[1] For this reason, an early unused name for retroactive data structures was "time-travel data structures", as evidenced by the keyword list of [5].

Applied to time travel, our results show that making a change in the past and then jumping forward to the present should take roughly as much computation as the universe needed to originally advance through that much time. In other words, adding "Back To The Future" time travel to the universe requires the universe computation to slow down by a factor nearly linear in the lifetime of the universe, which seems computationally unrealistic. Here we assume the fine-grained physical Church–Turing Thesis described above, and ignore possible speedups from quantum computation (which remains an area for future research). If the universe is a parallel system of computers, then our worst-case instance consists of a proportional number of instances of our worst-case data structures. Our result can be seen as justification for the model of time travel implemented in the video game *Achron* (2011), where changing the past propagates a wave of consequential changes to the future, but the wave moves "slowly" (a constant factor faster than normal forward time travel).

1.2 Related Work

The original paper on retroactive data structures [5] proves an $\Omega(m)$ worst-case lower bound on the multiplicative overhead required for partial retroactivity, where m is the number of operations in the timeline [5, Theorem 2]. Asymptotically, this result is an improvement over our result, matching the $O(m)$ upper bound up to a constant factor instead of an $m^{o(1)}$ factor. The difference is in the model: the existing $\Omega(m)$ lower bound holds in the history-dependent algebraic-computation-tree model, the integer RAM, and generalized real RAM. (The reduction is from online polynomial evaluation, which provably requires linear time in these models.) But the result is not known to hold on the word RAM, for example, where the best known unconditional lower bound on retroactivity is $\Omega(\sqrt{m/\log m})$ [5, Theorem 3].

By contrast, our results hold wherever the fine-grained assumptions above hold, which so far seems to include all "reasonable" models of computation. For example, the fastest known algorithm for 3SUM on a word RAM runs in $O\left(n^2\left(\frac{\lg\lg n}{\lg n}\right)^2\right)$ time [2]. This bound is roughly a logarithmic factor better than the fastest known algorithm for 3SUM on a real RAM, which runs in $O\left(n^2\,\frac{\lg\lg n}{\lg n}\right)$ time [8], an improvement on the recent subquadratic breakthrough [11]. But all of these algorithms are quadratic up to polylogarithmic factors.

On the other hand, in the decision tree model of computation, 3SUM can be solved in $O(n^{3/2}\sqrt{\log n})$ time [11], breaking the 3SUM conjecture. However, this model of computation is generally considered *unreasonable* because it allows the choice of algorithm to depend on the problem size n. Further, like most such decision-tree results, the proof is not "algorithmic": the best known algorithm to compute which algorithm to run for a given n uses exponential time.

Our work is closely based on another paper proving a separation between partial and full retroactivity [4]. In particular, our fine-grained complexity assumptions are the same as theirs. Under these assumptions, Chen et al. [4] proved that the worst-case separation between a partially retroactive and a fully retroactive data structure for the same underlying problem is a multiplicative factor of $\Omega(m^{1/2-o(1)})$, which is essentially tight against the known upper bound of $O(\sqrt{m})$ [5].

This paper is not the first to consider applications of computational complexity to time travel. Aaronson, Bavarian, and Giusteri [1] show that consistent timelines from closed timelike curves (another common model for time travel in popular science fiction) require solving PSPACE-complete problems. This result suggests that that this model of time travel, despite being plausible in existing models of physics, is not actually computationally feasible. Our result can be viewed as a complementary negative result for the "Back To The

Future" model of time travel. (However, our result shows only a conjectured linear separation, whereas [1] shows a conjectured exponential separation.) Both negative results only apply to "long-distance" time travel; making changes in the recent past remains computationally feasible, leaving ample room for science fiction. Both results also leave open the possibility that a time traveler jumping forward simply does not experience time (their subjective time is frozen) while the universe rolls time forward.

1.3 Techniques

Our approach builds on previous work by Chen et al. [4], which proves an $\tilde{\Omega}(\sqrt{m})$ gap between partial and full retroactivity under the same fine-grained complexity assumptions, where m is the number of operations in the data structure's timeline. This result implies an $\tilde{\Omega}(\sqrt{m})$ worst-case separation between a basic (nonretroactive) data structure and a fully retroactive version of that data structure. Our work improves on this separation in two ways: it provides a separation between a basic data structure and a *partially* retroactive version of that data structure, and it improves the separation from $\tilde{\Omega}(\sqrt{m})$ to $\tilde{\Omega}(m)$.

To achieve these improvements, we adapt Chen et al.'s constructions using the following techniques:

1. We define the "lazy" version of an arbitrary abstract data type (data structure interface), which allows us to move all queries to the end while still preserving their answers, making partial and full retroactivity essentially equivalent. We prove a general result (Theorem 8) which lets us translate gaps for full retroactivity into gaps for partial retroactivity.
2. We use fewer operations than Chen et al. by combining a long sequence of simple operations into one more complicated operation. In particular, Chen et al.'s data structures use one operation to set a single element of a list, while ours set the entire list in a single operation. At the cost of operations being more expensive, this reduces the number of operations performed. As a result, the same gap is larger when considered as a function of the number of operations.

Using Technique 1 alone, applying Theorem 8 to Chen et al.'s results, we obtain an $\tilde{\Omega}(\sqrt{m})$ gap between nonretroactivity and partial retroactivity, which is interesting but not optimal. Using Technique 2 without Technique 1 does not achieve anything new, because Chen et al. use a large number of nonconsecutive query operations. To significantly reduce the number of operations, we need to batch the query operations into a single more complicated operation, which our lazy transform makes possible by moving them all to the end.

1.4 Outline

In Section 2, we provide the definitions needed for this paper, including fully and partially retroactive data structures, and the three fine-grained complexity assumptions.

In Section 3, we define lazy data structures, and use them to prove a result that converts gaps for full retroactivity into gaps for partial retroactivity. Applying this to Chen et al.'s work gives a $\Omega(\sqrt{m})$ gap between a base data structure and the partial retroactive version, under each of the three fine-grained complexity assumptions.

In the remaining sections, we prove a stronger $\Omega(m)$ slowdown for partial retroactivity under each of the same assumptions. Each proof is based on a construction due to Chen et al. [4], with Lazy applied. We also apply the second idea described above–merging consecutive operations into a single more complicated operation–which gives the improvement from $\Omega(\sqrt{m})$ to $\Omega(m)$. We are also able to make a few simplifications: first, we do not always need the full power of inserting queries and calling EVALUATE, and can use an ADT which can only query some simpler summary of past states. Second, Chen et al.'s lower bound from CircuitSAT is more complicated than necessary.

We prove this $\Omega(m)$ separation under Conjecture 1 about CircuitSAT in Section 4, under Conjecture 2 about Online $(\min, +)$ Product in Section 5, and finally under Conjecture 3 about 3SUM in Section 6.

2 Definitions

In this section, we introduce a simple formalism for data structures and their problem/interface specifications (ADTs) that lets us to define general transformations on those specifications. This enables formal definitions of "a retroactive version of a data structure" as well as our Lazy transformation, which in turn help us state our results precisely.

▶ **Definition 1** (ADT). An ***abstract data type*** (***ADT***) $\mathcal{A}$ consists of

- A set $\mathcal{U}$ of ***update*** operations, where each $u \in \mathcal{U}$ takes an argument in domain(u) but does not return a value. (Instead, each update influences the results of future queries.)
- A set $\mathcal{Q}$ of ***query*** operations, which each $q \in \mathcal{Q}$ takes an argument in domain(q) and returns a value in codomain(q). (Queries cannot affect the result of future queries.)
- A partial function $\mathcal{A}$ specifying the result of a query operation given the sequence of prior calls to update operations. That is, given a sequence $\hat{u} = [u_1(x_1), \ldots, u_k(x_k)]$ of calls to update operations and a call to a query operation $q(x)$, the ADT may specify the result $\mathcal{A}(\hat{u}, q(x))$ of $q(x)$ after calling precisely the operations in $\hat{u}$. Note that the result of $q(x)$ can depend on prior update operations, but not on prior query operations. Some behavior may be left ***undefined*** (meaning that any result is valid).[2]

For instance, the Dictionary ADT has two update operations Insert and Delete, and one query operation Contains, all of which take an integer as an argument. The desired behavior is that Contains(x) returns True if Insert(x) has been called more recently than Delete(x), and False otherwise. This ADT represents a set which starts empty, and can have elements inserted and deleted.

▶ **Definition 2** (Data structure, implementation). A ***data structure*** consists of some internal storage (e.g., a pointer machine or word RAM), and some algorithms acting on this internal storage, starting from a specified initial state. A data structure ***implements*** an ADT $\mathcal{A}$ if the algorithms are labelled with the operations of $\mathcal{A}$, and have the correct behavior: suppose we run a sequence of operation calls on the data structure, ending in a query $q(x)$. Let $\hat{u}$ be the sequence of these operations that are updates. Then the final query must return $\mathcal{A}(\hat{u}, q(x))$, if this value is defined.

An ADT may have many different implementations; for example, the Dictionary ADT can be implemented by hash tables or balanced search trees. The usual goal of data structure design is to invent faster implementations for a given ADT.

2.1 Retroactivity

Now we can define partial and full retroactivity as transformations on ADTs.

▶ **Definition 3** (Partially retroactive). Let $\mathcal{A}$ be an ADT. We define an ADT called ***partially retroactive*** $\mathcal{A}$, written Partial-Retro$(\mathcal{A})$. It has the following operations:

[2] For convenience in future ADT definitions, we allow a data structure to output anything when behavior is not defined, but in all of our results it is easy to test for undefined behavior, and thus one could modify the ADTs we define to have no undefined behavior by defining a "default" return value, without affecting our results.

- For each update operation u of $\mathcal{A}$, an update operation $\textsc{Insert}_u$ with domain $\mathbb{N} \times \text{domain}(u)$.
- An update operation $\textsc{Delete}$ with domain $\mathbb{N}$.
- The same query operations as $\mathcal{A}$.

To describe the expected behavior, we imagine that a data structure maintains a sequence $\hat{u}$ of calls to update operations of $\mathcal{A}$ (the "timeline"). Then $\textsc{Insert}_u(k, x)$ means we insert $u(x)$ at position k in this list, and $\textsc{Delete}(k)$ means we remove the call at position k. A call to query operation $q(x)$ in $\mathsf{Partial\text{-}Retro}(\mathcal{A})$ should return $\mathcal{A}(\hat{u}, q(x))$ if that value is defined; that is, we evaluate $q(x)$ at the end of the sequence of updates, which is thought of as the "present".

▶ **Definition 4** ($\mathsf{Fully\ retroactive}$)**.** Let $\mathcal{A}$ be an ADT. We define an ADT called ***fully retroactive*** $\boldsymbol{\mathcal{A}}$, written $\mathsf{Full\text{-}Retro}(\mathcal{A})$. It has the following operations:

- For each update operation u of $\mathcal{A}$, an update operation $\textsc{Insert}_u$ with domain $\mathbb{N} \times \text{domain}(u)$.
- An update operation $\textsc{Delete}$ with domain $\mathbb{N}$.
- For each query operation q of $\mathcal{A}$, a query operation $\textsc{Query}_q$ with domain $\mathbb{N} \times \text{domain}(q)$ and the same codomain as q.

As with partially retroactive $\mathcal{A}$, we imagine that a data structure maintains a sequence $\hat{u}$ of calls to update operations of $\mathcal{A}$. The meanings of $\textsc{Insert}_u$ and $\textsc{Delete}$ are the same as above. A call $\textsc{Query}_q(k, x)$ in $\mathsf{Full\text{-}Retro}(\mathcal{A})$ should return $\mathcal{A}(\hat{u}_{1\cdots k}, q(x))$ if that value is defined, where $\hat{u}_{1\cdots k}$ is the first k elements of $\hat{u}$; that is, it should give the result of calling $q(x)$ after time k in the timeline.

▶ **Lemma 5** ([5, Theorem 1])**.** *Suppose $\mathcal{A}$ has an implementation in which each operation takes $O(T)$ time. Then $\mathsf{Partial\text{-}Retro}(\mathcal{A})$ and $\mathsf{Full\text{-}Retro}(\mathcal{A})$ have implementations in which each operation takes $O(m\,T)$ time, where m denotes the number of calls to update operations in the timeline. Furthermore, the $\mathsf{Partial\text{-}Retro}(\mathcal{A})$ implementation can support query operations in the same time bound as $\mathcal{A}$.*

Proof. Both partial and full retroactivity can be achieved by the simple rollback method in which all operations and the changes they make to the data structure's internal storage get recorded in a stack. These operations can then be "rolled back", undoing their effect; the new update operation applied at the appropriate location; and then all rolled back operations re-applied. ◀

2.2 Complexity Assumptions

Our results rely on the same computational complexity assumptions used by Chen et al. [4] to show their gap between partial and full retroactivity. These assumptions are implied by (i.e. are weaker than) the three main conjectures used in fine-grained complexity – SETH, the APSP Conjecture, and the 3SUM Conjecture [14, 4] – respectively.

▶ **Conjecture 1.** Time $2^{n-o(n)}$ is required to solve $\boldsymbol{SIZE(2^{o(n)})}$ ***CircuitSAT***: given an n-input circuit of size $2^{o(n)}$, decide whether it is satisfiable.

Boolean circuits are more general structures than Boolean formulas, making this conjecture weaker, and thus more believable than, the foundational Strong Exponential Time Hypothesis (SETH) [13].

▶ **Conjecture 2.** Time $n^{3-o(1)}$ is required to solve ***Online* (min, +) *Product***: given an $n \times n$ integer matrix A, and n vectors $v_1, \ldots, v_n$ which are revealed one by one, compute each $(\min, +)$ product $A \diamond v_i$. The next vector v_{i+1} is revealed after outputting $A \diamond v_i$. The $(\min, +)$ product $A \diamond v$ is an n-component vector whose jth component is defined as

$$(A \diamond v)_j = \min_{k=1}^{n} (A_{j,k} + v_k).$$

The offline (and thus easier) version of Online $(\min, +)$ Product [12] is equivalent to the well-known all-pairs shortest path problem, which is commonly assumed hard in fine-grained complexity. Online $(\min, +)$ Product is also a generalization of Online Boolean Matrix–Vector Product, another problem commonly used in fine-grained complexity and not currently known to be equivalent to APSP [12].

▶ **Conjecture 3.** Time $n^{2-o(1)}$ is required to solve ***3SUM***: given three size-n sets A, B, and C of integers, decide whether there exists $(a, b, c) \in A \times B \times C$ such that $a + b + c = 0$.

The 3SUM conjecture was one of the first and remains one of the main conjectures in fine-grained complexity [9]. The version of 3SUM stated in Conjecture 3 is actually called 3SUM′ in [9], while Section 1.1 states the original 3SUM problem. But there is a simple linear-time reduction between 3SUM and 3SUM′ [9, Theorem 3.1], so these two versions of the 3SUM conjecture are equivalent. We use the version stated in Conjecture 3 in our construction.

3 Transforming Partially Retroactive Transformations into Fully Retroactive Transformations

In this section, we develop the first technique mentioned in Section 1.3. First we define a "lazy" version of an abstract data type (Definition 1) $\mathcal{A}$. Roughly speaking, $\mathsf{Lazy}(\mathcal{A})$ makes the queries of $\mathcal{A}$ record their value but not return anything, and adds a new query operation EVALUATE to extract the recorded result of a query. This transformation allows us to simulate full retroactivity using partial retroactivity: insert a "query" that is actually an update operation at some point in the past, and then call EVALUATE in the present to read its result. This is the idea behind the main result of this section, Theorem 8.

▶ **Definition 6** (Lazy)**.** Let $\mathcal{A}$ be an ADT. We define an ADT called ***lazy* $\mathcal{A}$**, written $\mathsf{Lazy}(\mathcal{A})$. It has the following operations:

- The same update operations as $\mathcal{A}$.
- For each query operation q of $\mathcal{A}$, an *update* operation $q^\dagger$ with the same domain as q.
- A query operation EVALUATE with domain $\mathbb{N}$ and codomain $\bigsqcup_{\text{query } q \text{ of } \mathcal{A}} \text{codomain}(q)$.

Suppose $\hat{u}^\dagger$ is the sequence of calls to update operations of $\mathsf{Lazy}(\mathcal{A})$, and suppose the kth call $u_k^\dagger$ is a modified query $q^\dagger(x)$. Then EVALUATE(k) is supposed to return what $q(x)$ would return in $\mathcal{A}$ in the corresponding sequence of calls. Formally, let $\hat{u}_{1\ldots k}$ be the sequence of update calls not of the form $q^\dagger(x)$ from the first k elements of $\hat{u}^\dagger$, so $\hat{u}_{1\ldots k}$ is a sequence of calls to update operations of $\mathcal{A}$. Then $\mathsf{Lazy}(\mathcal{A})(\hat{u}^\dagger, \text{EVALUATE}(k)) = \mathcal{A}(\hat{u}_{1\ldots k}, q(x))$, provided this is defined. If the kth call $u_k^\dagger$ is not of the form $q^\dagger(x)$, then the result of EVALUATE(k) is undefined.

▶ **Lemma 7.** *Suppose a list of m items can be maintained subject to looking up the ith item, and appending or removing a new item at the end, in $O(\ell(m))$ time per operation. (For word-RAM machines, $\ell(m) = 1$; for pointer machines, $\ell(m) = \log m$.)*

Suppose $\mathcal{A}$ has an implementation in which each operation f takes $O(T_f)$ time. Then Lazy($\mathcal{A}$) *has an implementation in which* EVALUATE *takes $O(\ell(m))$ time, and each other operation f or $f^\dagger$ takes $O(T_f + \ell(m))$ time, where m is the total number of calls to operations.*

Proof. To achieve Lazy($\mathcal{A}$) with the desired running times, we create an auxiliary list to store results of queries. For a modified query call $q^\dagger(x)$ occurring as the ith update call, we store the corresponding result of $q(x)$ in position i. Then EVALUATE(i) can simply look up the value stored at position i. The $O(\ell(m))$ overhead comes from maintaining and accessing this list. ◀

Composing Lemmas 5 and 7, we obtain an implementation of Partial-Retro(Lazy($\mathcal{A}$)) in which EVALUATE takes time $O(\ell(m))$, and each other operation takes $O(m(T + \ell(m))$ time. We now show how a partially retroactive version of Lazy($\mathcal{A}$) is able to efficiently simulate a fully retroactive version of $\mathcal{A}$. The key idea is that the lazy version of a data structure has effectively converted the queries into update operations which are then allowed to be inserted at different times in the partially retroactive model.

▶ **Theorem 8.** *If* Partial-Retro(Lazy($\mathcal{A}$)) *has an implementation in which each operation takes amortized time $O(T)$, then so does* Full-Retro($\mathcal{A}$).

Proof. We will use the implementation $\mathcal{D}$ of Partial-Retro(Lazy($\mathcal{A}$)) to implement INSERT$_u$, DELETE, and QUERY$_q$ with constant overhead. The operations available to $\mathcal{D}$ are INSERT$_u$, INSERT$_{q^\dagger}$, DELETE, and EVALUATE. The collision of notation will not be a problem because the functions that share a name are handled by calling their namesake.

Calls to INSERT$_u$ and DELETE can simply be run on $\mathcal{D}$. On QUERY$_q(k, x)$,

1. Call INSERT$_{q^\dagger}(k + 1, x)$.
2. Call EVALUATE$(k + 1)$.
3. Call DELETE$(k + 1)$.
4. Output the result from step 2.

This new data structure uses $\mathcal{D}$ to maintain a list $\hat{u}^\dagger$ of calls to update operations of Lazy($\mathcal{A}$), but we make sure to immediately remove any calls to $q^\dagger$. Thus it equivalently maintains the list $\hat{u}$ of update calls to $\mathcal{A}$.

When we call QUERY$_q(k, x)$, we insert the operation $q^\dagger(x)$ into $\hat{u}^\dagger$ at position $k + 1$, and then use EVALUATE to extract the result of the corresponding $q(x)$ in $\hat{u}$. More formally, the result is Lazy($\mathcal{A}$)($\hat{u}^\dagger$, EVALUATE$(k + 1)$) (where $\hat{u}^\dagger$ is taken in the middle of the call, so it includes the $q^\dagger$), which by the definition of Lazy is $\mathcal{A}(\hat{u}_{1\cdots k}, q(x))$, as desired. ◀

This result allows us to transform a separation between $\mathcal{A}$ and Full-Retro(ADT) into a separation between Lazy($\mathcal{A}$) and Partial-Retro(Lazy($\mathcal{A}$)). In particular, suppose $\mathcal{A}$ has an $O(f)$ implementation but Full-Retro($\mathcal{A}$) requires $\Omega(g)$. Then by Lemma 7, Lazy($\mathcal{A}$) is $O(f + \ell(m))$, and by the contrapositive of Theorem 8, Partial-Retro(Lazy($\mathcal{A}$)) requires $\Omega(g)$. That is, a slowdown for fully retroactive $\mathcal{A}$ implies a slowdown for partially retroactive Lazy($\mathcal{A}$). If $\ell(m) = 1$ such as for the word-RAM model, this slowdown is as large as the original.

Assuming any one of Conjectures 1–3, Chen et al. [4] define an ADT $\mathcal{A}$ with an $O(f)$ implementation and prove a lower bound of $O(m^{1/2-o(1)}f)$ on implementing Full-Retro($\mathcal{A}$). Assuming $\ell(m) = m^{o(1)}$, the generic result of Theorem 8 implies that Lazy($\mathcal{A}$) has an $O(f)$-time implementation but Partial-Retro(Lazy($\mathcal{A}$)) requires $\Omega(m^{1/2-o(1)}f)$, giving an $\Omega(m^{1/2-o(1)})$ separation between an ADT and its partially retroactive version under each of the three assumptions. In Sections 4, 5, and 6, we improve the separation to $m^{1-o(1)}$ when assuming Conjecture 1, 2, and 3 respectively.

4 Lower Bound from SIZE($2^{o(n)}$) CircuitSAT

In this section, we prove a conditional gap for partial retroactivity using the ADT ***Circuit Counter***, which has the following operations and behavior:

- INITIALIZE(C): given a description of a circuit C of size $r = 2^{o(n)}$ which takes n inputs, remember C. This can only be called once, and must be the first operation (otherwise the behavior is undefined). We assume $r > n$ for convenience.
- SET(x): given an n-bit string x, set the ***current string*** to x.
- INCREMENT(): increment the current string as a binary number.
- QUERY(): output True if *any* past value of the current string satisfies C, and False otherwise.

▶ **Lemma 9.** *There is an implementation of Circuit Counter in which each operation takes $2^{o(n)}$ time.*

Proof. The implementation will maintain what the current response to QUERY would be; initially this is False. The rest is straightforward:

- INITIALIZE(C) simply records C, taking time $O(r) = 2^{o(n)}$.
- SET(x) sets the current string to x, and evaluates C on it. If C is satisfied, set the response to True. The runtime is dominated by evaluating C, which takes time $O(r)$.
- INCREMENT() increments the current string, evaluates C on it, and sets the response to True if C is satisfied. This also takes time $O(r)$.
- QUERY() returns the current response, in time $O(1)$. ◀

▶ **Theorem 10.** *There is a sequence of $\Theta(2^{n/2})$ operation calls for Partial-Retro(Circuit Counter) such that, assuming Conjecture 1, any implementation requires $2^{n-o(n)}$ total time, for an amortized lower bound of $n^{n/2-o(n)}$ time per operation.*

Proof. We will use a partially retroactive Circuit Counter to solve SIZE($2^{o(n)}$) CircuitSAT. Given a circuit C of size $r = 2^{o(n)}$, we run the following sequence of operations:

- $\text{INSERT}_{\text{INITIALIZE}}(1, C)$
- $2^{n/2}$ copies of $\text{INSERT}_{\text{INCREMENT}}(2)$
- For each binary string y of length $n/2$:
  - $\text{INSERT}_{\text{SET}}(2, y0^{n/2})$
  - QUERY()
  - DELETE(2)

After the first three steps, the sequence of calls to the Circuit Counter is INITIALIZE(C) and then $2^{n/2}$ copies of INCREMENT(). For each y, we insert SET($y0^{n/2}$) (i.e. y followed by $n/2$ zero bits) right after the INITIALIZE, and query at the end. Since the INCREMENTs count through all strings starting with y, the QUERY returns True if and only if any such string satisfies C. We then clean up the operation inserted before the next loop.

After all $2^{n/2}$ loops, we have tested every possible input to C: thus C is satisfiable if and only if any call to QUERY returned True. Conjecture 1 says that this whole algorithm must take $2^{n-o(n)}$ time. ◀

The implementation from Lemma 9 uses $O(2^{o(n)})$ time per operation and we have $m = \Theta(2^{n/2})$ operations, so the gap here is $2^{n/2-o(n)}/2^{o(n)} = 2^{n/2-o(n)} = m^{1-o(1)}$.

5 Lower Bound from Online $(\min, +)$ Product

In this section, we prove a conditional gap for partial retroactivity using the ADT **$(\min, +)$ *Multiplier***, which has the following operations and behavior:

- SET-A(v): given an n-component row vector v, save it internally as a.
- SET-B(v): given an n-component column vector v, save it internally as b.
- QUERY(): output the list of $a \diamond b$ for all historical values of the pair (a, b). Here $a \diamond b$ is the $(\min, +)$ product $\min_i(a_i + b_i)$.

▶ **Lemma 11.** *There is an implementation of* $(\min, +)$ *Multiplier in which* SET-A *and* SET-B *take* $O(n)$ *time, and* QUERY *takes* $O(m)$ *time, where* m *is the number of calls to update operations.*

Proof. This is straightforward; we maintain the list of what QUERY should output:

- SET-A(v) records v as a, computes $a \diamond b$, and appends it to the list. This takes time $O(n)$.
- SET-B(v) records v as b, computes $a \diamond b$, and appends it to the list. This also takes time $O(n)$.
- QUERY() returns the list, which takes time $O(m)$ because it has length $O(m)$. ◀

▶ **Theorem 12.** *There is a sequence of* $\Theta(n)$ *operation calls for* *Partial-Retro*$((\min, +)$ *Multiplier*$)$ *such that, assuming Conjecture 2, any implementation requires* $n^{3-o(1)}$ *total time, for an amortized lower bound of* $n^{2-o(1)}$ *time per operation.*

Proof. We will use a partially retroactive $(\min, +)$ Multiplier to solve Online $(\min, +)$ Product. Given the $n \times n$ matrix A and the n vectors v_i one by one, we run the following sequence of operations:

- For each row A_j of A:
  - $\text{INSERT}_{\text{SET-A}}(j, A_j)$
- For each v_i:
  - $\text{INSERT}_{\text{SET-B}}(1, v_i)$
  - QUERY(), and output the result
  - DELETE(1)

The first loop just sets up a sequence of n SET-A operations for the rows of A. The second loop inserts SET-B(v_i) at the beginning. Now the historical values of (a, b) are precisely (A_j, v_i) for each row A_j, so the result of the QUERY is $A \diamond v_i$. Finally it removes the SET-B to prepare for the next iteration.

Conjecture 2 says that this whole algorithm must take $n^{3-o(1)}$ time. ◀

The sequence of operations in Theorem 12 has $m = \Theta(n)$ operations, in which case all operation running times from Lemma 11 are $O(n)$. So the gap is $n^{1-o(1)} = m^{1-o(1)}$.

6 Lower Bound from 3SUM

In this section, we prove a conditional gap for partial retroactivity using the ADT ***3-Summer***, which has the following operations and behavior:

- SET-A(L): given a length-$\sqrt{n}$ list L, save it internally as A.
- SET-B(L): given a length-$\sqrt{n}$ list L, save it internally as B.
- SET-C(L): given a length-n list L, save it internally as C.
- QUERY(): return True if at any time, there has been a triple $(a, b, c) \in A \times B \times C$ satisfying $a + b + c = 0$. We call such a triple ***good***.

Note that the internal lists A and B have length $\sqrt{n}$ while C has length n.

▶ **Lemma 13.** *There is an implementation of 3-Summer supporting each* SET *operation in $O(n)$ time and* QUERY *in $O(1)$ time.*

Proof. This is a bit more complicated than the implementations from the previous two sections. As usual, we maintain the value that QUERY should return. The key fact is that we can test whether there is *currently* a good triple in time $O(n)$: make a hash table containing the values of C, then iterate through all n values of $(a, b) \in A \times B$, and check whether $-a-b$ is in the hash table. The rest is straightforward:

- SET-A(L) records L as A, checks if there's a good triple, and sets the query value to True if so. The runtime is dominated by checking for a good triple, which takes time $O(n)$.
- SET-B(L) records L as B, and then proceeds just like SET-A.
- SET-C(L) records L as C, and checks for a good triple. Both steps take $O(n)$ time.
- QUERY() returns the current query value, in time $o(1)$. ◀

▶ **Theorem 14.** *There is a sequence of $\Theta(\sqrt{n})$ operation calls for Partial-Retro(3-Summer) such that, assuming Conjecture 3, any implementation requires $n^{2-o(1)}$ total time, for an amortized lower bound of $n^{3/2-o(1)}$ time per operation.*

Proof. We will use a partially retroactive 3-Summer to solve 3SUM. Given the lists A, B, and C of length n, we first divide A into $\sqrt{n}$ lists $A_1, \ldots, A_{\sqrt{n}}$ of length $\sqrt{n}$, and similarly for B.[3] Now we run the following sequence of operations on the partially retroactive 3-Summer:

- $\text{INSERT}_{\text{SET-C}}(C)$
- For j from 1 to $\sqrt{n}$:
  - $\text{INSERT}_{\text{SET-B}}(j+1, B_j)$
- For i from 1 to $\sqrt{n}$:
  - $\text{INSERT}_{\text{SET-A}}(2, A_i)$
  - QUERY()
  - DELETE(2)

The first two steps set up operations to initialize C, and then set the internal list for B to each section of B in order. Then, for each section A_i of A, we insert SET-A(A_i) right after the SET-C. Now the historical values of (A, B) are (A_i, B_j) varying over j, so the QUERY returns True if and only if there is a good triple $(a, b, c) \in A_i \times B \times C$. Finally, we remove the SET-A to prepare for the next iteration.

Through the second loop, we check each section of A for a good pair. Ultimately, at least one of the calls to QUERY returns True if and only if there is a good pair $(a, b, c) \in A \times B \times C$. Conjecture 3 says that this whole algorithm must take $n^{2-o(1)}$ time. ◀

The gap between the retroactive lower bound in Theorem 14 and the nonretroactive upper bound in Lemma 13 is $n^{1/2-o(1)}$. Because the sequence of operations in Theorem 14 uses $m = \Theta(\sqrt{n})$ operations, this gap is again $m^{1-o(1)}$.

7 Conclusion

This paper gives compelling evidence that there is no general way to make certain data structures retroactive with significantly better running time than the naive rollback method. This worst-case impossibility motivates the question of which problems *can* be made retroactive with low overhead (say, a polylogarithmic factor), such as deques [5], priority queues [5, 6], predecessor [10], and data structures whose updates all commute[5].

[3] For simplicity we're assuming n is a perfect square; otherwise one can take ceilings as appropriate.

On the other side, the L-evaluation framework of Chen et al. [4] appears to be a fairly general method for taking a problem with a computational lower bound and using it to generate a gap between partial and full retroactivity. Perhaps the methods in that paper and this one can be generalized to understand in a broader sense what makes data structures resistant to being made retroactive. This perspective may also be useful in the other direction, granting some intuition about what makes algorithmic problems computationally difficult.

Our applications to the computational cost of time travel fail to account for the physical world's ability to perform quantum computation. It would be interesting to extend our results to obtain similar separations for quantum models of computation, where in particular search can be sped up by Grover's algorithm. See [3] for quantum analogs to some of our fine-grained assumptions.

References

1 Scott Aaronson, Mohammad Bavarian, and Giulio G. Giusteri. Computability theory of closed timelike curves. Technical Report TR16-146, Electronic Colloquium on Computational Complexity, 2016. URL: `https://eccc.weizmann.ac.il/report/2016/146`, `arXiv:TR16-146`.

2 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2007.

3 Harry Buhrman, Subhasree Patro, and Florian Speelman. A framework of quantum Strong Exponential-Time Hypotheses. In Markus Bläser and Benjamin Monmege, editors, *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science*, volume 187 of *LIPIcs*, pages 19:1–19:19, 2021. `doi:10.4230/LIPIcs.STACS.2021.19`.

4 Lijie Chen, Erik D. Demaine, Yuzhou Gu, Virginia Vassilevska Williams, Yinzhan Xu, and Yuancheng Yu. Nearly optimal separation between partially and fully retroactive data structures. In *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 33:1–33:12, Malmö, Sweden, June 2018.

5 Erik D. Demaine, John Iacono, and Stefan Langerman. Retroactive data structures. *ACM Transactions on Algorithms*, 3(2): Article 13, May 2007.

6 Erik D. Demaine, Tim Kaler, Quanquan Liu, Aaron Sidf ord, and Adam Yedidia. Polylogarithmic fully retroactive priority queues via hierarchical checkpointing. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures*, pages 263–275, Victoria, Canada, August 2015.

7 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

8 Ari Freund. Improved subquadratic 3SUM. *Algorithmica*, 77(2):440–458, 2017. `doi:10.1007/s00453-015-0079-6`.

9 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, 1995.

10 Yoav Giora and Haim Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Transactions on Algorithms*, 5(3): Article 28, July 2009. `doi:10.1145/1541885.1541889`.

11 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *Journal of the ACM*, 65(4):22:1–22:25, 2018. `doi:10.1145/3185378`.

12 Monika Henzinger, Sebastian Krinninger, Danupon Na Nongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual Symposium on the Theory of Computing*, pages 21–30, Portland, OR, June 2015.

13 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

14 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians*, pages 3447–3487, Rio de Janeiro, Brazil, 2018. World Scientific.

Minimizing the Maximum Flow Time in the Online Food Delivery Problem

Xiangyu Guo
University at Buffalo, NY, USA

Kelin Luo
Institute of Computer Science, Universität Bonn, Germany

Shi Li
University at Buffalo, NY, USA

Yuhao Zhang
Shanghai Jiao Tong University, China

Abstract

We study a common delivery problem encountered in nowadays online food-ordering platforms: Customers order dishes online, and the restaurant delivers the food after receiving the order. Specifically, we study a problem where k vehicles of capacity c are serving a set of requests ordering food from one restaurant. After a request arrives, it can be served by a vehicle moving from the restaurant to its delivery location. We are interested in serving all requests while minimizing the maximum flow-time, i.e., the maximum time length a customer waits to receive his/her food after submitting the order.

We show that the problem is hard in both offline and online settings even when $k = 1$ and $c = \infty$: There is a hardness of approximation of $\Omega(n)$ for the offline problem, and a lower bound of $\Omega(n)$ on the competitive ratio of any online algorithm, where n is number of points in the metric.

We circumvent the strong negative results in two directions. Our main result is an $O(1)$-competitive online algorithm for the uncapacitated (i.e, $c = \infty$) food delivery problem on tree metrics; we also have negative result showing that the condition $c = \infty$ is needed. Then we explore the speed-augmentation model where our online algorithm is allowed to use vehicles with faster speed. We show that a moderate speeding factor leads to a constant competitive ratio, and we prove a tight trade-off between the speeding factor and the competitive ratio.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithm, Capacitated Vehicle Routing, Flow Time Optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.33

Related Version *Full Version*: https://arxiv.org/abs/2110.15772

Funding *Kelin Luo*: Supported by the National Natural Science Foundation of China, Grant No. 72071157.
Yuhao Zhang: Supported by the National Natural Science Foundation of China, Grant No. 62102251.

1 Introduction

Online food-ordering-and-delivery services (e.g., UberEats and Doordash) have become more and more popular in the past decade. Customers can order dishes online and wait for the restaurant to deliver the food to their home. Arguably one of the most important factors for service quality is the *flow time*, i.e., how long a customer waits to receive the food after submitting his/her order. We formalize the problem as the following *Online Food Delivery Problem* (Online FDP). Let (V, d) be a metric space with $|V| = n$, where d is a metric on V. Let $o \in V$ be the *depot* that has k unit-speed vehicles, each with a capacity $c \in \mathbb{Z}_{>0} \cup \{\infty\}$. The requests arrive online, where a request $\rho = (r_\rho, v_\rho)$ is released at time r_ρ with delivery

location v_ρ. To serve ρ, a vehicle needs to first pick up the ordered food at the depot (after ρ is released) and deliver it to v_ρ. The goal of the online FDP is to schedule vehicles to deliver the food to their delivery locations (i.e., to serve the requests), satisfying the requirement that a vehicle can only carry food for at most c requests at any time. The objective of the problem is to minimize the maximum flow time. This model captures the real-world scenario that one restaurant owns several vehicles and needs to deliver food to customers, and the restaurant wants to make all customers satisfied, i.e, wait for a short period of time.

The *offline* food delivery problem (FDP) is closely related to many vehicle routing problems. For example, the uncapacitated (i.e., $c = \infty$) single-vehicle (i.e., $k = 1$) FDP with all requests released at time 0 already captures a variant of the traveling salesman path problem. When the vehicle has finite capacity c, the FDP just mentioned is also known as the Capacitated Vehicle Routing Problem (CVRP). A more general setting is studied under the name Dial-a-Ride Problem (DaRP), where besides the delivery location, each customer can specify its own pickup location. To satisfy the request, a vehicle has to take the customer from the pick-up location to its delivery location. Most results on offline CVRP/DaRP focused on the total travel distance objective [1, 24, 12, 23]; while in the online model, much effort has been devoted to completion time objectives like average completion time [25, 9] or makespan [19, 2]. Compared with previous studies, we are focusing on a much harder and less-studied objective – maximum flow time, in the online setting. The only theoretical results we know regarding maximum flow time are two lower bounds for online TSP and online DaRP: Krumke et al.[33] show that the single-vehicle *finite-capacity* DaRP does not admit $O(1)$-competitive algorithms, and a similar lower bound holds for online TSP, even on line metrics [35]. But the lower bound does not hold for FDP, which is *not* a generalization of TSP in the online case.

Another motivation for FDP is from the seemingly unrelated broadcast scheduling problem. In this problem, a server holds n pages of varying sizes and requests are released over time, each of which is a query on one of the pages. The server can *broadcast* a page to all requests on that same page, which takes time equal to the page size. The objective is to minimize the maximum flow time. It is easy to reduce the broadcast scheduling problem to uncapacitated single-vehicle FDP on a star, where a page of size s in the broadcast scheduling problem corresponds to an edge of length $s/2$ in the FDP problem.[1] It is known that FIFO gives $O(1)$-competitiveness for the broadcast scheduling problem [13, 11], and the single-vehicle uncapacitated FDP on stars [22]. One of our main results in the paper is an $O(1)$-competitive algorithm for uncapacitated FDP on *trees*, generalizing the results to tree metrics and the multiple-vehicle case. Mapping to the broadcast scheduling application, this gives the following more general setting. There is a tree rooted at o, where the pages correspond to the leaves of a tree, and each internal node corresponds to a setup procedure that takes some time to run. To broadcast a set of pages, in addition to the broadcasting time for each page, we have to spend time on running the setup procedures correspondent to the ancestor nodes of the pages.

1.1 Our Results

We state our results in this section. The formal definition of the offline and online food delivery problem (FDP) can be found in Section 2. In all the theorems below, n is the number of points in the metric space.

[1] In FDP, we could define the completion time of a request as the time the vehicle returns to the depot after serving the request. The maximum flow time objective for the two versions differs by an $O(1)$ factor.

We first show that our problem also suffers from the maximum flow time objective, both in offline and online settings, even if we focus on the single-vehicle uncapacitated case.

▶ **Theorem 1.** *The following statements hold for the single-vehicle uncapacitated FDP:*
(1a) *It is NP-hard to approximate the offline problem within a factor of $o(n)$.*
(1b) *There is no $o(n)$-competitive algorithm for the online problem, even if it can run in exponential time and the metric is a bounded-pathwidth planar graph.*

Our Results for Tree Metrics. Given that (1b) holds even for bounded-pathwidth planar graphs, a natural candidate family of metrics is the tree metrics. Our first algorithmic result of the paper is that uncapacitated FDP on trees admit $O(1)$-competitive online algorithms:

▶ **Theorem 2** (Tree Metric). *There's an efficient $O(1)$-competitive online algorithm for uncapacitated FDP on trees.*

The results in Theorem 1 and 2 are for uncapacitated version of the problem ($c = \infty$). We remark that these results should be contrasted with the lower bound given by Krumke et al.[35], which says that *online* TSP (and thus *uncapacitated* DaRP) does not admit $O(1)$-competitive algorithms *even on line metrics*. Their lower bound however does not apply to (online) uncapacitated FDP, and this is crucially due to the fact that in FDP a vehicle has to return to *the* depot before it can serve a newly released request.

We now turn to the general capacitated case. One can ask if the algorithm in Theorem 2 works for general c. Unfortunately, we show that this is impossible even for a very special case. Below R is the set of requests in the instance:

▶ **Theorem 3.** *There is no $o(\sqrt{|R|})$-competitive online algorithm for single-vehicle FDP with $c = 2$ and on a tree with 5 vertices.*

Our Results with Speed Augmentation. Despite all the negative results, we show that the problem admits good competitive algorithms under the *speed augmentation* model (*in general metric*). In this model we allow vehicles in our algorithm to run faster than normal vehicles, while we compare the maximum flow time achieved by our algorithm using faster vehicles against the optimum maximum flow time that can be achieved with normal vehicles. The speeding factor defines how faster our vehicles are, and our goal is to decide the tradeoff between the speeding factor and the competitive (approximation) ratio of our algorithm. There are two reasons for which we consider this model. First, in the optimum solution of the hard instance for (1b), the vehicle is always busy. One mistake made by the online algorithm can delay the whole schedule. This situation rarely happens in practice, where one should expect the vehicles to have a reasonable amount of idle time. Second, the travel time between two points in practice is often an estimate and affected by many factors such as the driver's skill and the traffic condition. So, a solution that becomes very bad if the speeds of vehicles decrease slightly should be considered as very fragile. We remark that the speed augmentation model is also popular in many scheduling problems.

Let α_{TSP} be the infinum of all values α such that there is a (polynomial time) α-approximation algorithm for TSP. With the breakthrough result of Karlin, Klein, and Oveis Gharan [30], we know that α_{TSP} is strictly smaller than 1.5. Define α_{CVRP} similarly for the CVRP problem. It is known that $\alpha_{\mathsf{CVRP}} \leq \alpha_{\mathsf{TSP}} + 1$. We give our results for the food delivery problem with speeding below.

▶ **Theorem 4** (General Metric). *For any small enough constant $\epsilon > 0$, there are*

(4a) *an exponential time $(1+\epsilon)$-speeding $O(1/\epsilon)$-competitive online algorithm for FDP,*

(4b) *a polynomial time $(\alpha_{\mathsf{TSP}}+\epsilon)$-speeding $O(1/\epsilon)$-competitive online algorithm for uncapacitated FDP, and*

(4c) *a polynomial time $(\alpha_{\mathsf{CVRP}}+\epsilon)$-speeding $O(1/\epsilon)$-competitive online algorithm for (capacitated) FDP.*

▶ **Theorem 5.** *The following statements hold.*

(5a) *There is NO $(1+\epsilon)$-speeding $o(1/\epsilon)$-competitive online algorithm for FDP.*

(5b) *For any constant $\alpha \in [1, \alpha_{\mathsf{TSP}})$, there is NO (polynomial time) α-speeding $o(\sqrt{n})$-approximation algorithm for uncapaciated FDP.*

(5c) *For any constant $\alpha \in [1, \alpha_{\mathsf{CVRP}})$, there is NO (polynomial time) α-speeding $o(\sqrt{n})$-approximation algorithm for (capaciated) FDP.*

Therefore, by (5a), the $O(1/\epsilon)$-dependence on ϵ in (4a) is needed. (5b) and (5c) state that the speeding factors in (4b) and (4c) are almost tight.

1.2 An Overview of Techniques

The hardness results in (1a), (5b) and (5c) are proved using simple reductions from TSP or CVRP, in which we repeat a TSP/CVRP instance multiple times. Since an efficient algorithm can not find the best TSP/CVRP tour, it needs to use a longer tour for each instance, which creates a delay in the schedule. The delays for all instances will accumulate, resulting in a bad flow time. The hardness remains if the speeding factor is not big enough.

The lower bounds in (1b), (5a) and Theorem 3 are proved using the same idea. We build a base instance satisfying the following property. At the beginning of the time horizon, the online algorithm needs to make a decision between two choices. If it made the incorrect choice, the total length of trips it uses to satisfy all requests will be 1 unit time longer than the case if it made the correct choice. The catch is, the algorithm will only know which choice is the correct one until near the end of the time horizon. So, it has to either wait for almost the whole time horizon, which will incur a large maximum flow time, or it will spend 1 more unit time on the trips, delaying the schedule. By repeating the base instance many times, the delay of the online algorithm will accumulate, resulting in a large maximum flow time. Lastly, we remark that although all such lower bounds are proved for deterministic algorithms, it is not hard to extend the proofs to randomized algorithms by directly applying Yao's Minimax Principle: instead of giving the correct choice adversarially, we make it uniformly random.

The overview of the algorithms for online uncapacitated FDP on trees will be given at the beginning of Section 3. The online algorithms in Theorem 4 are based on a simple idea. We wait for γF time units (F is a given upper bound of the optimal flow time. See Section 2 for details.) for some $\gamma = \Theta\left(\frac{1}{\epsilon}\right)$. Then we know that the optimum solution can serve all the requests that arrived in the γF-length interval using $(\gamma + O(1))F$ time. With $(1+\epsilon)$-speeding, the online algorithm can serve them in γF time. Thus, we can always guarantee an $O(F/\epsilon)$-flow time for all requests. If the algorithm has to be efficient, we need to lose a speeding factor of $\alpha_{\mathsf{TSP}} + \epsilon$ or $\alpha_{\mathsf{CVRP}} + \epsilon$, depending on whether the instance is uncapacitated.

1.3 Other related work

Most of previous results on CVRP and DaRP focus on the single vehicle case and the total travel distance objective, which is equivalent to the makespan. Unless specified otherwise, all the results surveyed below for CVRP and DaRP are for this case.

Capacitated Vehicle Routing Problem (CVRP). The CVRP is mostly studided in the offline setting. It admits an $(\alpha_{\mathsf{TSP}}+1)$-approximation, where α_{TSP} is the approximation ratio for traveling salesman problem (TSP) [1, 24, 30]. A more general version of CVRP is studied in the literature: Each request has a demand, and we require the total demand of all requests satisfied in a single trip made by the vehicle is at most c. In the same papers, Altinkemer and Gavish [1] and Haimovich and Rinnooy Kan [24] gave an approximation algorithm with ratio $\alpha_{\mathsf{TSP}}+2$ for the general CVRP, which stood for over 30 years. Very recently, the approximation ratio for the problem has been improved to $\alpha_{\mathsf{TSP}}+2(1-\epsilon)$ by Blauth *et al.* [10], for some small constant $\epsilon>0$. There are also improved ratios on special metrics like Euclidean plane [16, 32], tree metrics [36, 44, 41], or other special graph metrics [15, 27].

One variant of CVRP that is related to the flow time objective is the *CVRP with bounded delay*: There is a *delay constraint* that any request ρ spends at most $\beta d(r, v_\rho)$ time *on the vehicle*, where v_ρ denotes the drop-off location of ρ. The goal is to find a minimum length route that satisfies all the delay constraints. For this variant, Gørtz *et al.* [21] gave a $\left(2.5+\frac{3}{\beta-1}\right)$-approximation algorithm for single-vehicle CVRP with bounded delay.

Dial-a-Ride Problem (DaRP). Like CVRP, the most studied setting for offline DaRP is also for the single-vehicle case with makespan objective, for which the best known algorithm achieves $\tilde{O}(\min\{\sqrt{n},\sqrt{c}\})$ approximation ratio [12, 23], where n is the number of requests. The best known lower bound is only APX-hardness. The *online* DaRP is mostly studied with objectives like makespan [2] or total/weighted completion time [19, 34, 9]: all of these objectives admit small constant competitive-ratio algorithm. However, if we turn to minimizing the maximum flow-time, there exists no $o(n)$-competitive algorithm even on a 3-point uniform metric with a unit-capacity vechile [33].

A variant of DaRP studied in the literature that seems much easier is the preemptive DaRP, in which the vehicle is allowed to temporarily unload the cargo it carries in the middle of a trip, and pick it up later to resume delivery. The best approximation ratio for the problem with single vehicle is $O(\log n)$ [12], and there is a hardness of $\Omega(\log^{1/4} n)$ hardness of approximation [20]. For the multi-vehicle case, Gørtz *et al.*[21] gave an $O(\log^3 n)$ for the preemptive DaRP, and an $O(\log n)$-approximation when vehicles have infinite capacity.

Flow-Time Scheduling. The maximum flow time objective is studied in many scheduling problems. The simple FIFO strategy is known to achieve the best competitive ratio 2 [40] in the identical machine setting, and constant competitive algorithm exists even in related machine setting [6] (i.e., machines have different speed). However, the approximability changes dramatically if we switch the objective to the (weighted) sum of flow-time: if no preemption is allowed, the problem is $\Omega(m)$-hard in the online setting where m is the number of machines, and $\Omega(m^{1/2-\epsilon})$-hard in the offline setting [31]. Even when preemption is allowed, offline $O(1)$-approximations (for the single-machine setting) are known only very recently [8, 18, 43], and there's an $\omega(1)$ lower bound for the online case [3].

For online broadcast scheduling, the maximum flow time objective also appears to be easier than other flow time objectives: max flow time admits 2-competitive algorithm [7, 11, 13], while average flow time has very strong lower bounds $\Omega(n)$ for deterministic algorithms [29] and $\Omega(\sqrt{n})$ for randomized algorithms [5].

Resource Augmention in Flow-Time Scheduling. Due to the strong lower bounds mentioned above for various flow time objectives, people proposed the *resource augmentation model* in order to find provably good algorithms. In a pioneering work, Kalyanasundaram

and Pruhs [28] proposes the speed augmentation model where the algorithm is allowed to use machines faster than the optimal solution, and give the first constant competitive algorithm for minimizing (nonclairvoy) total flow time. Later Phillips *et al.*[42] explores another type of resource – machines: they showed that there is a constant competitive algorithm for non-preemptive weighted flow time scheduling, but uses $m + O(\log P)$ machines (here P is the ratio of length between the longest and the shortest job, and the optimal solution uses only m machines). They also showed that one can combine the two resources and get a constant-competitive algorithm for total flow time, using $O(\log n)$ extra machines with $O(1)$ speeding. Epstein and van Stee [17] gave a ℓ-machine algorithm for the single-machine total flow time scheduling, and achieves an asymtotically optimal $O(\min\{\sqrt[\ell]{P}, \sqrt[\ell]{n}\})$-competitive ratio.

In the offline case, the resource augmentation model is also used to give constant approximation algorithms for *non-preemptive* flow time scheduling: first on a single machine [4], and then extended to multiple machines by Im *et al.*[26]. The resources are not equally powerful, though. Lucarelli *et al.*[39] showed that for the weighted flow time objective, the non-preemptive single-machine scheduling problem does not admit bounded-competitive algorithms *even with arbitrarily faster* machine. Choudhury *et al.*[14] therefore explored another type of "augmentation": allowing rejection of some jobs. This idea is used to obtain the first constant competitive algorithm for weighted flow time scheduling in the *non-preemptive* setting [39, 37, 38].

1.4 Organization

The remaining part of the paper is organized as follows. In Section 2, we formally define the offline and online food delivery problem. We prove our main algorithmic result, Theorem 2, by giving the $O(1)$-competitive algorithm for uncapacitated FDP on trees in Sections 3 and 4. The two sections consider the single-vehicle and multiple-vehicle cases respectively. Due to space limit, all lower bound results and the results with speed augmentation are included only in the full version.

2 Preliminary

We now define the food delivery problem formally, starting from the offline setting. We are given a graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}_{>0}$, where $\ell(e)$ denotes the time needed to traverse the edge e in either direction.[2] There is a special vertex $o \in V$ called the depot, which represents the restaurant. We are given a number $k \geq 1$ of vehicles which are initially located at o, and capacity $c \in \mathbb{Z}_{>0} \cup \{\infty\}$ on the vehicles. When $k = 1$, we say the problem is single-vehicle, and when $c = \infty$, we say the problem is uncapacitated. There is a set R of requests. Each request $\rho \in R$ is denoted by $\rho = (r_\rho, v_\rho)$, where $r_\rho \in \mathbb{R}_{\geq 0}$ and $v_\rho \in V$ are the arrival time and the delivery location of the request respectively.

To describe the output of the problem, we need to define *trips*. A trip is defined by a triple $(t, (u_0 = o, u_1, u_2, \cdots, u_z = o), R')$, where $t \geq 0$ is the starting time of the trip, $(u_0 = o, u_1, u_2, u_3, \cdots, u_{z-1}, u_z = o)$ is a (possibly complex) cycle in G that starts and ends at o, and $R' \subseteq R$, $|R'| \leq c$, is the set of requests served by the trip. So $z \geq 2$ and $(u_{z'-1}, u_{z'}) \in E$ for every $z' \in [z]$. We require the following properties to hold for a trip.

[2] Equivalently we could use a metric (V, d) to describe the travel times, but for many of our results it is more convenient to use the graph G with edge lengths.

First, $u_{z'} \neq o$ for every $z' \in [z-1]$; one can see easily soon that this is without loss of generality. Then, for every served request $\rho \in R'$, we have $r_\rho \leq t$ and v_ρ appears in the cycle. If $\tilde{z}$ is the smallest index such that $u_{\tilde{z}} = v_\rho$, then we say ρ is served by the trip at time $t + \sum_{z'=1}^{\tilde{z}} \ell(u_{z'-1}, u_{z'})$. The completion time of the trip is defined as $t + \sum_{z'=1}^{z} \ell(u_{z'-1}, u_{z'})$. Abusing the definition slightly, sometimes we also use the word "trip" to denote the cycle $(u_0 = o, u_1, u_2, \cdots, u_z = o)$, with t and R' specified separately.

The output of the food delivery problem contains k sequences of trips correspondent to the itineraries of the k vehicles. The following properties need to be satisfied. For every pair of adjacent trips in any of the k sequences, the starting time of the latter trip is at least the completion time of the former one. Moreover, each request in R is served by exactly one trip in the k sequences; in other words, the sets of served requests in all trips of the k sequences form a partition of R. Let t_ρ be the time that a request $\rho \in R$ is served (by the unique trip that serves it). We define the flow time of ρ to be $t_\rho - r_\rho$. The goal of the problem is to minimize the maximum flow time, i.e, $\max_{\rho \in R}(t_\rho - r_\rho)$.

So far we have defined the food delivery problem in the offline setting. In the online setting, G, ℓ, k and c are given upfront, but the requests arrive online. Once we decided to start a trip at time t, then we have to complete the trip as planed. Formally, we require that for any time $t \geq 0$, the prefixes of the k sequences of trips with starting time at or before t can only depend on the requests that arrive at or before t.

Guessing the Optimum Maximum Flow Time Online. When designing online algorithms, we shall use the standard doubling trick to assume that we are given an upper bound F on the optimum maximum flow time of the instance, and the competitive ratio is defined by comparing to F. That is, a β-competitive online algorithm has maximum flow time at most βF. If we have a β-competitive online algorithm $\mathcal{A}$ under this setting, then we can obtain an 8β-competitive algorithm $\mathcal{A}'$ under the setting where F is not given to us, while keeping the speeding factor and running time unchanged. In the paper, we are not trying to optimize the constant, and the factor of 8 will be hidden in the $O(\cdot)$ notation. Due to space limitation, we omit the detail here.

3 Online Uncapacitated Single-Vehicle FDP on Tree Metrics

In this section and the next one, we give the $O(1)$-competitive algorithm for online uncapacitated FDP on tree metrics, proving Theorem 2. In this section, we consider the case $k = 1$, i.e, there is only one vehicle. We separate this case from the general problem as its algorithm is simpler and the competitive ratio we obtain is better.

First we setup some notations here. Let $T = (V, E)$ be the tree and we assume it is rooted at the depot $o \in V$. For every $v \in V$, we use V_v to denote the set of descendants of v (including v itself). For an edge $e = (u, v)$ with v being the child, we define $V_e = V_v$. We use d to denote the metric induced by the tree T with lengths $\ell(\cdot)$. Given a set $X \subseteq V$ of vertices, we define $\mathsf{mst}(X)$ to be the cost of the minimal sub-tree of T containing X and o (Notice that we require the tree to contain o). Suppose we are further given an element $e \in V \cup E$. We define $\mathsf{mst}_e(X) = \mathsf{mst}(V_e \cap X)$, which is the cost of the minimal sub-tree of T containing o and all vertices in $V_e \cap X$. So if $V_e \cap X = \emptyset$, then $\mathsf{mst}_e(X) = 0$; otherwise, the tree contains all the edges from o to e (including e itself if it is an edge). By definition we have $\mathsf{mst}_o(X) = \mathsf{mst}(X)$.

Since most of the time we deal with requests, it is convenient for us to use $\mathsf{mst}(R')$ and $\mathsf{mst}_e(R')$ to denote $\mathsf{mst}(\{v_\rho : \rho \in R'\})$ and $\mathsf{mst}_e(\{v_\rho : \rho \in R'\})$ respectively for a set $R' \subseteq R$ of requests. Abusing notations slightly, sometimes we also use $\mathsf{mst}(X)$ to denote the actual sub-tree of T achieving the cost $\mathsf{mst}(X)$. This also extends to $\mathsf{mst}_e(X), \mathsf{mst}(R')$ and $\mathsf{mst}_e(R')$.

We now overview the algorithm for the online uncapacitated FDP problem on tree metrics, for both the single and multiple-machine cases. We break the time horizon into intervals of length F: $\{[0, F), [F, 2F), [2F, 3F), [3F, 4F), ...\}$ and index them by $1, 2, 3, 4, \cdots$. Recall that F is an upper bound on the optimum maximum flow time, and the competitive ratio of our algorithm is defined against F. Let R_i be the set of requests that arrive in interval i. For each even integer i, we break R_i into R_i^{left} and R_i^{right}, merge R_i^{left} with R_{i-1}, merge R_i^{right} with R_{i+1}, so as to minimize some carefully designed objective. So for each odd integer i, we have constructed a merged set $R'_i := R_{i-1}^{\mathsf{right}} \cup R_i \cup R_{i+1}^{\mathsf{left}}$, which we call a *bundle*. The bundles then can be constructed in an online manner: the bundle R'_i is determined by requests that arrive before time $(i + 2)F$.

Then our online algorithm handles the bundles separately. When $k = 1$, we view each bundle R'_i as a job of size $2\mathsf{mst}(R'_i)$; when $k \geq 2$, we break R'_i into many *groups* and treat each group Q as a job of size $2\mathsf{mst}(Q)$. We think of all these jobs are released at time $(i+2)F$. We then assign the jobs to the vehicles online in a greedy manner. A crucial lemma we show is that the jobs have a small backlog: The jobs released in any interval $[aF, bF]$ have total size at most $(b - a)F + O(1) \cdot kF$, and each job has size $O(F)$. The properties guarantee that all the jobs complete with an $O(F)$ flow time.

3.1 Description of Algorithm for Single-Vehicle ($k = 1$) Case

Now we formally state the algorithm for the case $k = 1$. For every integer $i \geq 1$, let $R_i = \{\rho \in R : (i-1)F \leq r_\rho < iF\}$ as stated. Indeed, once we know a request is in R_i, we do not care about its precise arrival time anymore. In the first step of the online algorithm, we partition the requests into bundles $(R'_i)_{i \geq 1 : i \text{ is odd}}$, using Algorithm 1, where the bundle R'_i is generated at time $(i + 2)F$. For now let us focus on the case $k = 1$ in the algorithm. The formal definition for $\mathsf{cost}(\cdot|\cdot)$ is postponed to section 4 since it's not used here. Throughout the section and the next one, we assume all undefined sets are set to $\emptyset$. See Figure 1(a) for an illustration of the relationships between R_i, R'_i, R_i^{left} and R_i^{right}.

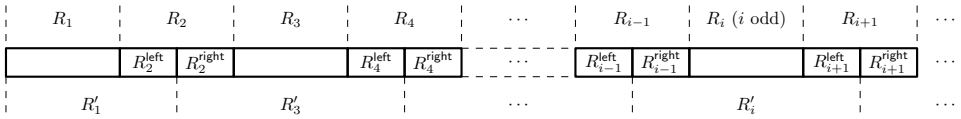
Algorithm 1 Partition of Requests into Bundles for Both Single and Multiple-Vehicle Cases.

1: **for** $i = 2, 4, 6, 8, \cdots$ **do**
2: wait until time $(i + 1)F$
3: let $R_i = \{\rho \in R : r_\rho \in [(i - 1)F, iF)\}$ be as defined in the text
4: partition R_i into R_i^{left} and R_i^{right} so as to minimize

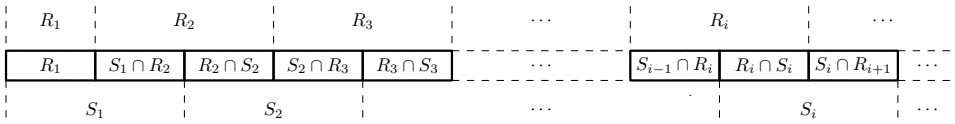
$$\begin{cases} \mathsf{mst}(R_{i-1} \cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}} \cup R_{i+1}) & \textbf{if } k = 1 \\ \mathsf{cost}(R_i^{\mathsf{left}} | R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}} | R_{i+1}) & \textbf{if } k > 1 \end{cases}$$

5: release the bundle $R'_{i-1} := R_{i-2}^{\mathsf{right}} \cup R_{i-1} \cup R_i^{\mathsf{left}}$ (at time $(i + 1)F$)

We briefly talk about the efficiency of the algorithm for $k = 1$. It is easy to see that the following simple strategy will find the partition $(R_i^{\mathsf{left}}, R_i^{\mathsf{right}})$ that minimizes $\mathsf{mst}(R_{i-1} \cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}} \cup R_{i+1})$. For every $\rho \in R_i$, if $d(\rho, \mathsf{mst}(R_{i-1})) \leq d(\rho, \mathsf{mst}(R_{i+1}))$, then we put ρ in R_i^{left}. Otherwise we put ρ in R_i^{right}. Here $d(\rho, \mathsf{mst}(R_{i-1}))$ ($d(\rho, \mathsf{mst}(R_{i+1}))$, resp.) is the shortest distance between r_ρ and any vertex in $\mathsf{mst}(R_{i-1})$) ($\mathsf{mst}(R_{i+1})$, resp.).



(a) Illustration of relationships between R_i's, R_i^{left}'s, R_i^{right}'s and R'_i's. For every even i, we have $R_i = R_i^{\text{left}} \cup R_i^{\text{right}}$. For every odd i, we have $R'_i = R_{i-1}^{\text{right}} \cup R_i \cup R_{i+1}^{\text{left}}$.



(b) Illustration of relationships between R_i's and S_i's. For every i, we have $R_i \subseteq S_{i-1} \cup S_i$ and $S_i \subseteq R_i \cup R_{i+1}$.

Figure 1 Illustration of sets used in the algorithm and analysis for online uncapacitated FDP on tree metrics.

Notice that starting from the depot o, serving all the requests in R'_i and traveling back to the depot take time exactly $2\mathsf{mst}(R'_i)$. Therefore, we can view each bundle R'_i, for an odd $i \geq 1$, as a job of size $2\mathsf{mst}(R'_i)$ released at time $(i+2)F$. We view the vehicle as a machine. In the second step of the online algorithm, we then schedule the jobs (i.e., bundles) on the machine (i.e., the vehicle) in their order of releasing: Whenever the machine is idle and there are available jobs, we process the job that is released the earliest. This finishes the description of the algorithm for the single vehicle case.

3.2 Analysis of Backlog of Jobs

We define the flow time of a job R'_i, for an odd $i \geq 1$, as its completion time minus its release time (which is $(i+2)F$). By the folklore result, if the total size of all jobs released in time $[t, t']$ is at most $t' - t + D$ for some D and for every pair $t \leq t'$ of time points, then every job has flow time at most D in the schedule using greedy algorithm. Therefore, it remains to prove that D is small, which is stated in Lemma 9 later.

Before describing and proving the lemma, we introduce a partition $(S_i)_{i \geq 1}$ of requests that depends on the offline optimum solution, and prove some helper lemmas. Notice that in the offline optimum solution a trip can not serve two requests whose arrival times are more than F apart: If that happens, then the flow time of the earlier request of the two is more than F. Thus, a trip in the optimum solution serves either requests from a single set R_i for some i, or requests from two sets R_i and R_{i+1} for some i. In both the cases, we put all the requests in the trip into the set S_i. Therefore, $S_1, S_2, S_3, \cdots$ form a partition of R. Notice that $S_i \subseteq R_i \cup R_{i+1}$ and $R_i \subseteq S_{i-1} \cup S_i$ for every integer i. See Figure 1(b) for an illustration of the relationship between R_i's and S_i's. Notice that S_i's is only used in our analysis, as they depend on the offline optimum solution, which our algorithm does not know.

We need the following simple lemma and corollaries.

▶ **Lemma 6.** *Let X_1, X_2, X_3 and X_4 be subsets of V and $e \in E \cup V$. Then the following inequalities hold:*

$$\mathsf{mst}_e(X_1 \cup X_2) \leq \mathsf{mst}_e(X_1) + \mathsf{mst}_e(X_2), \tag{1}$$

$$\mathsf{mst}_e(X_2) + \mathsf{mst}_e(X_1 \cup X_2 \cup X_3) \leq \mathsf{mst}_e(X_1 \cup X_2) + \mathsf{mst}_e(X_2 \cup X_3), \tag{2}$$

$$\mathsf{mst}_e(X_1 \cup X_2 \cup X_3) + \mathsf{mst}_e(X_2 \cup X_3 \cup X_4) \leq \mathsf{mst}_e(X_1 \cup X_2) + \mathsf{mst}_e(X_2 \cup X_3) + \mathsf{mst}_e(X_3 \cup X_4). \tag{3}$$

Proof. It suffices to prove the inequalities for $e = o$, since the other cases can be proved by changing X_1, X_2, X_3 and X_4 to $V_e \cap X_1, V_e \cap X_2, V_e \cap X_3$ and $V_e \cap X_4$ respectively.

(1) is easy to see. For (2), we focus on each edge $e' \in E$ and count the number of times $\ell(e')$ is considered on the left and right sides respectively. Notice that $\ell(e')$ is counted in $\mathsf{mst}(Y)$ for some set Y if and only if $V_{e'} \cap Y \neq \emptyset$.

- If $\ell(e')$ is counted 0 times on the left side, then $V_{e'} \cap Y = \emptyset$ for every $Y \in \{X_1, X_2, X_3\}$. Clearly $\ell(e')$ is counted 0 times on the right side.
- If $\ell(e')$ is counted in $\mathsf{mst}(X_1 \cup X_2 \cup X_3)$ but not in $\mathsf{mst}(X_2)$, then either $V_{e'} \cap X_1 \neq \emptyset$ or $V_{e'} \cap X_3 \neq \emptyset$. Then $\ell(e')$ is counted at least once on the right side.
- If $\ell(e')$ is counted in $\mathsf{mst}(X_2)$, then it is counted twice on both the left right sides.

Similarly for the proof of (3), we consider the number of times each $\ell(e')$ is counted on both sides:

- If e' is counted 0 times on the left side, then $V_{e'} \cap Y = \emptyset$ for every $Y \in \{X_1, X_2, X_3, X_4\}$. Thus, $\ell(e')$ is counted 0 times on the right side.
- Assume $\ell(e')$ is counted once on the left side. W.l.o.g assume it is counted in $\mathsf{mst}(X_1 \cup X_2 \cup X_3)$ but not in $\mathsf{mst}(X_2 \cup X_3 \cup X_4)$. Then clearly e' is counted at least once on the right side.
- Finally assume $\ell(e')$ is counted in both $\mathsf{mst}(X_1 \cup X_2 \cup X_3)$ and $\mathsf{mst}(X_2 \cup X_3 \cup X_4)$. If $V_{e'} \cap X_2 \neq \emptyset$ or $V_{e'} \cap X_3 \neq \emptyset$, then $\ell(e')$ is counted at least twice on the right side. Otherwise we have $V_{e'} \cap X_1 \neq \emptyset$ and $V_{e'} \cap X_4 \neq \emptyset$. In this case $\ell(e')$ is counted twice on the right side. ◀

▶ **Corollary 7.** *For every odd $i \geq 1$ and $e \in E \cup V$, we have $\mathsf{mst}_e(R'_i) \leq \mathsf{mst}_e(R^{\mathsf{right}}_{i-1} \cup R_i) + \mathsf{mst}_e(R_i \cup R^{\mathsf{left}}_{i+1}) - \mathsf{mst}_e(R_i)$.*

Proof. Applying (2) with $X_1 = R^{\mathsf{right}}_{i-1}, X_2 = R_i$ and $X_3 = R^{\mathsf{left}}_{i+1}$, and using $X_1 \cup X_2 \cup X_3 = R'_i$ proves the lemma. [3] ◀

▶ **Corollary 8.** *For every integer $i \geq 1$ and $e \in E \cup V$, we have $\mathsf{mst}_e(S_{i-1} \cup R_i) + \mathsf{mst}_e(R_i \cup S_i) - \mathsf{mst}_e(R_i) \leq \mathsf{mst}_e(S_{i-1}) + \mathsf{mst}_e(S_i)$.*

Proof. We apply (3) with $X_1 = R_{i-1} \cap S_{i-1}, X_2 = S_{i-1} \cap R_i, X_3 = R_i \cap S_i$ and $X_4 = S_i \cap R_{i+1}$. The corollary follows by that $X_1 \cup X_2 = S_{i-1}$, $X_2 \cup X_3 = R_i$, $X_3 \cup X_4 = S_i$, $X_1 \cup X_2 \cup X_3 = S_{i-1} \cup R_i$ and $X_2 \cup X_3 \cup X_4 = R_i \cup S_i$. ◀

With the corollaries established, we now state and prove the lemma that bounds the backlog of bundles.

▶ **Lemma 9.** *For any two odd positive integers $a \leq b$, we have $2 \sum_{i \in [a,b]: i \text{ odd}} \mathsf{mst}(R'_i) \leq (b - a + 4)F$.*

Proof.

$$\begin{aligned} &\sum_{i \in [a,b]: i \text{ odd}} \mathsf{mst}(R'_i) \\ &\leq \sum_{i \in [a,b]: i \text{ odd}} \left(\mathsf{mst}(R^{\mathsf{right}}_{i-1} \cup R_i) + \mathsf{mst}(R_i \cup R^{\mathsf{left}}_{i+1}) - \mathsf{mst}(R_i)\right) \qquad \text{(by Corollary 7)} \end{aligned}$$

[3] Recall that in the notation mst_e, we can view requests as vertices by ignoring their arrival times.

$$
\begin{aligned}
&= \sum_{i\in(a,b):i \text{ even}} \Big(\mathsf{mst}(R_{i-1}\cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}}\cup R_{i+1})\Big) - \sum_{i\in[a,b]:i \text{ odd}} \mathsf{mst}(R_i) \\
&\quad + \mathsf{mst}(R_{a-1}^{\mathsf{right}}\cup R_a) + \mathsf{mst}(R_b\cup R_{b+1}^{\mathsf{left}}) \qquad \text{(by reorganizing terms)} \\
&\le \sum_{i\in(a,b):i \text{ even}} \Big(\mathsf{mst}(R_{i-1}\cup S_{i-1}) + \mathsf{mst}(S_i\cup R_{i+1})\Big) - \sum_{i\in[a,b]:i \text{ odd}} \mathsf{mst}(R_i) \\
&\quad + \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \mathsf{mst}(S_{a-1}\cup R_a) + \mathsf{mst}(R_b\cup S_b) + \mathsf{mst}(R_{b+1}\cap S_{b+1}) \qquad (4) \\
&= \sum_{i\in[a,b]:i \text{ odd}} \big(\mathsf{mst}(S_{i-1}\cup R_i) + \mathsf{mst}(R_i\cup S_i) - \mathsf{mst}(R_i)\big) \\
&\quad + \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \mathsf{mst}(R_{b+1}\cap S_{b+1}) \qquad \text{(by reorganizing terms)} \\
&\le \sum_{i\in[a,b]:i \text{ odd}} (\mathsf{mst}(S_{i-1}) + \mathsf{mst}(S_i)) + \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \mathsf{mst}(R_{b+1}\cap S_{b+1}) \\
&\qquad \text{(by Corollary 8)} \\
&= \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \sum_{i=a-1}^{b} \mathsf{mst}(S_i) + \mathsf{mst}(R_{b+1}\cap S_{b+1}).
\end{aligned}
$$

It remains to argue about (4). Notice that $S_{i-1}\cap R_i$ and $R_i\cap S_i$ form a partition of R_i. By the way we obtain R_i^{left} and R_i^{right}, we have $\mathsf{mst}(R_{i-1}\cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}}\cup R_{i+1}) \le \mathsf{mst}(R_{i-1}\cup(S_{i-1}\cap R_i)) + \mathsf{mst}((R_i\cap S_i)\cup R_{i+1}) = \mathsf{mst}(R_{i-1}\cup S_{i-1}) + \mathsf{mst}(S_i\cup R_{i+1})$. Then $R_{a-1}^{\mathsf{right}}\cup R_a \subseteq R_{a-1}\cup R_a = (S_{a-2}\cap R_{a-1})\cup S_{a-1}\cup R_a$ and $R_b\cup R_{b+1}^{\mathsf{left}} \subseteq R_b\cup R_{b+1} = R_b\cup S_b\cup(R_{b+1}\cap S_{b+1})$. Combining the facts with (1) gives (4).

Now we prove that $2\left(\mathsf{mst}(S_{a-2}\cap R_{a-1}) + \sum_{i=a-1}^{b}\mathsf{mst}(S_i) + \mathsf{mst}(R_{b+1}\cap S_{b+1})\right) \le (b-a+4)F$, which finishes the proof of the lemma. We define $\mathcal{S} = \{(S_{a-2}\cap R_{a-1}), S_{a-1}, S_a, S_{a+1}, \cdots, S_b, (R_{b+1}\cap S_{b+1})\}$ for convenience. Then $\mathcal{S}$ forms a partition of $Q := R_{a-1}\cup R_a\cup R_{a+1}\cup\cdots\cup R_{b+1}$. Focus on the trips in the optimum solution that serve at least one request in Q. By the definition of S_i's, each such trip can not serve requests from two different sets in $\mathcal{S}$. Moreover, such a trip can not start before time $(a-2)F$ since all requests in Q arrive no earlier than $(a-2)F$. It can not end after $(b+1)F+F=(b+2)F$ since all requests in Q arrive before $(b+1)F$ and the maximum flow time of the optimum solution is at most F. Therefore, $2\sum_{S\in\mathcal{S}}\mathsf{mst}(S)$ is at most the total length of these trips, which is at most $(b+2)F-(a-2)F=(b-a+4)F$. ◀

By Lemma 9, the total size of jobs (i.e, bundles) released in $[t,t']$ is at most $(t'-t)+4F$ for any $t\le t'$, using the language of the scheduling setting. Therefore, the greedy scheduling algorithm produces a schedule with maximum flow time at most $4F$. For an odd i, R'_i is released at time $(i+2)F$, and thus all requests in R'_i are served by time $(i+6)F$. Since requests in R'_i arrive no earlier than $(i-2)F$, the maximum flow time of all requests is at most $8F$. This finishes the proof of Theorem 2 for the case $k=1$.

4 Online Uncapacitated Multiple-Vehicle FDP on Tree Metrics

Now we move on to the case of general k for online FDP on trees. The first step is, again, using Algorithm 1 to partition jobs into bundles. As we mentioned, the main differences between the multi-vehicle case and the single-vehicle case are: We use a different criteria to break R_i for an even $i\ge 2$ into R_i^{left} and R_i^{right}, and we break a bundle R'_i for an odd $i\ge 1$ into groups and treat each group as a job (instead of treating the whole bundle as a job).

4.1 Partitioning R into Bundles

As before we also generate the set $(R'_i)_{i\geq 1, i \text{ is odd}}$ in the first step, which is also described in Algorithm 1. To break R_i into R_i^{left} and R_i^{right}, we use the function $\mathsf{cost}(R_i^{\mathsf{left}}|R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}}|R_{i+1})$. We define the relevant notations now.

Given a real number $F' > 0$ and a set X of requests, we define

$$c_{F'}(X,e) = \left\lceil \frac{\mathsf{mst}_e(X)}{F'} \right\rceil, \forall e \in E, \text{ and } \mathsf{cost}_{F'}(X) = 2\sum_{e\in E} c_{F'}(X,e)\ell(e).$$

Most of the time we shall use the definitions with $F' = F$. Therefore we simply use $c(X,e)$ and $\mathsf{cost}(X)$ to denote $c_F(X,e)$ and $\mathsf{cost}(X)$.

Then, for two sets X and X' of requests, we define

$$c(X,e|X') = \begin{cases} \left\lceil \frac{\mathsf{mst}_e(X)}{F} \right\rceil = c(X,e) & \text{if } V_e \cap X' = \emptyset \\ \left\lfloor \frac{\mathsf{mst}_e(X'\cup X) - \mathsf{mst}_e(X')}{F} \right\rfloor & \text{if } V_e \cap X' \neq \emptyset \end{cases}, \quad \forall e \in E,$$

$$\text{and } \mathsf{cost}(X|X') = 2\sum_{e\in E} c(X,e|X')\ell(e).$$

By the definition, we have $c(X,e|X') = c(X \setminus X', e|X')$, and $\mathsf{cost}(X|X') = \mathsf{cost}(X \setminus X'|X')$.

We now elaborate more on the definitions. Unlike the single-vehicle case, we need to break each bundle R'_i created into many groups, to make sure that each group can be served in time $O(F)$. Then an edge $e \in E$ in $\mathsf{mst}(R'_i)$ may need to be used by many trips to satisfy the property. Then $c(X,e)$ gives a lower bound on the number of bi-directional traverses of e needed to serve X in the offline optimum solution. Thus $\mathsf{cost}(X)$ gives the total time needed to serve X. Notice if a trip uses an edge e, it uses e twice, hence the factor of 2.

For $c(X,e|X')$, one can think of it as $c(X\cup X', e) - c(X', e) = \left\lceil \frac{\mathsf{mst}_e(X\cup X')}{F} \right\rceil - \left\lceil \frac{\mathsf{mst}_e(X')}{F} \right\rceil$, which is at least $\left\lfloor \frac{\mathsf{mst}_e(X\cup X')}{F} - \frac{\mathsf{mst}_e(X')}{F} \right\rfloor$. That is, the extra number of traverses of e needed if we grow the set of request positions from X' to $X \cup X'$. In the actual definition, we use the lower bound instead if $V_e \cap X' \neq \emptyset$.

4.2 Upper Bound on Costs of Bundles

The main goal of the section is to prove the following theorem:

▶ **Theorem 10.** *For every two odd positive integers $a \leq b$, we have $\sum_{i\in[a,b]} \mathsf{cost}_{3F}(R'_i) \leq k(b-a+4)F$.*

We prove the following three inequalities, which will imply the theorem:

$$\sum_{i\in[a,b]} \mathsf{cost}_{3F}(R'_i) \leq \sum_{i\in[a,b]: i \text{ odd}} \Big(\mathsf{cost}(R_i) + \mathsf{cost}(R_{i-1}^{\mathsf{right}}|R_i) + \mathsf{cost}(R_{i+1}^{\mathsf{left}}|R_i)\Big) \tag{5}$$

$$\leq \mathsf{cost}(S_{a-2} \cap R_{a-1}) + \sum_{i=a-1}^{b} \mathsf{cost}(S_i) + \mathsf{cost}(R_{b+1} \cap S_{b+1}) \tag{6}$$

$$\leq k(b-a+4)F. \tag{7}$$

We first prove (7), by showing that the $\mathsf{cost}()$ function indeed capture the length of trips in optimum solution.

▷ Claim 11. Let $R' \subseteq R$, and $\mathcal{P}$ be the set of trips in the offline optimum solution that serve at least one request in R'. Then every $e \in E$ is used by at least $c(R', e)$ trips in $\mathcal{P}$, implying that the total cost of $\mathcal{P}$ is at least $\mathsf{cost}(R')$.

Proof. Focus on each edge $e \in E$, and assume $e = (u, v)$, where v is the child vertex. If $V_e \cap R' = \emptyset$ then $c(R', e) = 0$ and the statement holds trivially. If $V_e \cap R' \neq \emptyset$ and $\mathsf{mst}_e(R') \leq F$, then $c(R', e) = 1$ and at least 1 trip in $\mathcal{P}$ uses e and the claim also holds. So, from now on, we assume $\mathsf{mst}_e(R') > F$.

Focus on a trip $P \in \mathcal{P}$ that uses e. For convenience, we treat P as the sub-tree of edges used by P, without double-counting each edge. The total length of descendant edges of e in P (excluding e itself) is at most $F - d(o, v)$. This holds since P contains edges with a total length of at most F, and it contains the edges from o to v. On the other hand, all the descendant edges of v in $\mathsf{mst}_e(R')$ should be contained in $\mathcal{P}$. The total length of these edges is $\mathsf{mst}_e(R') - d(o, v)$. Therefore, the number of trips that use e is at least $\left\lceil \frac{\mathsf{mst}_e(R') - d(o,v)}{F - d(o,v)} \right\rceil \geq \left\lceil \frac{\mathsf{mst}_e(R')}{F} \right\rceil = c(R', e)$, where the inequality comes from that $\mathsf{mst}_e(R') > F \geq d(o, v)$. The lemma holds as each trip that traverses e does this twice. ◁

Proof of (7)**.** The argument is similar to that in the last paragraph inside the proof of Lemma 9, except now we use $\mathsf{cost}(S)$, instead of $2\mathsf{mst}(S)$, to lower bound the length of trips for a set $S \in \mathcal{S}$ of requests, and there are $k \geq 2$ vehicles. ◀

Then we turn to (6). We first show some simple inequalities about cost function.

▶ **Lemma 12.** *For any $X_1, X_2, X_3, X_4 \subseteq V$, we have*

$$\mathsf{cost}(X_1 \cup X_2 | X_3) \leq \mathsf{cost}(X_1) + \mathsf{cost}(X_2 | X_3), \tag{8}$$

$$\mathsf{cost}(X_2 \cup X_3) + \mathsf{cost}(X_1 | X_2 \cup X_3) + \mathsf{cost}(X_4 | X_2 \cup X_3) \leq \mathsf{cost}(X_1 \cup X_2) + \mathsf{cost}(X_3 \cup X_4). \tag{9}$$

Let S_i's be defined in the same way as in the single-vehicle case: all the requests in a trip containing only requests from R_i, or requests from R_i and R_{i+1}, are put into S_i.

▶ **Corollary 13.** *For any i, we have* $\mathsf{cost}(R_i) + \mathsf{cost}(R_{i-1} \cap S_{i-1} | R_i) + \mathsf{cost}(S_i \cap R_{i+1} | R_i) \leq \mathsf{cost}(S_{i-1}) + \mathsf{cost}(S_i)$.

Proof. The corollary follows from (9) by setting $X_1 = R_{i-1} \cap S_{i-1}, X_2 = S_{i-1} \cap R_i, X_3 = R_i \cap S_i$ and $X_4 = S_i \cap R_{i+1}$. ◀

Recall that in our algorithm, for every even i, we break R_i into R_i^{left} and R_i^{right} so as to minimize $\mathsf{cost}(R_i^{\mathsf{left}} | R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}} | R_{i+1})$. Then, we can proceed to show (6) and (5), whose proof are tedious but straightforward. Due to space limit, we leave the complete calculation in the full version.

4.3 Breaking Bundles into Groups

In this section, we break each bundle R'_i into many groups $\mathcal{Q}_i$ as in the following lemma.

▶ **Lemma 14.** *For every odd integer $i \geq 1$, we can efficiently find a partition $\mathcal{Q}_i$ of R'_i such that* $\mathsf{mst}(Q) \leq 8F$ *for every $Q \in \mathcal{Q}_i$, and* $2 \sum_{Q \in \mathcal{Q}_i} \mathsf{mst}(Q) \leq \mathsf{cost}_{3F}(R'_i)$.

Proof. Within this proof, we need to change T to a binary tree by replacing each internal vertex v with at least three children with a binary tree. For every such vertex v with $d_v \geq 3$ children, we replace the star containing v and its children by a gadget, which is a complete binary tree with d_v leaves. We then identify the root of the gadget with v, and the leaves with the d_v children of v. In the gadget, the length of an edge incident to a child u of v is set to $\ell(u, v)$, and the length of an edge not incident to a child is set to 0. It is easy to see that this transformation does not change the instance, except now we have edges of length 0. After this step, every internal vertex of T has degree exactly 2.

For any vertex v in T and a set R' of requests, we define $\mathsf{mst}'_v(R')$ to be cost of the minimum spanning tree containing v and $V_v \cap R'$, where V_v is the set of descendant vertices of v in T (including v itself). Notice an important difference between the definition of $\mathsf{mst}'_v(R')$ and $\mathsf{mst}_v(R')$ for an edge e: for $\mathsf{mst}'_v(R')$ the tree does not need to contain the depot o and thus the quantity does not count the total length $d(o, v)$ of edges from o to v. Similarly, for an edge $e = (u, v)$ with v being the child end-vertex, we use $\mathsf{mst}'_e(R')$ to denote $\mathsf{mst}'_v(R')$. Also, sometimes we use $\mathsf{mst}'_v(R')$ to denote the tree achieving the cost $\mathsf{mst}'_v(R')$.

The following is the pseudo-code for constructing $\mathcal{Q}_i$:

1: $R' \leftarrow R'_i$, $\mathcal{Q}_i \leftarrow \emptyset$.
2: **while** $R' \neq \emptyset$ **do**
3: choose a lowest vertex v such that $\mathsf{mst}'_v(R') \geq 3F$; if no vertices v satisfy the condition, let $v = o$
4: $Q \leftarrow \{\rho \in R' : v_\rho \in V_v\}$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q\}$, $R' \leftarrow R' \setminus Q$.

It is easy to see that $\mathcal{Q}$ form a partition of R'_i. Focus on any iteration of the loop, and let v be the vertex chosen in the iteration. We argue that we have $\mathsf{mst}'_v(R') + d(o, v) \leq 8F$, implying that the set Q added in the iteration has $\mathsf{mst}(Q) = \mathsf{mst}'_v(R') + d(o, v) \leq 8F$. To see this, assume the two children of v in T are u and u'. (If v is a leaf the statement is trivial.) By our choice of v we have $\mathsf{mst}'_u(R') < 3F$ and $\mathsf{mst}'_{u'}(R') < 3F$. Then we have $\mathsf{mst}_v(R') \leq \mathsf{mst}'_u(R') + \mathsf{mst}'_{u'}(R') + d(o, v) + d(v, u) + d(v, u') \leq 3F + 3F + d(o, u) + d(o, u') \leq 6F + F + F = 8F$. The first inequality may not hold with equality since (v, u) or (v, u') may not be in $\mathsf{mst}'_v(R')$.

It remains to prove that for every edge $e \in E$, at most $\left\lceil \frac{\mathsf{mst}'_e(R'_i)}{3F} \right\rceil \leq \left\lceil \frac{\mathsf{mst}_e(R'_i)}{3F} \right\rceil = c_{3F}(R'_i, e)$ groups $Q \in \mathcal{Q}_i$ use the edge e. Consider any edge e and assume the group Q is constructed in an iteration in which the vertex we choose is v. If $v \in V_e$, then $\mathsf{mst}'_e(R')$ is reduced by at least $3F$ in the iteration since all the edges in $\mathsf{mst}'_v(R')$ are removed from $\mathsf{mst}'_e(R')$. Otherwise, v must be an ancestor of the parent end-vertex of e. In this case, $\mathsf{mst}'_e(R')$ becomes $\emptyset$ and thus this is the last time e is used. Moreover once $\mathsf{mst}'_e(R')$ becomes 0, there are no requests in V_e. Therefore, e is used at most $\left\lceil \frac{\mathsf{mst}'_e(R'_i)}{3F} \right\rceil$ times. ◀

4.4 Scheduling Groups Greedily

Once we partitioned each bundle R'_i into many groups $\mathcal{Q}_i$, we can then treat the groups as jobs and the k vehicles as k machines. Each group $Q \in \mathcal{Q}_i$ can be viewed as a job of size $2\mathsf{mst}(Q)$ that is released at time $(i + 2)F$. We need to analyze the maximum flow time achieved using the FIFO algorithm.

Now we define the goal more formally in the language of the scheduling problem. A job j arrives time r_j, and upon its arrival, we know its processing time p_j. We need to schedule the jobs on the k machines non-preemptively so as to minimize the maximum flow time over

all jobs, where the flow time of a job is its completion time minus its release time. We need to design an online algorithm: Once we started processing a job on a machine, we need to complete the job on the machine. Needless to say, the decisions made at or before time t can only depends on the jobs arrived at or before time t.

We consider the simple FIFO (first-in-first-out) algorithm: whenever there is an idle machine and an available job (an arrived job that is not being processed), we process the available job with the earliest releasing time on the machine. The following simple lemma is implicit in the analysis of FIFO on scheduling to minimize the maximum flow time:

▶ **Lemma 15.** *Let P be the maximum processing times of all jobs and $D \geq 0$. Assume for any two time points $a \leq b$, the total size of jobs released in time $[a, b]$ is at most $k(b-a)+D$, then the maximum flow time achieved by FIFO is at most $\frac{D}{k} + \frac{2(k-1)P}{k}$.*

Notice that when $k = 1$, the upper bound is simply D, and it is a folklore that the FIFO algorithm is optimum.

Proof. Focus on a job j and let s_j be its starting time in the schedule produced by the greedy algorithm. Let $t \leq s_j$ be the latest time such that some machine is idle in $(t-\epsilon, t)$ for some positive ϵ. (It is possible that $t = 0$.) Notice that we have $t \leq r_j$ since otherwise j could be started on that machine at time $t - \epsilon$.

All the k machines are busy in time (t, s_j). All but at most $k-1$ jobs that are processed fully or partially in (t, s_j) are released in $[t, r_j]$. The total size of the jobs (including the at most $k-1$ jobs) is at most $(k-1)P+k(r_j-t)+D$, by the conditions of the lemma. Therefore we have $k(s_j - t) + p_j \leq (k-1)P + k(r_j - t) + D$, which is $s_j + \frac{p_j}{k} \leq \frac{(k-1)P}{k} + r_j + \frac{D}{k}$. This implies the flow time of j is $s_j + p_j - r_j \leq \frac{(k-1)P}{k} + \frac{D}{k} + \frac{(k-1)}{k} p_j \leq \frac{D}{k} + \frac{2(k-1)P}{k}$. ◀

We then use Lemma 15 to bound the maximum flow time for the food delivery problem. By Lemma 14, all jobs (which correspond to groups) have size at most $16F$. By Lemma 14 and Theorem 10, the total size of jobs released at any interval $[t, t']$ is at most $(t'-t)k+4kF$. Therefore, using the greedy algorithm and Lemma 15, every job will have flow time at most $\frac{4kF}{k} + 2 \times 8F = 20F$. Notice that requests in R'_i has arrival time at least $(i-2)F$ and is released at time $(i+2)F$. So, every request has flow time at most $20F + 4F = 24F$. This finishes the proof of Theorem 2 for general k, except for the proof of the running time of Algorithm 1, which we argue next.

Assuming all lengths are integers. Then it is easy to design a pseudo-polynomial time algorithm to find a partition $(R_i^{\mathsf{left}}, R_i^{\mathsf{right}})$ of R_i to minimize $\mathsf{cost}(R_i^{\mathsf{left}}|R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}}|R_{i+1})$ using dynamic programming. Consider any two edges e' and e such that e' is an ancestor of e. Then given $\mathsf{mst}_e(R_i^{\mathsf{left}} \cup R_{i-1}) - \mathsf{mst}_e(R_{i-1})$, $c(R_i^{\mathsf{left}}, e'|R_{i-1})$ does not depend on the set $\left\{\rho \in R_i^{\mathsf{left}} : v_\rho \in V_e\right\}$. Similarly, given $\mathsf{mst}_e(R_i^{\mathsf{right}} \cup R_{i+1}) - \mathsf{mst}_e(R_{i+1})$, $c(R_i^{\mathsf{right}}, e'|R_{i+1})$ does not depend on the set $\left\{\rho \in R_i^{\mathsf{right}} : v_\rho \in V_e\right\}$. Therefore, we can design a dynamic programming where for each e and two integers M_{left} and M_{right}, we have a cell indicating if there is a partition $(R_i^{\mathsf{left}}, R_i^{\mathsf{right}})$ of R_i such that $\mathsf{mst}_e(R_i^{\mathsf{left}} \cup R_{i-1}) - \mathsf{mst}_e(R_{i-1}) = M_{\mathsf{left}}$ and $\mathsf{mst}_e(R_i^{\mathsf{right}} \cup R_{i+1}) - \mathsf{mst}_e(R_{i+1}) = M_{\mathsf{right}}$. To compute the value of a cell $(e, M_{\mathsf{left}}, M_{\mathsf{right}})$, we look at all cell $(e', M'_{\mathsf{left}}, M'_{\mathsf{right}})$s with e' being a *child* of e, and check if there's a way to combine the M'_{left}s (resp. M'_{right}s) to get M_{left} (resp. M_{right}). (To make this step fast we may assume the input tree is a binary tree: this can be done by adding zero-length dummy edges and vertices) To make the algorithm truly polynomial, we can round all edge lengths to integer multiples of $\epsilon F/(n|R|)$ for any small constant $\epsilon > 0$. Since every one of the n edges is used by at most $|R|$ requests, the total error incurred due to the rounding is at most $\epsilon F/(n|R|) \cdot n|R| = \epsilon F$, which can be ignored.

References

1 Kemal Altinkemer and Bezalel Gavish. Technical note - heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24:294–297, 1990.

2 Norbert Ascheuer, Sven O Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 639–650. Springer, 2000.

3 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'09, pages 1238–1244, USA, 2009. Society for Industrial and Applied Mathematics.

4 Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Baruch Schieber, and Cliff Stein. Non-preemptive min-sum scheduling with resource augmentation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 614–624. IEEE, 2007.

5 Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph Naor. Approximating the average response time in broadcast scheduling. In *SODA*, volume 5, pages 215–221. Citeseer, 2005.

6 Nikhil Bansal and Bouke Cloostermans. Minimizing maximum flow-time on related machines. *Theory of Computing*, 12(1):1–14, 2016.

7 Yair Bartal and Shan Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 558–559, 2000.

8 Jatin Batra, Naveen Garg, and Amit Kumar. Constant factor approximation algorithm for weighted flow time on a single machine in pseudo-polynomial time. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 778–789. IEEE Computer Society, 2018.

9 Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. Traveling repairperson, unrelated machines, and other stories about average completion times. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2021.

10 Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 1–14. Springer, 2021.

11 Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. *ACM Transactions on Algorithms (TALG)*, 7(4):1–14, 2011.

12 Moses Charikar and Balaji Raghavachari. The finite capacity dial-a-ride problem. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 458–467. IEEE, 1998.

13 Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Minimizing maximum response time and delay factor in broadcast scheduling. In *European Symposium on Algorithms*, pages 444–455. Springer, 2009.

14 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1114–1133. SIAM, 2014.

15 Vincent Cohen-Addad, Arnold Filtser, Philip N Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 589–600. IEEE, 2020.

16 Aparna Das and Claire Mathieu. A quasi-polynomial time approximation scheme for euclidean capacitated vehicle routing. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 390–403. SIAM, 2010.

17 Leah Epstein and Rob van Stee. Optimal on-line flow time with resource augmentation. *Discrete applied mathematics*, 154(4):611–621, 2006.

18 Uriel Feige, Janardhan Kulkarni, and Shi Li. A polynomial time constant approximation for minimizing total weighted flow-time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1585–1595. SIAM, 2019.

19 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

20 Inge Li Gørtz. Hardness of preemptive finite capacity dial-a-ride. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 200–211. Springer, 2006.

21 Inge Li Gørtz, Viswanath Nagarajan, and R Ravi. Minimum makespan multi-vehicle dial-a-ride. In *European Symposium on Algorithms*, pages 540–552. Springer, 2009.

22 Xiangyu Guo, Kelin Luo, Zhihao Gavin Tang, and Yuhao Zhang. The online food delivery problem on stars. *Theoretical Computer Science*, 2022. doi:10.1016/j.tcs.2022.06.007.

23 Anupam Gupta, MohammadTaghi Hajiaghayi, Viswanath Nagarajan, and Ramamoorthi Ravi. Dial a ride from k-forest. *ACM Transactions on Algorithms (TALG)*, 6(2):1–21, 2010.

24 Mordecai Haimovich and Alexander HG Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.

25 Dawsen Hwang and Patrick Jaillet. Online scheduling with multi-state machines. *Networks*, 71(3):209–251, 2018.

26 Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1070–1086. SIAM, 2014.

27 Aditya Jayaprakash and Mohammad R Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 877–893. SIAM, 2022.

28 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.

29 Bala Kalyanasundaram, Kirk Pruhs, and Mahe Velauthapillai. Scheduling broadcasts in wireless networks. In *European Symposium on Algorithms*, pages 290–301. Springer, 2000.

30 Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 32–45, 2021.

31 Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 418–426. ACM, 1996. doi:10.1145/237814.237989.

32 Michael Khachay and Roman Dubinin. Ptas for the euclidean capacitated vehicle routing problem in r^d. In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.

33 Sven O Krumke, Willem E de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In *International Workshop on Approximation and Online Algorithms*, pages 258–269. Springer, 2005.

34 Sven O Krumke, Willem E De Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1-3):279–294, 2003.

35 Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness helps: An $o(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *APPROX*, pages 200–214. Springer, 2002.

36 Martine Labbé, Gilbert Laporte, and Hélene Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

37 Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling on unrelated machines with rejections. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 291–300, 2018.

38 Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling to minimize weighted flow-time on unrelated machines. *arXiv preprint*, 2018. arXiv:1804.08317.

39 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *European Symposium on Algorithms (ESA 2016)*, volume 57(63), pages 1–17, 2016.

40 Monaldo Mastrolilli. Scheduling to minimize max flow time: Off-line and on-line algorithms. *International Journal of Foundations of Computer Science*, 15(02):385–401, 2004.

41 Claire Mathieu and Hang Zhou. A ptas for capacitated vehicle routing on trees. *arXiv preprint*, 2021. arXiv:2111.03735.

42 Cynthia A Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 140–149, 1997.

43 Lars Rohwedder and Andreas Wiese. A $(2 + \varepsilon)$-approximation algorithm for preemptive weighted flow time on a single machine. *arXiv preprint*, 2020. arXiv:2011.05676.

44 Yuanxiao Wu and Xiwen Lu. Capacitated vehicle routing problem on line with unsplittable demands. *Journal of Combinatorial Optimization*, pages 1–11, 2020.

Minimum Link Fencing

Sujoy Bhore
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Fabian Klute
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Maarten Löffler
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Martin Nöllenburg
Algorithms and Complexity Group, TU Wien, Austria

Soeren Terziadis
Algorithms and Complexity Group, TU Wien, Austria

Anaïs Villedieu
Algorithms and Complexity Group, TU Wien, Austria

Abstract

We study a variant of the geometric multicut problem, where we are given a set $\mathcal{P}$ of colored and pairwise interior-disjoint polygons in the plane. The objective is to compute a set of simple closed polygon boundaries (*fences*) that separate the polygons in such a way that any two polygons that are enclosed by the same fence have the same color, and the total number of links of all fences is minimized. We call this the *minimum link fencing* (MLF) problem and consider the natural case of *bounded minimum link fencing* (BMLF), where $\mathcal{P}$ contains a polygon Q that is unbounded in all directions and can be seen as an outer polygon. We show that BMLF is NP-hard in general and that it is XP-time solvable when each fence contains at most two polygons and the number of segments per fence is the parameter. Finally, we present an $O(n \log n)$-time algorithm for the case that the convex hull of $\mathcal{P} \setminus \{Q\}$ does not intersect Q.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases computational geometry, polygon nesting, polygon separation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.34

Related Version *Full Version*: arXiv:2209.14804 [4]

Funding *Fabian Klute*: Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.651 and the Austrian Science Foundation (FWF) grant J4510.
Anaïs Villedieu: Supported by the Austrian Science Fund (FWF) under grant P31119.

Acknowledgements The authors would like to thank Thekla Hamm and Irene Parada for the valuable discussion in particular concerning the XP results of Section 3.

1 Introduction

In the *geometric multicut* problem [2], we are given κ disjoint sets of polygons in the plane, each with a different *color*, and are asked for a subdivision of the plane such that no cell of the subdivision contains multiple colors. The goal is to minimize the total length of the subdivision edges.

A different kind of separation is achieved in the *polygon nesting* problem [3], where for two polygons P and Q with $P \subset Q$ one asks for a polygon P' with the smallest number of links, s.t. $P \subset P' \subset Q$. There exists a series of work that addressed the algorithmic complexity of nesting problems for various polygon families [3, 5, 9, 12, 13]. See Section 1.2 for more detail.

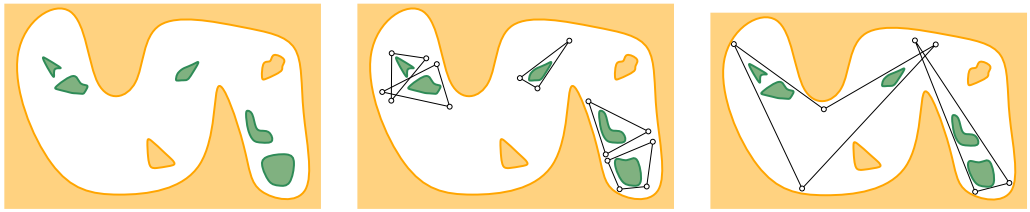


Figure 1 Two sets of polygons in the plane (left) with different colors (green and yellow). The yellow set effectively acts as an outer polygon with holes. Separating the two sets with, possibly intersecting, individual fences (middle) can lead to significantly more links in the fences (here 16) than grouping same-colored polygons (right), which achieves this with just seven links.

In this paper, we consider a variant of geometric multicut inspired by polygon nesting, where we separate the sets from each other with a set of closed polygon boundaries called *fences*, which enclose only polygons of one color and have the smallest possible number of links. If one or more sets are not connected, we need to solve the combinatorial problem of choosing which polygons should be grouped in each fence. Figure 1 illustrates the problem. Some variants of the fencing problem already become NP-hard for point objects with two colors, e.g., if we require the fence to be a single closed curve [6].

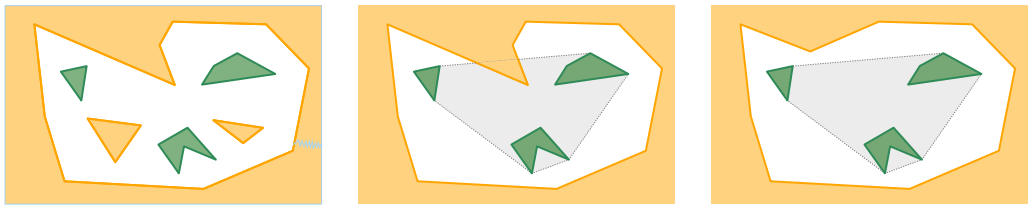
In this paper, we assume the input sets are collections of polygons, one color covers the plane minus a single polygonal hole (the outer polygon, a parallel to polygon nesting), and we will focus on the case $\kappa = 2$ of two colors. We use n to denote the total number of corners of the input polygons. Even in this simple setting the problem turns out to be non-trivial. If both sets are connected, then the problem is equivalent to finding a minimal *nested* polygon, which can be solved in $O(n \log n)$ time [3]. If both sets are not connected we show this problem to be NP-hard in Section 2. Note the contrast to the geometric multicut problem, which is polynomially solvable for $\kappa = 2$ [1] but becomes NP-hard when $\kappa = 3$ [2]. In Section 3 we show that, when restricting every fence to contain at most two polygons, the problem admits an XP-algorithm when parameterized by the maximal number of segments per fence, a result which holds for any κ. Finally, in Section 4, we show that the problem is polynomial-time solvable if the convex hull of the second color (the *inner polygons*) is contained in the outer polygon and the first color is connected.

1.1 Problem Definition

Throughout this paper we consider polygons in $\mathbb{R}^2$ without self-intersections but potentially with holes. Moreover, we consider a polygon as the boundary together with its interior, unless stated otherwise. We consider the following problem.

▶ **Definition 1** (Minimum Link Fencing (MLF)). *We are given n pairwise interior-disjoint polygons $\mathcal{P} = \{P_1, \ldots, P_{|\mathcal{P}|}\}$ in the plane, with a coloring function $f : \mathcal{P} \to \{1, \ldots, \kappa\}$, which assigns a color to every input polygon. We write $\mathcal{P}_i = \{P \mid f(P) = i\}$. We want to find a set of simple closed polygon boundaries $\mathcal{F} = \{F_1, \ldots, F_m\}$ such that the total number of links $|F|$ on the boundary of $F = \bigcup_{i=1}^{k} F_i$ is minimized and if two polygons P_a and P_b are enclosed by the same fence or are both in $\mathbb{R}^2 \setminus \bigcup_{i=1}^{k} \overline{F_i}$, where $\overline{F_i}$ is the polygon bounded by F_i, then $f(P_a) = f(P_b)$. We call F_i a fence and $\mathcal{F}$ a* minimum link fencing *of $\mathcal{P}$.*

Note the important difference in Definition 1 between $\mathcal{F}$, which is the set of all fences of a solution, and F, which is the union over all fences, i.e., one (possibly disconnected) polygon. Thus $|\mathcal{F}|$ is the number of fences and $|F|$ is the number of all segments in these fences.



(a) BMLF **(b)** SMLF **(c)** CMLF

Figure 2 Different problem inputs corresponding to (a) BMLF, (b) SMLF and (c) CMLF. In (b) and (c) the convex hull of all input polygons indicated in gray.

Throughout the paper we refer to $\mathbb{R}^2 \setminus \bigcup_{i=1}^{|\mathcal{P}|} P_i$ as the *free space* (between polygons). We refer to $\mathcal{P}$, which contains polygons of κ different colors, as κ-colored and to the problem setting as the κ-colored problem. We consider several problem variations.

If there exists a polygon $Q \in \mathcal{P}$ which is unbounded in every direction, i.e. $\mathbb{R}^2 \setminus Q$ is finite, this polygon Q effectively acts as an outer boundary. In this case we call the problem Bounded Minimum Link Fencing (BMLF). We denote the polygon Q as the *outer polygon*. As a consequence, the size of the outer polygon automatically bounds the length of any link in a fence. Else, in general, one fence could conain a very long link, while retaining small complexity when counting the number of links only. Note that Q can be emulated in an instance of Minimum Link Fencing, by adding a large rectangular polygon $P_c \setminus (\mathbb{R}^2 \setminus Q)$, i.e., a large rectangle, of which the area, which did not belong to Q is cut out (light blue channel in Figure 2a). If Q is the only polygon of its color $f(Q)$ we call this setting Simply Bounded Minimum Link Fencing (SMLF). Moreover, if in an instance of SMLF we have $CH(\bigcup_{i=1}^{\kappa} \mathcal{P}_i \setminus Q) \subset \mathbb{R}^2 \setminus Q$, i.e., the convex hull of all input polygons except Q does not intersect Q, we speak of Convex Bounded Minimum Link Fencing (CMLF). The differences are illustrated in Figure 2.

1.2 Related Work

Despite the fact that the problem is natural and fundamental, little previous work exists. The problem of *enclosing* a set of objects by a shortest system of fences has recently been considered with a single set B_1 [1]. The task is to "enclose" the components of B_1 by a shortest system of fences. This can be formulated as a special case of our problem with $\kappa = 2$ colors: We add an additional set B_2, far away from B_1 and large enough so that it is never optimal to surround B_2. Thus, we have to enclose all components of B_1 and separate them from the unbounded region. In this setting, there will be no nested fences. Abrahamsen et al. [1] gave an $O(n \text{ polylog } n)$-time algorithm for inputs that consist of n unit disks.

Some variations with additional constraints on the fence become NP-hard already for point objects with two colors. For example, if we require the fence to be a single closed curve, it has been observed by Eades and Rappaport [6] already in 1993 that one can model the Euclidean Traveling Salesman Problem of computing the shortest tour through a given set of sites by placing two tiny objects of opposite color next to each site. If we require the fence to be connected, the same construction will lead to the Euclidean Steiner Tree Problem, which was shown to be NP-hard by Garey et al. in 1977 [8].

Polygon Nesting & Separation. Polygon nesting is considered to be a fundamental problem in computational geometry, and has been extensively studied since its inception. Aggarwal et al. [3] considered the problem of finding a polygon nested between two given convex polygons

that has a minimal number of vertices. They gave an $O(n \log k)$ time algorithm for solving the problem, where n is the total number of vertices of the given polygons, and k is the number of vertices of a minimal nested polygon. Das [5] considered a variant of MLF in his thesis, which restricts every fence to enclose exactly one polygon, and showed that the problem is NP-hard. Given a polygon Q of m vertices inside another polygon P of n vertices, Ghosh [9] gave an $O((n+m)\log k)$ time algorithm for constructing a minimum nested convex polygon, where k is the number of vertices of the output polygon, improving upon the $O((n+m)\log(n+m))$ time algorithm of Wang and Chan [14]. However, on the other hand, given a family of disjoint polygons $P_1, P_2, \ldots, P_k$ in the plane, and an integer parameter m, it is NP-complete to decide if the P_i's can be pairwise separated by a polygonal family with at most m edges. Mitchell and Suri [12] presented efficient approximation algorithms for constructing separating families of near-optimal size.

Full proofs of statements marked by (⋆) are found in the full paper [4].

2 Two-colored BMLF is NP-hard

In this section we will call polygons of color 1 *boundary polygons* and polygons of color 2 *inner polygons*. An instance of planar 3, 4-SATconsists of a Boolean CNF-formula ϕ with a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$ and a set of clauses $\mathcal{C} \subset 2^{\mathcal{V}}$, s.t. every clause is a disjunction of three literals and every variable occurs at most four times as a literal in a clause. Additionally, we are given the embedded plane incidence graph $G_\phi = (\mathcal{V} \cup \mathcal{C}, E)$, where $E = \{vc \mid v \in \mathcal{V}, c \in \mathcal{C}, v \text{ occurs as a literal in } c\}$. It is known that deciding if a 3, 4-SAT-formula has a satisfying assignment is NP-complete [11].

Given an instance of planar 3, 4-SAT we create an instance of 2-colored BMLF $\mathcal{P}$, emulating the shape of G_ϕ with one unbounded outer polygon Q and multiple boundary polygons of the same color $f(Q) = 1$ (Figure 3), s.t. ϕ is satisfiable if and only if there exists a minimum link fencing for $\mathcal{P}$ with at most a certain fixed number of total segments.

Note that each gadget is described as a basic construction of gray polygons, in which inner polygons are placed. This is possible, because we will invert all gray polygons at the end of the reduction, s.t. the area of their union makes up exactly the actual free space of our entire construction, see Figure 3. Stating that fences are computed inside the gray polygons should be understood as fences being placed in the free space between polygons. Throughout this reduction we distinguish fences based on the inner polygons they include. We call two fences F and F' *congruent*, if and only if they enclose the same set of inner polygons. We call two fencings $\mathcal{F}$ and $\mathcal{F}'$ *congruent* if there is a bijective mapping $f : \mathcal{F} \to \mathcal{F}'$, s.t., every $F \in \mathcal{F}$ is congruent to $f(F) \in \mathcal{F}'$.

Let $\mathcal{P}$ be an instance of BMLF and S_1, S_2, and S_3 disjoint connected subsets of $\mathbb{R}^2 \setminus \cup_{P\in\mathcal{P}} P$. We call the ordered set $\mathcal{S} = \{S_1, S_2, S_3\}$ a *non-collinear triple* if there are no three points $p_1 \in S_1$, $p_2 \in S_2$, and $p_3 \in S_3$ such that the straight-line segment s from p_1 to p_3 contains p_2 and s lies completely inside $\mathbb{R}^2 \setminus \cup_{P\in\mathcal{P}} P$. The choice of S_2 only matters if there exists a straight-line segment in $\mathbb{R}^2 \setminus \cup_{P\in\mathcal{P}} P$ connecting points in S_1 and S_3. Therefore we can often omit S_2 from the description of the triple or assume it as arbitrarily chosen. We call S_2 the *bend-set* of $\mathcal{S}$. Let $\mathcal{F}$ be a fencing of $\mathcal{P}$, and S_1, S_2, and S_3 a non-collinear triple. We say a fence $F \in \mathcal{F}$ *crosses* the triple $\{S_1, S_2, S_3\}$ if the boundary of F contains at least one point p_i from each set S_i for $i = 1, 2, 3$ and there is a cyclic traversal of the boundary of F in which we see first p_1, then p_2 and finally p_3. We write $]S_1, S_3[$ to denote the part of the boundary of F that lies in between p_1 and p_3 and contains p_2.

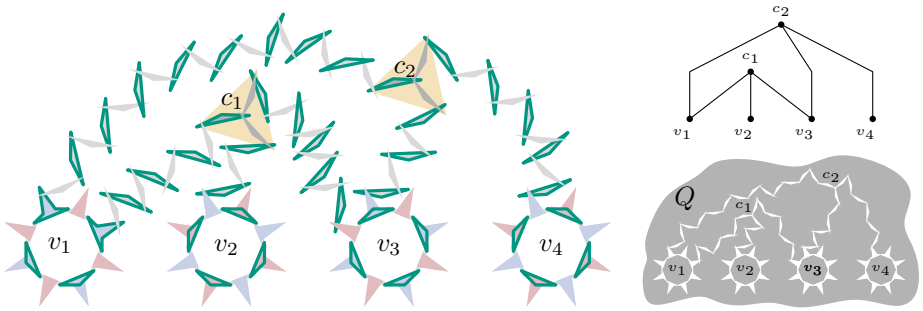


Figure 3 A (schematized) complete construction for a small instance $(v_1 \vee v_2 \vee \neg v_3) \wedge (v_1 \vee v_3 \vee v_4)$. The incidence graph is shown in the top right. Fences are highlighted in green. Also note that the boundary polygons make up most of the available area including an unbounded outer polygon Q as shown in the bottom right corner. For better readability, we will invert these colors in all subsequent figures.

▶ **Observation 2.** *Any fence in a fencing for an instance of BMLF crossing a non-collinear triple $\{S_1, S_2, S_3\}$ contains at least one bend in the interval $]S_1, S_3[$.*

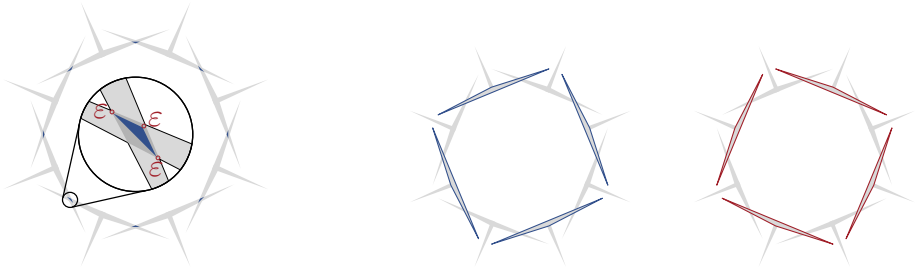
For $t > 0$ let $\mathcal{S}_1, \ldots, \mathcal{S}_t$ be non-collinear triples that are crossed by a fence F of a fencing $\mathcal{F}$ for some instance of BMLF. Let $\mathcal{S}_i = \{S_j, S_{j+1}, S_{j+2}\}$ for $i = 1, \ldots, t$ and $j = 3(i-1)+1$. We say that the triples are crossed by F *in-order* if there exist points p_i on F such that p_i is in the bend-set of $\mathcal{S}_i$ and there exists a cyclic traversal of F in which we see the points p_i in order of their indices. Without loss of generality we will assume throughout that when F crosses $\mathcal{S}_1, \ldots, \mathcal{S}_t$ in-order it always crosses for some $\mathcal{S}_i$ first the set S_i, then the bend-set S_{i+1}, and finally S_{i+2}. We write $]S_a, S_b[$ with $a < b$ and $a = 1, \ldots, 3t-1$ for the part of the boundary of F that lies between a point $p_a \in S_a$ and $p_b \in S_b$ such that there exist points $p_a, \ldots, p_b$ with $p_i \in S_i$ that we see in this order in a cyclic traversal of F. For a segment s of F we say it is *completely* contained in $]S_a, S_b[$ if the start- and endpoint of s are contained in $]S_a, S_b[$ for any choice of points p_a and p_b.

We say two non-collinear triples $\mathcal{S} = \{S_1, S_2, S_3\}$ and $\mathcal{S}' = \{S'_1, S'_2, S'_3\}$ are *non-overlapping* if there exist no two segments that intersect all six elements of $\mathcal{S} \cup \mathcal{S}'$ in order S_1, S_2, S_3, S'_1, S'_2, S'_3. In other words we require at least three different straight-line segments to connect a point $p_1 \in S_1$ with a point $p_6 \in S'_3$ and containing points $p_2 \in S_2$, $p_3 \in S_3$, $p_4 \in S'_1$, and $p_5 \in S'_2$ in order of their indices. Observe that by this definition the non-collinear triples $\{S_1, S_2, S_3\}$ and its reverse $\{S_3, S_2, S_1\}$ are non-overlapping. For a sequence of non-collinear triples $\mathcal{S}_1, \ldots, \mathcal{S}_t$ we say that the triples are non-overlapping if S_i is non-overlapping with S_{i+1} for $i = 1, \ldots, t+1 \mod t$.

Observation 2 together with the definition of non-overlapping gives the following.

▶ **Observation 3.** *Any fence in a fencing for an instance of BMLF crossing $t > 0$ non-overlapping non-collinear triples $\mathcal{S}_i = \{S_j, S_{j+1}, S_{j+2}\}$ for $i = 1, \ldots, t$ and $j = 3(i-1)+1$ in-order contains at least t bends and therefore at least $t-1$ complete straight-line segments. in the interval $]S_1, S_{3t}[$.*

We can now show a lower bound for the number of links a minimum-link fence uses in any solution of a BMLF instance. The lower bound essentially follows from Observation 3 after observing that the segment closing the fence can never reuse one of the $t-1$ segments that lie completely inside the sequence of non-collinear triples.



(a) Variable gadget construction with ε-gaps. **(b)** Fencing for *true*-state. **(c)** Fencing for *false*-state.

Figure 4 The variable gadget and its two possible fencings.

▶ **Lemma 4** (⋆). *Let $\mathcal{P}$ be an instance of BMLF and $\mathcal{F}$ a minimum-link fencing for $\mathcal{P}$, then any fence $F \in \mathcal{F}$ that crosses $t > 0$ non-overlapping non-collinear triples in-order consists of at least t straight-line segments.*

2.1 Variable gadget

Every variable gadget consists of eight *T-polygons* (two per clause in which the variable can occur). Figure 4a illustrates the construction; T-polygons are marked in gray. Every T-polygon has an isosceles triangle as the *arm* of the T (the horizontal part of the T shape) and a *spike* (alternatively called a *true spike* and a *false spike*) protruding from the arm and two consecutive polygons overlap at the end of their arms. For every variable $v \in \mathcal{V}$, we construct a variable gadget $\mathcal{G}(v)$ as a circular arrangement of eight overlapping T-polygons.

For every pair of overlapping T-polygons A and B, we place an inner polygon P, s.t. $P \subset A \cap B$. Let us fix some A, B, and P as above, then we place P such that its three corner points have only a very small distance $\varepsilon > 0$ to some corner point of $A \cap B$. All three ε-length segments between a corner point of P and the closest corner point of $A \cap B$ have to be crossed by every fence enclosing P.

Crucially, the variable gadget has only two minimum link fencings. These two states are shown in Figure 4. We associate the one shown in Figure 4b to the variable gadget encoding the value *true* and the one shown in Figure 4c to encoding *false*.

▶ **Lemma 5** (⋆). *There are exactly two minimum link fencings $\mathcal{F}_t$ and $\mathcal{F}_f$ of the variable gadget, both of which will enclose only triangles in the same T-polygon with each fence, resulting in a fencing with 12 links for the whole variable gadget, s.t. every other minimum link fencing is congruent to either $\mathcal{F}_t$ or $\mathcal{F}_f$.*

2.2 Clause gadget

For every clause $c \in \mathcal{C}$ in which three variables v_1, v_2, v_3 occur either as a positive or a negative literal, we create a clause gadget $\mathcal{G}(c)$. A clause gadget consists of three chains of an even number of gray triangles. These triangles are placed s.t. their hypotenuses intersect at an angle of at most π as shown in Figure 5a. The triangles are sufficiently long and thin, s.t., we can define two sets in every gray triangle (one to either side of the central line), s.t., the second set a' of the i-th triangle and the first set b of the $(i+1)$-th triangle form a non-collinear triple. By construction the non-collinear triple between the $(i-1)$-th and the i-th triangle and the one between the i-th and the $(i+1)$-th triangle are non-overlapping.

We place the three chains such that the three first triangles of the chains have a common intersection. Moreover, they intersect in such a way that their hypotenuses pairwise form $\frac{2\pi}{3}$ angles (Figure 5b). The last gray triangle of the first, second and third chain intersect a spike

(a) (b)

Figure 5 Wires are constructed from consecutive gray triangles places such that two consecutive triangles always contain a non-collinear triple, which are pairwise non-overlapping.

of $\mathcal{G}(v_1)$, $\mathcal{G}(v_2)$ and $\mathcal{G}(v_3)$, respectively. They intersect a true or false spike if the variable occurs as a positive or negative literal, respectively. We refer to each chain of gray polygons as a *wire*. The *length* of a wire is the number of gray triangles in its corresponding chain.

Let W_1, W_2, and W_3 be the wires of a clause gadget $\mathcal{G}(c)$ for clause c, where W_i intersects the spike of $\mathcal{G}(v_i)$ for $i \in \{1, 2, 3\}$. We place an inner triangle, denoted the *clause triangle* B_c of $\mathcal{G}(c)$, in the overlap of W_1, W_2, and W_3. Moreover, for wire W_i with gray triangles T_j^i we place inner triangles B_j^i in the overlap of the j-th and $(j+1)$-th gray triangle of the respective wire and a final triangle in the intersection with the spike of $\mathcal{G}(v_i)$. In the following we write $T_1, \ldots, T_k$ for the gray triangles and $B_1, \ldots, B_k$ for the inner polygons of one wire W_i, if i is clear from the context. Hence, inner triangle B_i is contained in the gray triangles T_i and T_{i+1} and gray triangle T_i for $i > 1$ contains the inner triangles B_i and B_{i+1}.

Let $B_1, \ldots, B_k$ be the inner polygons of a wire and F a fence containing B_i and B_j for some $i < j - 1$ and $i = 1, \ldots, k-2$ but not B_z for $i < z < j$, then we say F *bypasses* B_z. For indices $1 \leq i_1 < i_2 < j_1 < j_2 \leq k$, we say two fences F_1 and F_2 containing some polygons of the wire *interleave* if B_{i_1} and B_{j_1} are in F_1 and F_1 bypasses B_{i_2} as well as B_{i_2} and B_{j_2} are in F_2 and F_1 bypasses B_{j_1}.

Let F be a fence of a minimum link fencing $\mathcal{F}$ for a clause gadget $\mathcal{G}(c)$. Let s be a segment contained in the union of the gray triangles that form $\mathcal{G}(c)$ such that F crosses s in two points p and q. Then *splitting* F at s means the following. Delete F in an ε-region around p and q this creates two polygonal-chains, say F' and F'' with endpoints p' and q' on one side of s and p'' and q'' on the other. Connect p' with q' and p'' with q'' to form the two new fences F' and F''. Clearly, $|F'| + |F''| = |F| + 2$.

One isolated wire

For the following we fix an arbitrary clause c. Let $\mathcal{G}(c)$ be the clause gadget of c and W one of the wires of $\mathcal{G}(c)$ with inner polygons $B_1, \ldots, B_k$. We denote as *isolated wire* the gray triangles of the chain of W that do not contain the clause triangle.

We are interested in how a minimum link fencing of an isolated wire looks like. Crucially, we first show that a fence of a minimum link fencing of an isolated wire cannot bypass any inner polygon of an inner polygon.

▶ **Lemma 6** ($\star$). *A minimum link fence $\mathcal{F}$ of an isolated wire W of $\mathcal{G}(c)$ does not contain a fence $F \in \mathcal{F}$ such that F bypasses an inner polygon B_i with $i \in \{2, \ldots, k-1\}$ of W.*

In the following we are going to bound the number of consecutive polygons that are contained in one minimum link fence of an isolated wire. We compare this then to a fence containing all inner polygons of an isolated wire. Such a fence, by construction, contains

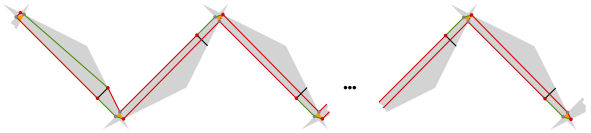


Figure 6 Non-colinear triples in a fence including a series of consecutive inner polygons.

$2z$ non-collinear triples and hence requires $2z$ segments by Lemma 4. Figure 6 shows these triples. Constructing such a fence is straight-forward by following these non-collinear triples. The following lemma summarizes this statement.

▶ **Lemma 7.** *Let $\mathcal{F}$ be a minimum link fencing of an isolated wire W of $\mathcal{G}(c)$, any fence $F \in \mathcal{F}$ that contains $z > 2$ consecutive inner polygons of W has at least $2z$ segments and such a fence exists.*

▶ **Lemma 8.** *Let $\mathcal{F}$ be a minimum link fencing of an isolated wire W of $\mathcal{G}(c)$, then every fence of $\mathcal{F}$ contains at most three consecutive inner polygons.*

Proof. Let $B_1, \ldots, B_k$ be the inner polygons of W. By Lemma 6 we can assume that the inner polygons of W contained in F are consecutive in the sequence of inner polygons. Let $F \in \mathcal{F}$ be a fence containing $z > 3$ inner polygons of W.

By Lemma 7 we know that F consists of $2z$ segments. We replace F with a fence F_1 including the two first polygons included in F and a fence F_2 including all $z-2$ following inner polygons. Again by Lemma 7 it follows that $|F_2| = 2z - 4$ and it holds that $|F_1| = 3$. In sum, we get that $|F_1| + |F_2| = 2z - 4 + 3 = 2z - 1 \leq |F|$, a contradiction. ◀

Lemmas 7 and 8 now lead to a characterization of minimum link fences of isolated wires.

▶ **Lemma 9** (⋆). *Let $\mathcal{F}$ be a minimum link fencing of an isolated wire W of $\mathcal{G}(c)$ with k inner polygons, then $\mathcal{F}$ has in total $3k/2$ segments and $F \in \mathcal{F}$ contains exactly two consecutive inner polygons B_i and B_{i+1} for i odd.*

Integrating the clause triangle

So far we only considered one arbitrary isolated wire of $\mathcal{G}(c)$. To put things together we need to consider the interaction of the three wires of $\mathcal{G}(c)$. Specifically, we need to show that no fence in a minimum link fencing of $\mathcal{G}(c)$ contains inner polygons from two different wires.

We extend the definition of bypassing an inner polygon of a wire to a whole clause gadget. Let F be a fence for $\mathcal{G}(c)$, then F *bypasses* an inner polygon B_j^i of wire W_i of $\mathcal{G}(c)$ if F contains the clause triangle B_c or some inner polygon of a wire $W_{i'}$ with $i' \neq i$ and F contains B_l^i for wire W_i with $l > j$. We say F *bypasses* the clause triangle of $\mathcal{G}(c)$ if F contains inner polygons of at least two different wires of $\mathcal{G}(c)$ but not the clause triangle B_c of $\mathcal{G}(c)$.

As for an isolated wire we can show that no inner polygon for a whole clause gadget can be bypassed. This can be seen after observing that no fence can bypass the inner polygons of an isolated wire without violating Lemma 6. The remainder of the proof is then a careful case enumeration, which can be found in the full paper [4].

▶ **Lemma 10** (⋆). *Let $\mathcal{F}$ be a minimum link fencing of $\mathcal{G}(c)$ and $B_1, \ldots, B_k$ the inner polygons of one of the wires of $\mathcal{G}(c)$. Then there is no fence $F \in \mathcal{F}$ that bypasses an inner polygon B_i with $i \in \{1, \ldots, k-1\}$.*

(a) $\mathcal{G}(v)$ in correct state. **(b)** $\mathcal{G}(v)$ in incorrect state.

Figure 7 If $\mathcal{G}(v)$ is in the correct truth state (a) inclusion of the first inner polygon of $\mathcal{G}(v, c)$ induces an additional cost of two links, otherwise (b) the additional cost is at least three.

Finally, we show that no minimum link fence of a clause gadget can ever fence two polygons that are in different wires. Again, this is shown essentially via a case enumeration that considers how a minimum link fence includes the first two to three polygons of each wire together with the clause triangle. In each case we can conclude that there exists a fence with fewer segments that in fact does not use the inner polygons of two distinct wires.

▶ **Lemma 11** (⋆). *Let $\mathcal{F}$ be an optimal fencing of a clause gadget, then there exists no fence $F \in \mathcal{F}$, which includes inner polygons belonging to two different wires.*

We can now use Lemma 11 to argue that the clause triangle is only included in a fence together with inner polygons of at most one wire. We say that such a wire is in a *satisfying* state. The other two wires should therefore, by Lemma 9, only use fences including two inner polygons; leading to $\frac{3(k_a+k_b+k_c)}{2} + 3$ segments in total (k_a, k_b and k_c being the number of inner polygons in the wires). If we include the clause triangle in a fence of a wire, we get the same amount of segments, however, we can choose fences, s.t., the last inner polygon of the wire which fences the clause triangle, is fenced alone. This will be crucial in the argument of how the wires and therefore the clause gadget interacts with the variable gadget.

Interaction with the variable gadgets

It remains to describe the interaction between the variable and clause gadgets. Depending on the state of the variable gadget we can fence the last inner polygon of a wire in the fence of a variable gadget. We provide a fence with 5 segments (i.e., two additional ones) for the case, where the variable gadget is in the correct state and the existence of six non-collinear triples for the other case, see Figure 7.

▶ **Lemma 12.** *The last inner polygon of a wire can be included in a fence of the variable gadget, whose spike it is connected to for the cost of two additional segments if the variable gadget is in the correct state and at least three additional segments otherwise.*

Concluding the interaction between clause and variable gadget we show that given a variable gadget is in the correct state w.r.t. a clause gadget we can fence the inner polygons of the wires of a clause gadget using 3/2 segments per polygon and adding only two segments to the fence of the variable gadget.

▶ **Lemma 13** (⋆). *If and only if at least one of the connected variable gadgets is in the correct state, the clause gadget can be fenced with a total of $\frac{3(k_a+k_b+k_c)}{2}$ segments plus two additional segments to a fence of the variable gadget, which is connected to the wire in the satisfying state.*

Correctness

It remains to argue the correctness of our reduction which then implies our main theorem.

▶ **Theorem 14** (⋆). *Two-colored BMLF is* NP*-hard even when restricting all fences to include at most three polygons.*

Proof sketch. For an instance ϕ of planar 3, 4-SAT, we construct a variable gadget for every variable and connect the clause gadgets accordingly. By construction any fencing with $|\mathcal{V}| \cdot 12 + \sum_{c \in \mathcal{C}} (\frac{3(k^c)}{2} + 2)$ segments, requires one wire of every clause gadget to be in a satisfying state. The connected variable gadget is forced into the true or false state, depending on the connected spike. This implies a satisfying variable assignment for ϕ.

Conversely, since every variable is either true or false and for every clause there is a true literal, we can set all variable gadgets into the true or false state according to the assignment and are guaranteed to be able to put exactly one wire per clause gadget into a satisfying state for an additional cost of exactly two. ◀

3 An XP-algorithm for BMLF with at most two polygons in each fence

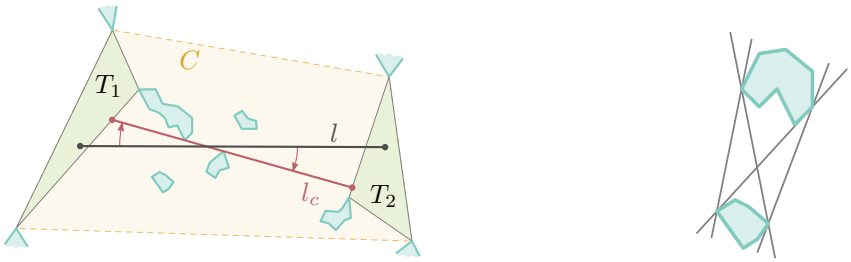
In Section 2 we showed that BMLF is NP-hard when there are only two colors, each fence contains at most three polygons, and each fence consists of at most five links. In contrast, we are going to show in this section that BMLF can be solved in XP-time when parameterizing the problem by the maximum number of links in any fence and allowing at most two polygons per fence, i.e., the problem can be solved in polynomial-time when fixing the maximum number of links in any fence and restricting each fence to contain at most two polygons.

For our algorithm we make use of the following result derived from the work of Hershberger and Snoeyink [10]. It allows us to compute for a given *loop*, i.e., a closed polygonal curve, inside a polygon with holes, a minimum-link loop of the same homotopy in time $O(nk)$, where n is the complexity of the polygon and k is the size of the resulting fence.

▶ **Theorem 15** (Derived from Section 5.2 [10]). *Given a polygon P without self-intersections but potentially with holes of complexity n, an integer k, and a loop α lying in the interior of P with $O(nk)$ corners, we can decide in time $O(nk)$ if there exists a loop α' of the same homotopy-class as α with at most k links.*

▶ **Remark 16.** *It is worth noting that in the paper by Hershberger and Snoeyink [10] Theorem 15 is only stated in text. The runtime is given as $O(C_\alpha + \Delta_\alpha + \Delta_{\alpha'})$, where C_α is the complexity of α, the free space between polygons is assumed to be triangulated and Δ_α and $\Delta_{\alpha'}$ are the number of triangulation edges intersected by α and the fence α', respectively. However an example of an instance with multiple obstacles is given, in which $\Delta_{\alpha'} \in \Omega(nk)$, where n is the number of corners over all polygons. Since in our scenario we can find a path α s.t. $C_\alpha \in O(nk)$ and $\Delta_\alpha \in O(nk)$, we can make the assumption that α''s complexity is in $O(nk)$.*

Let P be a polygon without self-intersections. We denote with $\mathcal{T}_P = \{T_1, \ldots, T_z\}$ a triangulation of P with triangles $T_1, \ldots, T_z$. Note that we do not require any further properties of $\mathcal{T}_P$. If P is clear from the context we omit it and set $\mathcal{T} = \mathcal{T}_P$. Let $T_1, T_2 \in \mathcal{T}$ be two triangles and let l be a line segment with endpoints p and q such that $p \in T_1$ and $q \in T_2$. We call l a *splitting segment*. Consider Figure 8a for an example for T_1 and T_2 if l contains no points of $\mathbb{R}^2 \setminus P$. Intuitively, a splitting segment separates the holes that intersect the convex hull of $T_1 \cup T_2$ into two sets. Let $\mathcal{H}$ be all the holes of P that intersect or are fully



(a) A splitting segment l_c equivalent to l and bitangent to two holes. **(b)** Bitangents.

Figure 8 (a) We can obtain the splitting segment l_c from a splitting segment l by rotating l clockwise until it is a bitangent to two holes. **(b)** Any pair of two polygons admits at most four bitangents, only one of which can not be rotated clockwise without intersecting one of the polygons.

contained in the interior of the convex hull of $T_1 \cup T_2$. We say that a hole $H \in \mathcal{H}$ is to the *left* (*right*) of l if the from p to q oriented supporting line of l leaves H in the left (right) half-plane. We call two splitting segments of T_1 and T_2 *equivalent* if the same holes of $\mathcal{H}$ are to their respective left and right. Segments which intersect holes are not splitting segments.

▶ **Lemma 17** (⋆). *Let P be a polygon without self-intersection, $\mathcal{H}$ a set of holes and $\mathcal{T}$ a triangulation of P. Then for every pair of triangles $T_1, T_2 \in \mathcal{T}$ with $T_1 \neq T_2$ there are at most $4|\mathcal{H}|^2$ different equivalence classes of splitting segments.*

▶ **Theorem 18** (⋆). *Given an instance $\mathcal{P}$ of BMLF with outer polygon $Q \in \mathcal{P}$, we can decide in time $O(kn^{2k+4})$ if a minimum link fencing $\mathcal{F}$ of $\mathcal{P}$ exists, in which every fence contains at most two polygons, each fence in the fencing has at most k segments, and n is the number of corners in $\mathcal{P}$.*

Proof sketch. For each polygon and for each pair of polygons we compute a minimum-link fence. Let λ_{uv} be the number of links for a minimum link fence containing $P_u, P_v \in \mathcal{P}$ and λ_u the number of links for a minimum link fence containing only $P_u \in \mathcal{P}$. Consider a complete graph G containing one vertex u for each polygon $P_u \in \mathcal{P}$ and one more vertex x if $|\mathcal{P}|$ is odd. Set the edge-weights $w(u, v) = \min\{\lambda_{uv}, \lambda_u + \lambda_v\}$ and $w(x, u) = \lambda_u$ for $P_u, P_v \in \mathcal{P}$. If for some $P_u \in \mathcal{P}$ or pair $P_u, P_v \in \mathcal{P}$ no fence with $\leq k$ segments exists we remove that edge. Computing a minimum weight perfect matching in this graph yields a minimum link fencing.

It remains to compute the minimum-link fences for each polygon and for each pair of polygons of $\mathcal{P}$. We consider a triangulation $\mathcal{T}$ of the free space of $\mathcal{P}$. For one single polygon we construct a plane loop around it by just traversing the incident triangles in the triangulation. To compute the minimum link fence for a pair of polygons in $\mathcal{P}$ we need to do more work. Since $\mathcal{T}$ contains only $O(n)$ triangles we can branch over the $O(n^k)$ ordered k-tuples of triangles. Moreover, by Lemma 17 we can branch over the $O(n^{2k})$ different splitting segments. If for our choice of triangles all splitting segments between consecutive triangles exist we construct a plane loop α if possible or otherwise reject the branch.

Let $T_1, \ldots, T_k$ be the chosen k-tuple of triangles and $l_1, \ldots, l_k$ the splitting segments. If none of the splitting segments intersect the same triangles in between two consecutive triangles T_i and T_{i+1} this is straight forward. If there are triangles that are intersected multiple times we have to evaluate $2^{O(k)}$ choices of how to resolve the self-crossings such a repetition induces for the loop α. For each valid choice we apply Theorem 15.

These are only $O(kn^{2k+4})$ choices in total and computing a minimum weight perfect matching can be done in $O(V^2E)$ time (with V being the number of vertices and E the

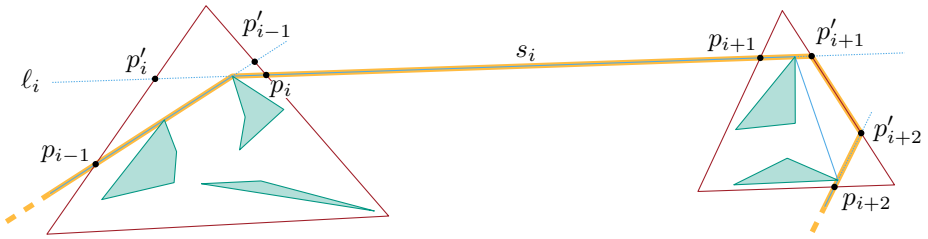


Figure 9 Computing a new fence (orange) from the old fences (purple) and the convex hull (blue).

number of edges) via finding a maximum weight perfect matching (e.g. [7]) on the same graph with edge weights set to maximum edge-weight plus one minus the original edge-weight. ◀

4 An algorithm for two-colored CMLF

In this section we present an algorithm for solving two-colored CMLF. Computing a minimum-link fence in this setting can be done by computing a fence for the convex hull of the contained polygons with the algorithm by Wang [13] which runs in time $O(n \log n)$ with n being the number of corners of the contained polygons. Throughout this section an instance of CMLF is given as $(\mathcal{P}, Q)$ where Q is the outer polygon and $\mathcal{P}$ is the set of polygons contained in Q.

▶ **Lemma 19.** *Given an instance $(\mathcal{P}, Q)$ of two-colored CMLF, let $\mathcal{F}$ be a solution for $(\mathcal{P}, Q)$. There exists a solution $\mathcal{F}'$ for the two-colored CMLF instance $(CH(\mathcal{P}), Q)$ with $|F| = |F'|$.*

Proof. As F is a minimum-link fencing of $(\mathcal{P}, Q)$, it suffices to consider the case where a minimal link fencing of $(CH(\mathcal{P}), Q)$ has strictly more segments than $|F|$. We will construct a new fence F° from this instance. Let $(p_1, \ldots, p_z)$ be the intersection points between F and $CH(\mathcal{P})$ ordered as they appear in a clockwise traversal of the convex hull, and observe that z is even. Let p_i, p_{i+1} be pairs of intersection points between F and $CH(\mathcal{P})$ such that the straight-line segment s_i connecting p_i and p_{i+1} lies on $CH(\mathcal{P})$ and completely outside of F (see Figure 9). Consider the supporting line ℓ_i of s_i. If the fence lies completely in one of the closed half-planes bounded by ℓ_i we add s_i to F°. Assume this is not the case. As s_i is on $CH(\mathcal{P})$ we get that ℓ_i does not intersect any polygon in $\mathcal{P}$. Moreover, as $\mathcal{F}$ consists of closed simple polygons we find two intersection points p'_i and p'_{i+1} that lie on ℓ_i, s.t., the parts of F appearing in a clockwise traversal from p'_i to p_i, as well as the ones in a clockwise traversal from p_{i+1} to p'_{i+1} lie outside of $CH(\mathcal{P})$. We add the segment s'_i between p'_i and p'_{i+1} to F°. Doing this for every pair of intersections we obtain a set of segments F°, where all segments are on the convex-hull of $\mathcal{P}$. Note that it is possible for these segments to intersect; if that is the case we only keep the parts until their intersection point. Finally, the start and end-points of connected chains of segments in F° lie on segments of fences in $\mathcal{F}$. We can convert F° into a fence of $CH(\mathcal{P})$ by connecting these endpoints along the fences in $\mathcal{F}$ and that fence will be disjoint from $\mathcal{P}$ (except possibly touching $\mathcal{P}$ in corner points).

It remains to argue that indeed $|F^\circ| \leq |F|$. We partition F° into two categories, segments that coincide with segments in F and segments that do not. Each of them is either a full segment of F or originates from the intersection of at most two different s'_i's and a segment of F. Furthermore, we add $z/2$ segments s'_i that are not sub-segments of segments in F. For each such s'_i we find at least one segment of F for which we did not add any sub-segment to F°. These are the segments of F on which p_i and p_{i+1} lie or that are fully outside of F°. ◀

▶ **Theorem 20.** *Two-colored CMLF can be solved in time $O(n \log n)$ where n is the number of corners of polygons in $\mathcal{P}$.*

5 Conclusion

We have shown BMLF to be NP-hard even if every fence contains at most three polygons, each fence has at most five links, and only two different colors of polygons are present. Our reduction holds regardless of requiring disjoint fences or not. Note, that our reduction can be adapted to not require the outer bounding polygon Q. Instead, we can replace Q by one polygon with a narrow and very complex channel, connecting the "inside" with the "outside". On the algorithmic side, we gave an XP-algorithm for BMLF parameterized by the maximum number of links in a fence and allowing at most two polygons per fence. We also showed that two-colored CMLF can be solved in polynomial time.

It is open if one can eliminate the exponential dependency on the number of links in our algorithm for BMLF. Furthermore, while our reduction holds when replacing the outer bounding polygon, our algorithm does not since we cannot immediately apply Theorem 15. Similarly, requiring the fences to be disjoint for BMLF is an interesting open direction.

References

1 Mikkel Abrahamsen, Anna Adamaszek, Karl Bringmann, Vincent Cohen-Addad, Mehran Mehr, Eva Rotenberg, Alan Roytman, and Mikkel Thorup. Fast fencing. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 564–573, 2018. doi:10.1145/3188745.3188878.

2 Mikkel Abrahamsen, Panos Giannopoulos, Maarten Löffler, and Günter Rote. Geometric multicut. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.9.

3 Alok Aggarwal, Heather Booth, Joseph O'Rourke, Subhash Suri, and Chee-Keng Yap. Finding minimal convex nested polygons. *Information and Computation*, 83(1):98–110, 1989.

4 Sujoy Bhore, Fabian Klute, Maarten Löffler, Martin Nöllenburg, Soeren Terziadis, and Anaïs Villedieu. Minimum link fencing, 2022. arXiv:2209.14804.

5 Gautam Das. *Approximation schemes in computational geometry*. PhD thesis, University of Wisconsin, Madison, 1991.

6 Peter Eades and David Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognition Letters*, 14(9):715–718, 1993. doi:10.1016/0167-8655(93)90140-9.

7 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

8 Michael R Garey, Ronald L Graham, and David S Johnson. The complexity of computing steiner minimal trees. *SIAM journal on applied mathematics*, 32(4):835–859, 1977. doi:10.1137/0132072.

9 Subir Kumar Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12(1):75–95, 1991. doi:10.1016/0196-6774(91)90024-S.

10 John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4:63–97, 1994. doi:10.1016/0925-7721(94)90010-8.

11 Klaus Jansen and Haiko Müller. The minimum broadcast time problem for several processor networks. *Theoretical Computer Science*, 147(1-2):69–85, 1995. doi:10.1016/0304-3975(94)00230-G.

12 Joseph SB Mitchell and Subhash Suri. Separation and approximation of polyhedral objects. *Computational Geometry: Theory and Applications*, 5(2):95–114, 1995. doi:10.1016/0925-7721(95)00006-U.

13 Cao An Wang. Finding minimal nested polygons. *BIT Computer Science section*, 31(2):230–236, 1991. doi:10.1007/BF01931283.

14 Cao An Wang and Edward P. F. Chan. Finding the minimum visible vertex distance between two non-intersecting simple polygons. In *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry, Yorktown Heights, NY, USA, June 2-4, 1986*, pages 34–42. ACM, 1986. doi:10.1145/10515.10519.

Multi-Robot Motion Planning for Unit Discs with Revolving Areas

Pankaj K. Agarwal
Duke University, Durham NC, USA

Tzvika Geft
Tel Aviv University, Israel

Dan Halperin
Tel Aviv University, Israel

Erin Taylor
Duke University, Durham, NC, USA

Abstract

We study the problem of motion planning for a collection of n labeled unit disc robots in a polygonal environment. We assume that the robots have *revolving areas* around their start and final positions: that each start and each final is contained in a radius 2 disc lying in the free space, not necessarily concentric with the start or final position, which is free from other start or final positions. This assumption allows a *weakly-monotone* motion plan, in which robots move according to an ordering as follows: during the turn of a robot R in the ordering, it moves fully from its start to final position, while other robots do not leave their revolving areas. As R passes through a revolving area, a robot R' that is inside this area may move within the revolving area to avoid a collision. Notwithstanding the existence of a motion plan, we show that minimizing the total traveled distance in this setting, specifically even when the motion plan is restricted to be weakly-monotone, is APX-hard, ruling out any polynomial-time $(1+\varepsilon)$-approximation algorithm.

On the positive side, we present the first constant-factor approximation algorithm for computing a feasible weakly-monotone motion plan. The total distance traveled by the robots is within an $O(1)$ factor of that of the optimal motion plan, which need not be weakly monotone. Our algorithm extends to an online setting in which the polygonal environment is fixed but the initial and final positions of robots are specified in an online manner. Finally, we observe that the overhead in the overall cost that we add while editing the paths to avoid robot-robot collision can vary significantly depending on the ordering we chose. Finding the best ordering in this respect is known to be NP-hard, and we provide a polynomial time $O(\log n \log\log n)$-approximation algorithm for this problem.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases motion planning, optimal motion planning, approximation, complexity, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.35

Related Version *Full Version*: https://arxiv.org/abs/2210.00123

Funding Work by Pankaj K. Agarwal and Erin Taylor is supported by IIS-1814493, CCF-2007556, and CCF-2223870. Work by Dan Halperin and Tzvika Geft has been supported in part by the Israel Science Foundation (grant no. 1736/19), by NSF/US-Israel-BSF (grant no. 2019754), by the Israel Ministry of Science and Technology (grant no. 103129), by the Blavatnik Computer Science Research Fund, and by the Yandex Machine Learning Initiative for Machine Learning at Tel Aviv University.

1 Introduction

Multi-robot systems are already in use in logistics, in a variety of civil engineering and nature preserving tasks, and in agriculture, to name a few areas. They are anticipated to proliferate in the coming years, and accordingly they attract intensive research efforts in diverse communities.

A basic motion-planning problem for a team of robots is to plan such collision-free paths for the robots between given start and final positions. Among the many dimensions along which the multi-robot motion planning (MRMP) problem has been studied, we focus on three: (1) we distinguish between distributed and centralized control. In the former each robot has limited knowledge of the entire environment where the robots move, and each robot may communicate with few neighboring robots. In the latter, which is typical in factory automation and other well-structured environments, a central authority has control over all the robots and the planning for each robot takes into consideration knowledge about the state of all the other robots in the system. (2) In the *labeled* version the robots are distinguishable from one another and each robot has its own assigned target, whereas in the *unlabeled* version the robots are indistinguishable, i.e., each target can be occupied by any robot in the team and the motion-planning problem is considered solved if at the end of the motion all the target positions are occupied. (3) We further distinguish between *continuous* or *discrete* domains. Much of the study of motion planning in computational geometry and robotics assumes that the workspace is *continuous*. In AI research, where the problem is typically called multi-agent path finding (MAPF) [21], the domain is modeled as a graph. Nowadays the MAPF problem is studied in diverse research communities, often as an approximation of the continuous domain.

In our study here we consider a *centralized, labeled*, and *continuous* version of MRMP. Furthermore, we are not only interested in finding a solution to the given motion-planning problem, but rather in finding a high-quality solution. Specifically, we aim to find a solution that minimizes the total path length traveled by the robots.

Related Work. Computing a feasible motion plan (not necessarily a good one) itself is in general computationally hard for MRMP (see, e.g., [4, 9, 11, 18]). In the results that we cite next, some additional mitigating conditions are assumed on the system to obtain efficient motion-planning algorithms.

There are few results that guarantee bounds on the quality of the motion plans for multi-robot systems. For complete algorithms[1] in the unlabeled case, there are bounds on the length of the longest path taken by a robot in the system [22], or on the sum of distance traveled by all the robots [20]. For the labeled case, Demaine et al. [7] provide constant-factor approximation algorithms for minimizing the execution time of a coordinated parallel motion if there are no obstacles. Still for the labeled case, Solomon and Halperin obtained a very crude bound on the sum of distances [17] (the approximation factor can be linear in the complexity of the environment in the worst case) in a setting identical to the setting of the current paper, namely assuming the existence of *revolving areas* – see below for a formal definition. No sublinear approximation algorithm is known for MRMP even if we assume the existence of revolving areas and the cost of a motion plan is the sum of the lengths of individual paths. In the current paper we significantly improve over and expand the results in [17] in several ways, as we discuss below.

[1] A motion planning algorithm is called *complete* if, in finite time, it is guaranteed to find a solution or determine that no solution exists.

An alternative approach to cope with the hardness of motion planning is to use *sampling-based* methods [14]. In their seminal paper, Karaman and Frazzoli [12] (see also [19]) introduced an algorithm, called RRT*, which guarantees near optimality if the number of samples tends to infinity. A related algorithm dRRT* handles the multi-robot case with the same type of guarantee [16]. Recently Dayan et al [6] have obtained near-optimality with finite sample size for the multi-robot case.

Problem Statement. Let $\mathcal{W}$ be a polygonal environment, that is, a polygon with holes in $\mathbb{R}^2$ and a total of m vertices. Let $R_1, \ldots, R_n$ be n robots, each modeled as a unit disc, that move around in $\mathcal{W}$. Let $\mathcal{O} = \mathbb{R}^2 \setminus \mathcal{W}$ be the obstacle space. For a point $p \in \mathbb{R}^2$, let D_p denote the unit disc centered at point p. Let $\mathcal{F} = \{x \in \mathcal{W} : D_x \cap \mathcal{O} = \emptyset\}$ represent the free space of $\mathcal{W}$ (with respect to one R_i). A *path* is a continuous function $\pi : I \to \mathbb{R}^2$ from an interval I to $\mathbb{R}^2$, and is *collision-free* if it is contained in $\mathcal{F}$. Let $\ell(\pi)$ denote the arc length of π, i.e., $\ell(\pi) = \int_I |\pi'(t)|dt$. The position of each R_i is specified by the x- and y-coordinates of its center c_i and we use $R_i(c)$ to denote R_i being at c (note that $R_i(c)$ is the same as D_c), and a motion of R_i is specified by the path followed by its center. Let int D denote the interior of disc D. A *path ensemble* $\Pi = \{\pi_1, \ldots, \pi_n\}$ is a set of n paths defined over a common interval I, i.e. $\pi_i : I \to \mathbb{R}^2$, for $1 \leqslant i \leqslant n$; Π is called *feasible* if (i) $\pi_i \subset \mathcal{F}$ for every $i \leqslant n$, and (ii) for any $t \in I$ and for any pair $i \neq j$, int $R_i(\pi_i(t)) \cap$ int $R_j(\pi_j(t)) = \emptyset$, i.e., the R_i's remain in $\mathcal{W}$ and they do not *collide* with each other (but may touch each other) during the entire motion. We also refer to Π as a *motion plan* of $R_1, \ldots, R_n$. The cost of Π, denoted by $\mathfrak{c}(\Pi)$, is defined as $\mathfrak{c}(\Pi) = \sum_{i=1}^{n} \ell(\pi_i)$.

We are given a set of *start* positions $\mathbf{s} = \{s_1, \ldots, s_n\}$ where the n robots initially lie and a set of *final* (also called *target*) positions $\mathbf{f} = \{f_1, \ldots, f_n\}$. Our goal is to find a path ensemble $\Pi^* = \{\pi_1^*, \ldots, \pi_n^*\}$ over an interval $[0, T]$ where T denotes the ending time of the last robot movement,

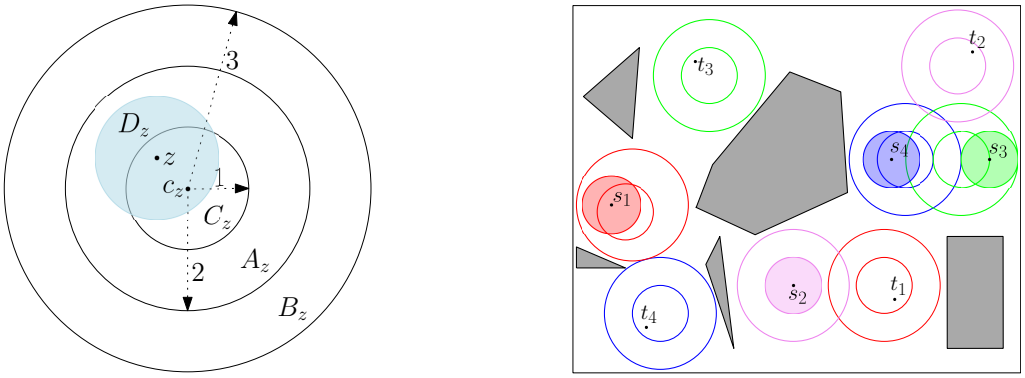
(i) $\pi_i^*(0) = s_i$ and $\pi_i^*(T) = f_i$ for all i, and

(ii) $\mathfrak{c}(\Pi^*) = \min_{\Pi} \mathfrak{c}(\Pi)$ where the minimum is taken over all feasible path ensembles.

We refer to the problem as *optimal multi-robot motion planning (MRMP)*. In this paper, we investigate optimal MRMP under the assumption that there is some free space around the starting and final positions of $R_1, \ldots, R_1$, a formulation introduced in [17]. A *revolving area* of a start or final position $z \in \mathbf{s} \cup \mathbf{f}$, is a disc A_z of radius 2 such that: (i) $D_z \subseteq A_z$, (ii) $A_z \cap \text{int}(\mathcal{O}) = \emptyset$, and (iii) for any other start or final position $y \in (\mathbf{s} \cup \mathbf{f}) \setminus \{z\}$, $A_z \cap \text{int}(D_y) = \emptyset$. That is, each R_i lies in a revolving area at its start and final position (note that z need not be the center of the revolving area A_z) and does not intersect any other revolving areas, and the revolving areas do not intersect any obstacles. We remark that the revolving areas may intersect one another; this makes the separation assumptions in the current paper lighter than in related results (e.g., [1]), which in turn makes the analysis more involved. See Figure 1 for an example. Set $\mathcal{A} = \{A_z : z \in \mathbf{s} \cup \mathbf{f}\}$. We refer to this problem as *optimal multi-robot motion planning with Revolving Areas (MRMP-RA)*.

We define the *active interval* $\tau_i \subseteq [0, 1]$ as the open interval from the first time R_i leaves the revolving area A_{s_i} of s_i to the last time R_i is not in the revolving area A_{f_i} of f_i. If the active intervals $\{\tau_1, \ldots, \tau_n\}$ are pairwise disjoint then we call Π a *weakly-monotone motion plan* (with respect to revolving areas).[2] Finally, an instance of optimal MRMP is specified as $\mathcal{I} = (\mathcal{W}, n, \mathbf{s}, \mathbf{f})$ where n is the number of moving robots and $\mathcal{W}, \mathbf{s}, \mathbf{f}$ are as defined above. Let $\Pi^*(\mathcal{I})$ denote an optimal solution of $\mathcal{I}$ and let $\mathfrak{c}^*(\mathcal{I}) = \mathfrak{c}(\Pi^*(\mathcal{I}))$.

[2] We use the term "weakly-monotone" because a plan is called monotone if the active interval of R_i is defined from when R_i leaves s_i the first time and reaches f_i the last time, (rather than the leaving/reaching the revolving area A_{s_i}/A_{f_i}).



(a) (b)

Figure 1 (a) On the left, revolving area A_z for some $z \in \mathbf{s} \cup \mathbf{f}$ with $D_z \subseteq A_z$, core C_z, and buffer B_z. (b) On the right, we show an instance of MRMP-RA. Each robot is shown as a filled disc in its starting revolving area, and its target revolving area is shown in the same color. Obstacles are dark gray.

Our Results. The paper contains the following three main results:

(A) Hardness results. In Section 2, we show that MRMP-RA is NP-hard under the weakly-monotone assumption. The NP-hardness of optimizing sum of distances (i.e., optimal MRMP) for the monotone and the general (non-monotone) case was shown in [10], but without revolving areas. Our main result here is the extension of the NP-hardness construction to prove that MRMP-RA, under the weakly-monotone assumption, is in fact APX-hard, which rules out a polynomial-time $(1+\varepsilon)$-approximation algorithm for it. To the best of our knowledge, this is the first APX-hardness result for any MRMP variant.

(B) Approximation algorithm. In Section 3, we present the first $O(1)$-approximation algorithm that given an instance $\mathcal{I} = (\mathcal{W}, \mathbf{s}, \mathbf{f}, \mathcal{A})$ of MRMP-RA computes a feasible path ensemble Π from $\mathbf{s}$ to $\mathbf{f}$ such that Π is weakly-monotone and $\mathscr{c}(\Pi) = O(1) \cdot \mathscr{c}^*(\mathcal{I})$; note that $\Pi^*(\mathcal{I})$ need not be weakly-monotone, i.e., we approximate the general optimal path ensemble. In fact, we show that the robots can be moved in any order, so our algorithm can be extended to an online setting where the robots R_i, and their start/final positions, (s_i, f_i) are given in an online manner, or R_i's may have to execute multiple tasks which are given in an online manner– the so-called life-long planning problem. Our algorithm ensures an $O(1)$ competitive ratio, i.e., the cost is $O(1)$ times the optimal cost of the offline problem.

The algorithm begins by computing a set of shortest paths Γ that avoid obstacles but ignore robot-robot collisions. Then, Γ is edited to avoid robot-robot collisions by moving non-active robots within their revolving areas. Our overall approach is the same as by Solomon and Halperin [17], but the editing of Γ differs significantly from [17], so that the cost of the paths does not increase by too much. We use a more conservative editing of Γ, which enables us to prove that the cost of the edited path ensemble is $O(1) \cdot \mathscr{c}(\Gamma)$ (see Section 4), while the cost of the edited path in [17] is[3] $O(\mathscr{c}(\Gamma) + mn + m^2)$. Our main technical contributions are defining a more conservative retraction, proving that the motion plan remains feasible even under this conservative retraction, and bounding the total cost of the motion plan by using a combination of local and global arguments. Analyzing both the feasibility and the cost of the motion plan are nontrivial and require new ideas.

[3] Notice that the roles of m and n here are reversed with respect to [17].

(C) Computing a good ordering. The result above shows that editing the paths increases the total cost of the motion plan only by a constant factor irrespective of the order in which we move the robots. However, the overhead in the overall cost due to editing (to avoid robot-robot collisions) can vary significantly depending on the ordering we chose. This raises the question whether we can find a "good" ordering that minimizes the overhead. The result in [17] implies that the problem of finding a good ordering that minimizes the amount of overhead is NP-hard.[4] We present a polynomial time $O(\log n \log\log n)$-approximation algorithm for finding a good ordering. This is achieved by reducing the problem to an instance of weighted feedback arc set in a directed graph, and applying an approximation algorithm for the latter problem [8]. This result is described in Section 5.

We emphasize that without additional, mitigating, assumptions, MRMP is intractable. Sampling-based planners assume that the full solution paths have some clearance around them – namely, each robot has some distance from the obstacles along its entire path, as well as from the other robots. Here, we assume certain clearance only at the start and goal positions; we do not make any assumption about the clearance along the paths. Indeed, we assume non-negligible clearance, as we require that each robot at a start or goal position is encapsulated inside a disc of radius 2, which does not contain any other robot at its start or goal position. The choice of the number 2 here is not arbitrary. In a couple of related results for MRMP of unit discs [1, 2] this is the critical value of clearance below which there does not always exist a solution to the problem.
Due to length constraints of the paper, some proofs have been moved to the appendix.

2 Hardness of Distance Optimal MRMP-RA

In this section we present our hardness results. Throughout this section all path ensembles are weakly-monotone, unless otherwise stated. With a slight abuse of notation we use $\mathfrak{c}^*$ to denote the cost of the optimal weakly-monotone path ensemble. Finding monotone path ensembles has been shown to be NP-hard in [9] using a similar grid-based construction without revolving areas.

NP-Hardness of weakly-monotone MRMP-RA. Let $Q(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m} C_i$ be an instance of 3SAT with n variables and m clauses. Each clause C_i is a disjunction of three *literals*, which are variables or their negations. We construct a corresponding MRMP-RA instance $\mathcal{I} := \mathcal{I}(Q) = (\mathcal{W}, \mathbf{s}, \mathbf{f}, \mathcal{A})$ with $N = 3m + 1$ robots and choose a real value $d \geq 0$ such that $\mathfrak{c}^*(\mathcal{I}) \leq d$ if and only if Q is satisfiable. Let $d(\mathcal{I}) := \sum_{i=1}^{N} d_i$, where d_i is the length of the optimal path of R_i from s_i to f_i in $\mathcal{W}$, ignoring other robots. In fact, our construction will choose d to be $d(\mathcal{I})$, that is, d is the lowest possible cost of a feasible path ensemble from $\mathbf{s}$ to $\mathbf{f}$ in $\mathcal{W}$. Our construction will ensure that the lowest cost is attained if and only if Q is satisfiable. $\mathcal{I}$ is constructed so that a path ensemble with such a cost is possible if and only if (a feasible) monotone motion plan exists. An example of the construction is shown in Figure 2.

Overall description. The workspace $\mathcal{W}$ consists of $m + n$ rectangular gadgets, one for each variable and each clause, referred to as *variable* and *clause* gadgets, respectively. All the gadgets have unit-width *passages* that are wider around revolving areas. For simplicity, the

[4] The model in [17] for defining the overhead is different from ours, their construction can nevertheless be adapted to our setting.

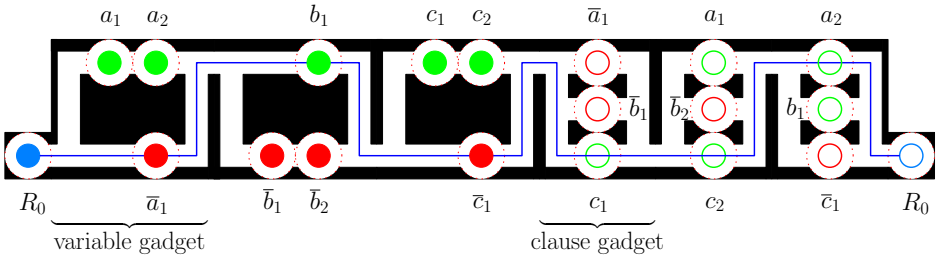


Figure 2 The MRMP-RA instance $\mathcal{I}$ that corresponds to the formula $Q = (\overline{a} \vee \overline{b} \vee c) \wedge (a \vee \overline{b} \vee c) \wedge (a \vee b \vee \overline{c})$. The start and target positions are the filled and unfilled discs, respectively. Positive literal robots are green, negative literal robots are red. Obstacles appear in black. Start and target positions of literal robots are labeled with unique indices in order to distinguish between appearances of the same literal. The path π_0 is shown in blue for the assignment $a = T, b = F, c = T$ for which the corresponding path ensemble has robots moving in the following order: $\overline{c}_1, b_1, \overline{a}_1, r_0, a_2, a_1, \overline{b}_2, c_2, \overline{b}_1, c_1$.

widened areas are shown as circular arcs, but they can easily be made polygonal. Each gadget has an entrance on the left and an exit on the right. The vertical positions of entrances and exits alternate so that a gadget's entrance is connected to the exit of the gadget on its left.

There are $N = 3m + 1$ robots, each being a unit disc: one robot for each appearance of a literal in Q, which are collectively called *literal robots*, and one special *pivot* robot R_0 (shown in blue in Figure 2). The robot R_0 has to pass through all the gadgets from left to right, by which it is able to verify the satisfiability of Q, and the literal robots will constrain its motion in order to ensure that $c^*(\mathcal{I}) \leq d$.

Each variable (resp.clause) gadget contains two (resp. three) horizontal passages, which offer two (resp. three) shortest paths from its entrance to its exit. Each such path consists of vertical and horizontal line segments. The horizontal passages of the gadgets contain all the start and target positions of literal robots. All the revolving areas are centered at their respective start or target positions, and they do not overlap.

Gadgets. Each variable gadget initially contains robots representing literals of a single variable of Q. The top and bottom horizontal passages of the gadget contain robots representing only positive and negative literals, respectively. Each clause gadget has three horizontal passages, each containing a target position of one the literals in the corresponding clause. The gadgets are placed within a horizontal strip from left to right such that variable gadgets are located to the left of clause gadgets. The order of gadgets of the same type is arbitrary, however it determines the order of the start positions, which is critical: the left to right order of start positions within each variable gadget is set to match the left to right order of the corresponding target positions. We refer to this order as the *intra-literal order property*. We say that a revolving area A is *congested* if it contains two robots at the same time. Intuitively, both optimal path ensembles and monotone path ensembles need to prevent revolving areas from becoming congested. The following lemma is proved in Appendix A.2. We first establish that finding an optimal weakly-monotone path ensemble is equivalent to finding a monotone one, then show the equivalence between a satisfying assignment and a monotone path ensemble.

▶ **Lemma 1.** *$\mathcal{I}$ has a weakly monotone path ensemble with a cost of d if and only if $\mathcal{I}$ has a monotone path ensemble.*

▶ **Lemma 2.** *Q has a satisfying assignment if and only if $\mathcal{I}$ has a monotone path ensemble.*

Proof. Assume that Q has a satisfying assignment Λ. Let $\mathcal{R}^+$ (resp. $\mathcal{R}^-$) denote the set of robots corresponding to literals that evaluate to true (resp. false) according to Λ. That is, for each variable gadget, $\mathcal{R}^+$ contains robots that are all initially either in the top or the bottom passage, according to Λ. We show that the robots can move along optimal paths in the order $\mathcal{R}^-$, R_0, $\mathcal{R}^+$, which is made precise below.

Let π_0 be a shortest collision-free path from s_0 to f_0 that passes only through the start positions of $\mathcal{R}^-$ and targets of $\mathcal{R}^+$; see Figure 2. The path π_0 exists because each clause gadget must contain a target of some robot in $\mathcal{R}^+$, or else Λ does not satisfy Q.

In the path ensemble, each $R_i \in \mathcal{R}^-$ follows the subpath of π_0 from s_i (through which π_0 passes) up to the gadget containing f_i, from which R_i can reach its final position f_i using the shortest path. The order in which the robots in $\mathcal{R}^-$ move is the right to left order of their start positions, which guarantees no collision with another robot located at its start position. Since the robots in $\mathcal{R}^-$ move before $\mathcal{R}^+$, the targets through which π_0 passes are unoccupied when the robots in $\mathcal{R}^-$ move, guaranteeing no collisions at clause gadgets. Next, R_0 moves using π_0, which passes through empty passages at this point. Finally, each $R_i \in \mathcal{R}^+$ joins π_0 at the vertical passage to its right, from which point it continues similarly to $\mathcal{R}^-$. The order of motion of the robots in $\mathcal{R}^+$ is the right to left order of their targets, which guarantees no collisions in the clause gadgets. Note that due to the intra-literal order property we also have no interferences among $\mathcal{R}^+$ within variable gadgets.

For the other direction, let us assume that there is a monotone path ensemble for $\mathcal{I}$. Let π_0 denote the path taken by R_0. Without loss of generality, π_0 is weakly x-monotone. Specifically, it passes through only one horizontal passage in each variable gadget. Therefore, we define an assignment Λ as follows: x is assigned to be true if and only if π_0 goes through the bottom passage of x's variable gadget, which corresponds to negative literals. Let C be a clause of Q and let f_j be a target in C's clause gadget that is unoccupied during R_0's motion, which must exist. It is easy to verify that the literal corresponding to R_j is true according to Λ. Therefore, C is satisfied. ◀

The construction can be carried out in polynomial time, therefore by combining Lemma 1 and 2, we obtain the following:

▶ **Theorem 3.** *MRMP-RA for weakly-monotone path ensembles is NP-hard.*

Hardness of Approximation. We now show that MRMP-RA is APX-hard, ruling out any polynomial time $(1+\varepsilon)$-approximation algorithm. We first go over some definitions. For an MRMP-RA instance $\mathcal{I}$, we use $c^*(\mathcal{I})$ to denote the cost of the optimal weakly-monotone path ensemble for $\mathcal{I}$. For a 3SAT formula Q, let $\mathrm{SAT}(Q)$ denote the largest fraction of clauses in Q that can be simultaneously satisfied. We say that a revolving area A is *occupied* if it contains the robot whose start or target position lies in A.

To prove the hardness of approximation we present a gap-preserving reduction from MAX-3SAT(5), which is APX-hard [23]. The input to MAX-3SAT(5) is a 3SAT formula with 5 appearances for each variable and the goal is to find an assignment maximizing the number of satisfied clauses. Let Q be a MAX-3SAT(5) instance with n variables and m clauses and let $\mathcal{I} \coloneqq \mathcal{I}(Q)$ be the MRMP-RA instance resulting from the NP-Hardness reduction described above, which we slightly modify as follows. Instead of the single pivot robot R_0 in $\mathcal{I}$, we now have m pivot robots. To this end, we modify the construction so that there is a horizontal passage that extends to the left of s_0 in I. The passage is lengthened to accommodate m start positions that lie on the same horizontal line, passing through s_0

in I. Similarly, another such passage is created to the right of f_0 to accommodate m target positions. The left to right order of the start positions of the pivot robots is set to match the left to right order of the corresponding target positions. Let $\mathcal{I}'$ denote the resulting MRMP-RA instance.

▶ **Lemma 4.** *Let Q be a 3SAT formula such that for any assignment to Q there are at least k unsatisfied clauses in Q. Then $\mathfrak{c}^*(\mathcal{I}') > d(\mathcal{I}') + km$.*

Proof. Let us examine $\Pi^*(\mathcal{I}')$, an optimal path ensemble for $\mathcal{I}'$. We say that a robot R_i has a *bad event* during the execution of $\Pi^*(\mathcal{I}')$ when R_i traverses an occupied revolving area. Note that each bad event results in R_i having a path longer than $1+d_i$, d_i being the length of R_i's shortest possible path. We claim that each of the m pivot robots has k bad events, which suffices for proving the lemma.

Suppose to the contrary that one of the pivot robots, say R_i, has $q < k$ bad events. We will show how to obtain an assignment for Q where there are at most q unsatisfied clauses. Since $\Pi^*(\mathcal{I}')$ is optimal, π_i, the path taken by R_i, is weakly x-monotone. We define an assignment Λ as follows (the same way as in the second direction of the proof of Theorem 2): x is assigned to be true if and only if π_i goes through the bottom passage of x's variable gadget. In other words, Λ sets a literal to be true if and only if the corresponding literal-robot's starting position does not lie on π_i. Let us examine π_i right before it is R_i's turn to move. Let $\mathcal{R}$ denote the set of robots that are intersected by π_i and are located at variable gadgets at this point in time. We can assume without any loss of generality that $\mathcal{R}$ is empty. If it is not, then let us examine the path ensemble Π where the robots in $\mathcal{R}$ move to their targets before R_i's turn. The number of bad events for R_i can only decrease in Π. This holds because by having some $R_j \in \mathcal{R}$ move before R_i we eliminate a bad event (for R_i) in R_j's variable gadget and possibly introduce a bad event in R_j's clause gadget.

Since there are q bad events for R_i, there are at most q clause gadgets where such an event occurs. Therefore, to get a contradiction it suffices to show that all other clause gadgets correspond to clauses that are satisfied by Λ. Let C be such a clause, i.e., in the corresponding clause gadget π_i passes through some empty revolving area A_{f_j}. Since π_i does not pass through any occupied revolving areas in the variable gadgets, the corresponding start position s_j must not lie on π_i. Therefore, r_j corresponds to a literal that is true by Λ, and so C is satisfied. ◀

We now make $d(\mathcal{I}')$ explicit using an upper bound for an arbitrary d_i. First, we bound the length of each vertical segment in the corresponding path π_i by 10, which provides sufficient distance for our gadgets. Since each variable appears in Q five times, we bound the horizontal length of an variable gadget by $4 \cdot 2 + 3 = 11$ (i.e., there at most 4 revolving areas on a horizontal passage and some additional length). Therefore, the path length through any gadget is $O(1)$. Hence, we have $d_i = O(m)$ and the number of robots is also $O(m)$ (we have $m = 5n/3$). Therefore, we can set $d(\mathcal{I}') = cm^2$ for some sufficiently large constant c (we can easily lengthen paths in $\mathcal{I}'$ if that is needed for the bound). We can now combine the latter equality with Lemma 4 and the NP-Hardness reduction. Let us define $f(Q) \coloneqq d(\mathcal{I}')$.

▶ **Theorem 5.** *There is a polynomial time reduction that transforms an instance Q of MAX-3SAT(5) with m clauses to an MRMP-RA instance $\mathcal{I}'$ such that* $\mathrm{SAT}(Q) = 1 \Rightarrow \mathfrak{c}^*(\mathcal{I}') = f(Q) \leqslant cm^2$ *for some constant $c > 0$ and otherwise* $\mathrm{SAT}(Q) < \alpha \Rightarrow \mathfrak{c}^*(\mathcal{I}') > f(Q) + (1-\alpha)m \cdot m = \left(1 + \frac{1-\alpha}{c}\right) f(Q)$, *for all $0 < \alpha < 1$.*

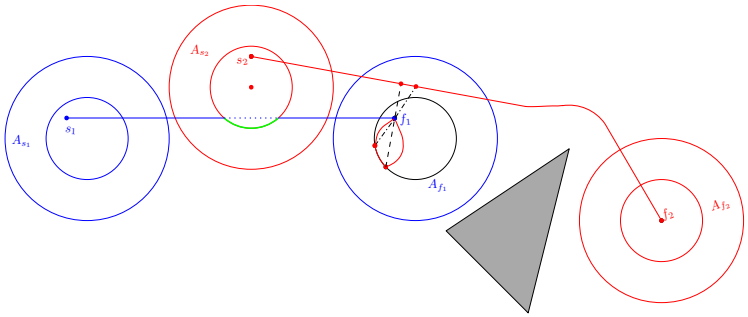


Figure 3 Path γ_1 is shown in blue from s_1 to f_1. Assume R_1 is active before R_2. In $\bar{\gamma}_1$, the dotted portion of the path is replaced with the green arc along ∂C_{s_2}. Path π_2 is shown in red from s_2 to f_2. R_1 must follow the red retraction during the movement of R_2 in B_{f_2}.

3 Algorithm

Let $\mathcal{I} = (\mathcal{W}, \mathbf{s}, \mathbf{f}, \mathcal{A})$ be an instance of MRMP-RA. Let n be the number of robots and m be the complexity of the environment W. We describe an $(n(n+m)\log m)$ algorithm for computing a weakly-monotone path ensemble $\tilde{\Pi} := \tilde{\Pi}(\mathcal{I})$ for $R_1, \ldots, R_n$ such that $\mathscr{C}(\tilde{\Pi}) = O(1) \cdot \mathscr{C}^*(\mathcal{I})$. We remark that $\tilde{\Pi}$ is weakly-monotone but $\Pi^*(\mathcal{I})$ need not be, i.e. $\tilde{\Pi}$ is an $O(1)$-approximation of any feasible motion plan. We parameterize the paths in $\tilde{\Pi}$ over the common interval $J = [0, n]$. We need a few definitions and concepts related to revolving areas. For any $z \in \mathbf{s} \cup \mathbf{f}$, let c_z denote the center of the revolving area A_z, and let C_z (resp. B_z) be the disc of radius 1 (resp. 3) centered at c_z, i.e., $C_z \subset A_z \subset B_z$. If $x \notin B_z$ then $D_x \cap A_z = \emptyset$. We refer to C_z and B_z as the *core* and *buffer*, respectively, of revolving area A_z. See Figure 1.

Overview of the Algorithm. The algorithm consists of three stages. We note that Stage (I) and (II) are used in [17]. However, Stage (III) differs significantly from previous work in order to ensure the total cost of paths is within an $O(1)$ factor of that of the optimal motion plan. We describe all stages for completeness.

I. We compute the free space $\mathcal{F}$ (with respect to one robot) using the algorithm of Ó'Dúnlaing and Yap [13, 24]. If s_i and f_i, for some $i \in [n] := \{1, 2, \ldots, n\}$, do not lie in the same connected component of $\mathcal{F}$, then a feasible path does not exist for R_i from s_i to f_i. Therefore, we stop and return that no feasible motion plan exists from $\mathbf{s}$ to $\mathbf{f}$. Next, for each i, we compute a shortest path γ_i from s_i to f_i, ignoring other robots using the algorithm of Chen and Wang [5]. Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be the path ensemble computed by the algorithm.
Although Γ does not intersect $\mathcal{O}$, it may not be feasible since two robots may collide during the motion. The next two steps deform Γ to convert it into a feasible motion plan. We take an arbitrary permutation σ of $[n]$. Without loss of generality assume $\sigma = \langle 1, 2, \ldots, n \rangle$. We say that R_i is *active* during the subinterval $[i-1, i]$ of $J := [0, n]$, during which it moves from s_i to f_i. During $[0, i-1]$ (resp. $[i, n]$) R_i only moves within the revolving area A_{s_i} (resp. A_{f_i}).

II. For each i, we first modify γ_i, as described below in Section 3.1, so that it does not intersect the interior of the core C_j of any revolving area A_j that is occupied by a robot R_j, for $j \neq i$; see Figure 3. Let $\overline{\gamma}_i$ be the deformed path. Abusing the notation a little, let $\overline{\gamma}_i : [i-1, i] \to \mathcal{F}$ denote a uniform parameterization of the path $\overline{\gamma}_i$, i.e. R_i moves with a fixed speed during $[i-1, i]$ from s_i to f_i along $\overline{\gamma}_i$. We extend $\bar{\gamma}_i$ to the interval $[0, n]$ by setting $\bar{\gamma}_i(t) = s_i$ for $t \in [0, i-1]$ and $\bar{\gamma}_i(t) = f_i$ for $t \in [i, n]$. Set $\bar{\Gamma} = \{\bar{\gamma}_1, \ldots, \bar{\gamma}_n\}$.

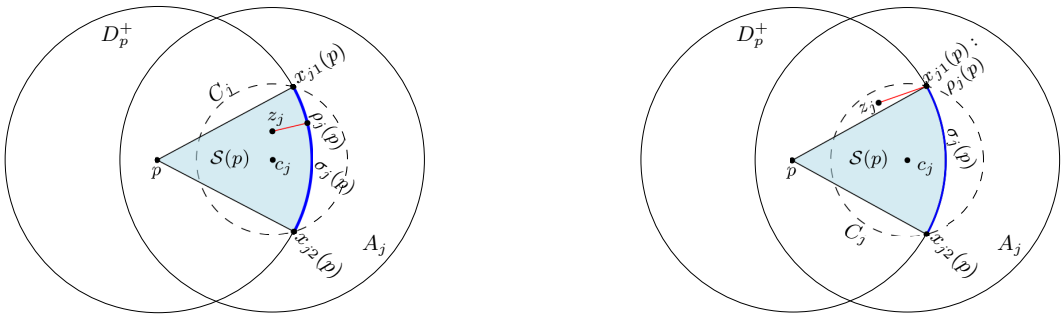


Figure 4 Retraction Map. (a) A sector type retraction (when z_j lies in $S(p)$); (b) an intersection type retraction, z_j lies outside $S(p)$, the retraction point is $x_{j1}(p)$.

III. Next, for each distinct pair $i, j \in [n]$, we construct a *retraction map* $\rho_{ij} : \mathcal{F} \to \mathcal{F}$ that specifies the position of R_j for a given position of R_i during the interval $[i-1, i]$ when R_i is active so that R_i and R_j do not collide as R_i moves along $\overline{\gamma}_i$. The retraction map ensures that R_j stays within the revolving area A_{s_j} (resp. A_{f_j}) for $j < i$ (resp. $j > i$), and it does not collide with any R_k for $k \neq i, j$, as well. See Figure 3. Using this retraction map, we construct the path $\pi_j : J \to \mathcal{F}$ as follows: $\pi_j(t) = \begin{cases} \rho_{ij}(\overline{\gamma}_i(t)) & \text{for } t \in [i-1, i] \text{ and } j \neq i, \\ \overline{\gamma}_j(t) & \text{for } t \in [j-1, j]. \end{cases}$
We prove below that each π_j is a continuous path. In Section 4, we prove that $\Pi = \{\pi_1, \ldots, \pi_n\}$ is a feasible path ensemble with $\mathfrak{c}(\Pi) = O(1) \cdot \mathfrak{c}^*(\mathcal{I})$.

3.1 Modifying path γ_i

Fix an $i \in [n]$. For $j < i$, let $z_j = f_j$ and for $j > i$, let $z_j = s_j$. Set $Z = \{z_j : 1 \leqslant j \neq i \leqslant n\}$. This step modifies γ_i to ensure that the path of R_i does not enter the core C_z of any $z \in Z$.

Fix a $z \in Z$. If $\gamma_i \cap C_z = \emptyset$, then R_j does not affect γ_i. If $\gamma_i \cap C_z \neq \emptyset$, then we modify γ_i as follows: let p_z, q_z be the first and last intersection points of γ_i and C_z along γ_i, respectively. Let Q_z be the shorter arc of ∂C_z, the boundary of the core C_z, between p_z and q_z. We replace $\gamma_i[p_z, q_z]$ with Q_z. We repeat this step for all $z \in Z$. Let $\overline{\gamma}_i$ be the resulting path from s_i to f_i; $\overline{\gamma}_i$ does not intersect $\text{int}(C_z)$ for any $z \in Z$. Note that C_{z_j}'s are pairwise disjoint, and that γ_i is a shortest path from s_i to f_i in $\mathcal{F}$, therefore $\gamma[p_z, q_z]$ and $\gamma[p_{z'}, q_{z'}]$, for any pair $z, z' \in Z$, are disjoint. We can thus process Z in an arbitrary order and the resulting path does not depend on the ordering. Furthermore $C_z \subseteq \mathcal{F}$ (since $A_z \subseteq \mathcal{W}$), so $\bar{\gamma}_i \subset \mathcal{F}$ for all i.

3.2 Retracting a robot R_j

We now describe the retraction motion of R_j when R_i is active, so that they do not collide. Note that for all $t \in [i-1, i]$, $\bar{\gamma}_j(t) = z_j$, i.e., before applying the retraction R_j is at z_j when R_i is active. We define the *retraction function* $\rho_{ij} : \mathbb{R}^2 \to \mathbb{R}^2$ that specifies the motion of R_j within A_j during $[i-1, i]$. Since i is fixed, for simplicity we use ρ_j to denote ρ_{ij}, and we use C_j (resp. A_j, B_j) for disc C_{z_j} (resp. A_{z_j}, B_{z_j}). If the center of R_i is at distance at least 2 from z_j, then R_i does not intersect D_j, so there is no need to move R_j from z_j. Therefore we set $\rho_j(p) = z_j$ for all $p \in \pi_i$ such that $\|p - z_j\| \geqslant 2$. On the other hand, $\overline{\gamma}_i$ does not intersect the interior of C_j so $\rho_j(p)$ is undefined for $p \in \text{int}(C_j)$. We thus focus on the case when $\|p - z_j\| \leqslant 2$, in which case p lies in the buffer disc B_j, and $p \notin \text{int}(C_j)$, i.e., $p \in B_j \setminus \text{int}(C_j)$.

Let $D^+(p)$ be the disc of radius 2 centered at p. Note that for a point $q \in \mathbb{R}^2$, $\text{int}(D_p) \cap \text{int}(D_q) \neq \emptyset$ if and only if $q \in D^+(p)$. Intuitively, we move the center of R_j from z_j (within C_{z_j}) as little as possible so that R_j does not collide with $R_i(p)$. Formally, we define ρ_j as: $\rho_j(p) = \text{argmin}_{q \in C_{z_j} \setminus D_p^+} \|q - z_j\|$ if $p \notin \text{int}(C_{z_j})$, and undefined otherwise.

In the remainder of the discussion, we assume $\|z_j - p\| \leqslant 2$ and $p \notin C_j$, so $p \in B_j \setminus C_j$. Therefore, $\rho_j(p)$ exists and additionally $\rho_j(p)$ is unique. We now discuss the two possible types of retraction. Refer to Figure 4 throughout this paragraph. Note that ∂C_j and ∂D_p^+ intersect at exactly two points since $p \in B_j \setminus C_j$, say, $x_{j1}(p), x_{j2}(p)$. Let $\sigma_j(p)$ be the smaller of the two arcs of ∂D_p^+ induced by $x_{j1}(p)$ and $x_{j2}(p)$, and let $S(p) = \text{conv}(\sigma_j(p) \cup \{p\}) \subseteq D_p^+$ be the *sector* of $D^+(p)$ induced by $x_{j1}(p)$, $x_{j2}(p)$. Observe that the retraction point $\rho_j(p)$ lies on $\sigma_j(p)$. If $z_j \in S(p)$, then $\rho_j(p)$ is the intersection point of the ray $\overrightarrow{pz_j}$ with ∂D_p^+, as this is the closest point in C_j from z_j, such that if we place R_j there it will not overlap with R_i at p. Since z_j lies inside $S(p)$, $\rho_j(p) \in \partial S(p)$. If $z_j \notin S(p)$, the retraction point is $\text{argmin}_{q \in \{x_{j1}(p), x_{j2}(p)\}} \|q - z_j\|$, i.e., the closest point to z_j in C_j is an endpoint of $\sigma_j(p)$. Note that our retraction ensures that R_j will be centered back at z_j after robot R_i moves away.

In the remainder of the paper, if $\rho_j(p) \in \{x_{j1}(p), x_{j2}(p)\}$ we say the that the retraction is of *intersection* type, otherwise we say that the retraction is of *sector* type. Since $\rho_j(p) \in C_j$ for all $p \notin C_j$ and none of the $\bar{\gamma}_i$'s enter C_j, the retraction path π_j of R_j lies in $\mathcal{F}$. We conclude this section with the following lemma, which follows from the fact that ρ_j is a continuous function, $\rho_j(p) = z_j$ for all p such that $\|p - z_j\| \geqslant 2$, and $\|z_j - s_i\|, \|z_j - f_i\| \geqslant 2$.

▶ **Lemma 6.** *For any $1 \leqslant i \leqslant n$, π_j is a continuous path from s_i to f_i.*

4 Correctness and Analysis of the Algorithm

We first prove that Π is feasible (Section 4.1), then we bound $\mathscr{c}(\Pi)$ (Section 4.2), and finally analyze the running time in Section A.1. We begin by summarizing a few relevant properties of revolving areas (see Figure 5), which are straightforward to prove.

▶ **Lemma 7.** *Let $x, y \in \mathbf{s} \cup \mathbf{f}$ such that $x \neq y$: (i) $x \in C_x$, that is, each start or final position lies inside the core of A_x; (ii) $\|c_x - c_y\| \geqslant 2$, i.e., int $C_x \cap$ int $C_y = \emptyset$; (iii) for any $p \in C_x$, $\|p - y\| \geqslant 2$, i.e., int $D_p \cap$ int $D_y = \emptyset$; (iv) $\|x - c_y\| \geqslant 3$, i.e., each start/final position lies outside the buffer of any other start/final position.*

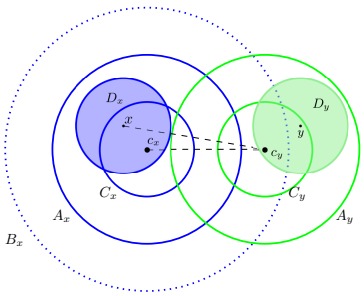


Figure 5 Illustration of Lemma 7. D_x and D_y are two robots, located in their respective revolving areas A_x, A_y. The distance between c_x and c_y is at least 2; y lies outside the buffer of x, i.e., $y \notin B_x$.

4.1 Feasibility

In this section, we show that path ensemble Π is feasible. Recall that stage I) of the algorithm reports that there is no feasible solution if any $s_i \in \mathbf{s}$ and $f_i \in \mathbf{f}$ do not lie in the same connected component. So assume that Stage I computes a feasible path γ_i for each R_i. Stages II and III modify these paths so that they remain in $\mathcal{F}$. Hence, we only need to show that no two robots collide with each other during the motion, i.e., for any $1 \leqslant i \neq j \leqslant n$ and for any $t \in J$, $\operatorname{int} D_{\pi_i(t)} \cap \operatorname{int} D_{\pi_j(t)} = \emptyset$. We fix some $i \in [n]$ and the corresponding active interval $T_i := [i-1, i] \subseteq J$ and prove the feasibility of π during this interval. Note that R_i is the only active robot in T_i and other robots stay in their revolving areas. By the definition of retraction, for any $t \in T_i$, and for any $j \neq i$, $\|\pi_i(t) - \rho_{ij}(\pi_i(t))\| \geqslant 2$, so R_i does not collide with R_j during interval T_i. Thus, we only need to show that for any pair $j, k \neq i$, R_j and R_k do not collide while they are moving along their retraction path. The following lemmas are proved in Appendix A. Since Lemma 9 holds for any interval T_i, we obtain the final statement of feasibility.

▶ **Lemma 8.** *For any $j \neq k$ and $z_j, z_k \in \mathbf{s} \cup \mathbf{f}$, the minimum distance between the line segments $c_j z_j$ and $c_k z_k$ is at least 2, i.e.,* $\min_{\substack{y_j \in c_j z_j, \\ y_k \in c_k z_k}} \|y_j - y_k\| \geqslant 2$.

▶ **Lemma 9.** *For any $j \neq k$, R_j and R_k do not collide during the interval T.*

▶ **Corollary 10.** *The path ensemble Π returned by the algorithm is feasible.*

4.2 Cost of path ensemble

We now analyze the cost of the path ensemble Π the algorithm returns. The algorithm starts by computing Γ, the shortest paths of all robots in $\mathcal{F}$ while ignoring other robots. Clearly, we have $\ell(\Gamma) \leqslant \ell^*(\mathcal{I})$. We show that $\ell(\Pi) = O(\ell(\Gamma))$. In Appendix A.2, we prove $\ell(\bar{\Gamma}) \leqslant 2\ell(\Gamma)$, so we focus on bounding the length of retraction paths of non-active robots, which is one of the main technical contributions of the paper.

Let $\pi_{ji} = \pi_j[i-1, i]$, and let $\Delta_{ij} = \{t \in [i-1, i] : \|\pi_i(t) - z_j\| \leqslant 2\}$, i.e., π_{ji} is the retraction of R_j due to the motion of R_i and $\pi_i[\Delta_{ij}]$ is the part of π_i that causes the retraction motion of R_j. Refer to Figure 3. We show that $\ell(\pi_{ji}) = O(\ell(\pi_i[\Delta_{ij}]))$ (cf Corollary 15) and charge π_{ji} to $\pi_i[\Delta_{ij}]$. We bound $\ell(\pi_{ji})$ by splitting into two scenarios. Roughly speaking, if π_i does not penetrate the buffer B_j too deeply, we use a Lipschitz condition on the retraction map to show $\ell(\pi_{ji}) = O(\ell(\pi_i[\Delta_{ij}]))$. More concretely, for $z \in \mathbf{s} \cup \mathbf{f}$, let W_z be the disk of radius $3/2$ centered at z. We prove a Lipschitz condition when the active robot lies outside W_j (cf Corollary 13). On the other hand, if π_i travels into W_j then the Lipschitz condition may not hold, but we argue that $\ell(\pi_i[\Delta_{ij}]) = \Omega(1)$ and that $\ell(\pi_{ji}) = O(1)$ (cf Lemma 14). Finally, using a packing argument, we show that each "point" of π_i is only charged $O(1)$ times, and thus $\ell(\Pi) = O(\ell(\bar{\Gamma})) = O(\ell(\Gamma))$.

Retraction of $\mathbf{R_j}$ outside $\mathbf{W_j}$. As in Section 4.1, we fix an interval $[i-1, i]$ for some $i \in [n] \setminus \{j\}$. Let $\Delta_j^o = \{t \in [i-1, i] : \|\pi_i(t) - z_j\| \leqslant 2 \text{ and } \pi_i(t) \notin W_j\}$. That is, Δ_j^o is the interval(s) of time in which the path of robot R_i forces the retraction of robot R_j while the center of R_i lies outside W_i. Let Φ_{ij} be the restriction of path π_i of robot R_i during the interval Δ_j^o, i.e. $\Phi_{ij}(t) = \pi_i(t)$ for $t \in \Delta_j^o$. Let $\Psi_{ji} : \Delta_j^o \subset \Delta_{ij} \to C_j$ be the retraction of R_j during Δ_j^o, i.e., $\Psi_{ji}(t) = \rho_{ij}(\pi_i(t))$ for $t \in \Delta_j^o$. We show that $\ell(\Psi_{ji}) = O(\ell(\Phi_{ij}))$ by proving a Lipschitz condition on $\ell(\Psi_{ji})$.

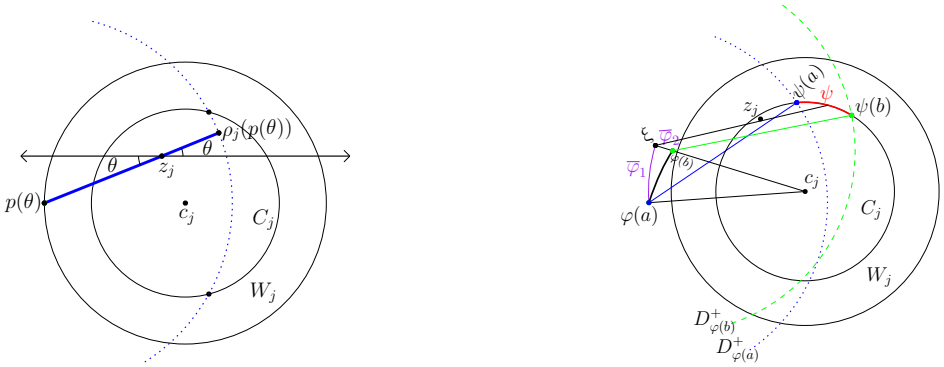


Figure 6 Illustration of Lemma 11 and 12. (a) On the left, the figure shows a sector-type retraction. $p(\theta) = (r(\theta), -\theta)$ and $\rho_j(p(\theta)) = (2 - r(\theta), \theta)$. (b) On the right, the figure shows an intersection-type retraction. The arc ψ on ∂C_j is the retraction path.

We will divide Φ_{ij} into subpaths, referred to as pathlets, so that there is only one type of retraction point associated with the subpath. We call a time instance $t \in \Delta_j^o$ an *event* if t is either an endpoint of a connected component of Δ_j^o (i.e., $\|\pi_i(t) - c_j\| = 3/2$ or $\|\pi_i(t) - z_j\| = 2$) or $z_j \in \partial\mathcal{S}_j(\pi_i(t))$, (i.e., the type of retraction point $\rho_j(\pi_i(t))$ changes at time t). Let $t_0 < t_1 < \cdots < t_k$ be the event points. We divide Φ_{ij} and Ψ_{ji} into *pathlets* at these events, i.e., $\Phi_{ij} = \varphi_1 \circ \varphi_2 \circ \cdots \circ \varphi_g$ and $\Psi_{ji} = \psi_1 \circ \psi_2 \circ \cdots \circ \psi_g$ where $\varphi_k = \pi_i[t_{k-1}, t_k]$ and $\psi_k = \rho_j(\varphi_k) = \pi_j[t_{k-1}, t_k]$. We prove the Lipschitz condition for each pathlet. All points on $\rho_j(\varphi_k)$ have the same type of of retraction by construction of Φ_{ji}. We call φ_k a *sector-type* (*intersection-type*) pathlet if all points have sector (resp. intersection) type retraction.

▶ **Lemma 11.** *For a sector-type pathlet φ_k of Φ_{ji}, $\ell(\rho_j(\varphi_k)) = O(\ell(\varphi_k))$.*

Proof. For each $p \in \varphi_k$, $\rho_j(p)$ is type sector, i.e. $\rho_j(p)$ lies on the ray $\overrightarrow{pz_j}$ at distance 2 from p. In this case, the retraction map $\psi_k = \rho_j(\varphi_k)$ traces a portion of a Conchoid [15].

We parameterize points on $\varphi := \varphi_k$ and $\psi := \psi_k$ using polar coordinates, with z_j as the origin. Let $\varphi(\theta) = (r(\theta), \theta)$ be a point on φ, where θ is the orientation of the point with respect to the x-axis (with z_j as the origin). Then, $\psi(\theta) = \rho_j(\varphi(\theta)) = (2 - r(\theta) - \theta)$. See Figure 6. Note that $\|\varphi'(\theta)\|^2 = r^2(\theta) + (r'(\theta))^2$ and $\|\psi'(\theta)\|^2 = (2 - r(\theta))^2 + (r'(\theta))^2$. Since φ lies outside W_j and $z_j \in C_j$, we have $r(\theta) \in [1/2, 2]$. Therefore, $2 - r(\theta) \leqslant 3r(\theta)$ and $\|\psi'(\theta)\| \leqslant 3\|\varphi'(\theta)\|$. Hence,

$$\ell(\psi) = \int \|\psi'(\theta)\| d\theta \leqslant 3 \int \|\varphi'(\theta)\| d\theta = 3\ell(\varphi).$$ ◀

▶ **Lemma 12.** *For an intersection-type pathlet φ_k of Φ_{ji}, $\ell(\rho_j(\varphi_k)) = O(\ell(\varphi_k))$.*

Proof. Again, we prove the lemma by showing that a Lipschitz condition holds. Let $\varphi := \varphi_k$. We parameterize both φ and $\rho_j(\varphi)$ in polar coordinates, but with c_j as the origin. Let $I = [a, b]$ be the interval over which φ is defined. Let $\varphi(t) = (r(t), \theta(t))$ for $t \in I$. We assume that φ is sufficiently small (otherwise we divide it into smaller pathlets and argue for each pathlet) so that φ both r- and θ-monotone.

Set $\Delta\varphi_r = |r(b) - r(a)|$ and $\Delta\varphi_\theta = |\theta(b) - \theta(a)|$. Since φ lies outside W_j, $r(t) \geqslant 3/2$ for all $t \in [a, b]$. W.l.o.g., assume both $r(t)$ and $\theta(t)$ are monotonically non-decreasing. We obtain:

$$\ell(\varphi) = \int_I \sqrt{r'(t)^2 + r(t)\theta'(t))^2} \geqslant \frac{1}{\sqrt{2}} \int_I \left(r'(t) + \frac{3}{2}\theta'(t) \right) dt \geqslant \frac{1}{\sqrt{2}} (\Delta\varphi_r + \Delta\varphi_\theta).$$

The retraction path $\psi(t)$ varies monotonically on the unit circle ∂C_z. Thus, we parameterize ψ by its direction on ∂C_z, and $\ell(\psi) = |\int_I \psi'(t)\ dt| = |\psi(b) - \psi(a)|$. To bound $\ell(\psi)$, consider the following path from $\varphi(a)$ to $\varphi(b)$, see Figure 6. Let $\overline{\varphi}_1$ be the arc from $\varphi(a)$ to point $\xi = (r(a), \theta(b))$ along the circle of radius $r(a)$ centered at c_z. Let $\overline{\varphi}_2$ be the segment ξ to $\varphi(b)$, this is a radial segment on line ξc_z. Then, $\ell(\psi) \leqslant \ell(\rho_j(\overline{\varphi}_1)) + \ell(\rho_j(\overline{\varphi}_2))$. Since the radius along $\overline{\varphi}_1$ does not change, $\ell(\rho_j(\overline{\varphi}_2)) = |\theta(b) - \theta(a)| = \Delta\varphi_\theta$.

For a point $p = (r, \theta)$, the orientation of $\rho_j(p)$ is $\theta + \cos^{-1}\left(\frac{3-r^2}{2r}\right)$ (by the law of cosines, considering triangle $\triangle\rho_j(p)c_z p$). Since θ does not change along $\overline{\varphi}_2$ and $r(a), r(b) \in [3/2, 3]$, we obtain $\ell(\rho_j(\overline{\varphi}_2)) = O(\Delta\varphi_r)$.

Putting everything together, $\ell(\rho_j(\psi)) = \ell(\psi) = O(\Delta\varphi_r + \Delta\varphi_\theta) = O(\ell(\varphi))$. ◀

Applying Lemmas 11 and 12 to all pathlets of Φ_{ij}, we obtain the following:

▶ **Corollary 13.** *Let $1 \leqslant i \neq j \leqslant n$. Let Φ_{ij} be the portion of π_i during the interval $t \in [i-1, i]$ such that $\|\pi_i(t) - z_j\| \leqslant 2$ and $\|\pi_i(t) - c_j\| \geqslant 3/2$, and let Ψ_{ji} be the retraction of R_j corresponding to Φ_{ij}. Then $\ell(\Psi_{ji}) = O(\ell(\Phi_{ij}))$.*

Retraction path inside $\mathbf{W_j}$. Recall that π_i does not intersect (int C_j), but possibly travels along ∂C_j. For a point $p \in \pi_i$, if $p \in \partial C_j$, then $\rho_j(p)$ is the point on ∂C_j diametrically opposite p. Thus, $\ell(\pi_i \cap C_j) = \ell(\rho_j(\pi_i \cap C_j))$. In the following, we consider only $\pi_i \setminus \partial C_j$.

▶ **Lemma 14.** *For a pathlet φ (i.e., a connected subpath) of path π_i such that $\varphi \subset W_j \setminus C_j$ for some $j \neq i$, $\ell(\rho_j(\varphi)) = O(\ell(\pi_i \cap A_j))$.*

Proof. Since $\varphi \subset W_j$, $\ell(\pi_i \cap A_j) = \Omega(1)$, therefore we only need to argue that $\ell(\rho_j(\varphi))$ is constant. We will bound the length of both types of retraction maps (intersection and sector) separately for φ, and use the sum as an upper bound on the length of the actual retraction map.

Sector retraction. We consider the sector type retraction map. Let z_j be the origin and consider polar coordinates. Let $\rho_j^s(p)$ be the sector type retraction point with respect to p. Since φ is a subpath of a shortest path in $\mathcal{F}$, we can divide $\pi_i \cap W_j$ into at most two pathlets such that each piece is r, θ-monotone. Abusing notation, let φ be one of these pieces with endpoints (r_0, θ_0) and (r_1, θ_1).

We write the retraction point parameterized by θ as $(\rho(\theta), \theta)$. Using the fact that $\rho(\theta) \leqslant 2$ for all θ, the arc length of the retraction map is

$$\ell(\rho_j^s(\varphi)) = \int_{\theta_0}^{\theta_1} \sqrt{\rho(\theta)^2 + \left(\frac{d\rho}{d\theta}\right)^2}\, d\theta \leqslant \int_{\theta_0}^{\theta_1} \rho(\theta) d\theta + \int_{\theta_0}^{\theta_1} \frac{d\rho(\theta)}{d\theta} d\theta$$
$$\leqslant \rho(\theta_1 - \theta_0) + (\rho(\theta_1) - \rho(\theta_0)) \leqslant 2(\theta_1 - \theta_0) + 2.$$

Therefore, $\ell(\rho_j^s(\varphi)) = O(1)$, for each φ.

Intersection retraction. We consider the retraction map defined by an intersection point of ∂D_p^+ and ∂C_j. We now let c_j be the origin and consider polar coordinates. Let $\rho_j^i(p)$ be the intersection type retraction point closest to z_j with respect to p. Again, we divide $\pi_i \cap W_j$ into at most two pathlets such that each of them is r, θ-monotone (one pathlet is the portion of π_i coming closer to the core C_j, and the other moves away from C_j). Let φ be one of the pathlets with endpoints (r_0, θ_0) and (r_1, θ_1). The retraction point lies on the unit circle ∂C_j, and as θ changes monotonically from θ_0 to θ_1, the retraction point $\rho_j^i(\theta)$ moves monotonically on ∂C_j. Therefore, $\ell(\rho_j^i(\varphi)) = O(1)$.

Finally, $\ell(\rho_j(\varphi)) \leqslant \ell(\rho_j^s(\varphi)) + \ell(\rho_j^i(\varphi)) = O(1)$, as claimed. ◀

Applying Lemma 14 to each of (at most two) connected components of $(\pi_i \cap W_j) \setminus C_j$ and combining with Corollary 13, we obtain the following:

▶ **Corollary 15.** *For $1 \leqslant i \neq j \leqslant n$, let Δ_{ij} be defined as $\Delta_{ij} = \{t \in [i-1, i] : \|\pi_i - z_j\| \leqslant 2\}$ and let $\pi_{ji} = \pi_j[i-1, i]$. Then $\ell(\pi_{ji}) = O(\ell(\pi_i[\Delta_{ij}]))$.*

Cost of Path Ensemble. We are now ready to bound the cost of the path ensemble Π returned by the algorithm.

▶ **Lemma 16.** *For an instance $\mathcal{I}$ of optimal MRMP with revolving areas, let $\Pi(\mathcal{I})$ be the path ensemble returned by the algorithm. Then $\mathfrak{c}(\Pi(\mathcal{I})) = O(1) \cdot \mathfrak{c}^*(\mathcal{I})$.*

We analyze the running time of the algorithm and show that the total running time is $O(n(m+n)\log m)$ in Appendix A.1.

▶ **Theorem 17.** *Let $\mathcal{I} = (\mathcal{W}, \mathbf{s}, \mathbf{f}, \mathcal{A})$ be an instance of optimal MRMP with revolving areas, and let m be the complexity of $\mathcal{W}$. If a feasible motion plan of $\mathcal{I}$ exists then a path ensemble Π of cost $O(\mathfrak{c}^*(\mathcal{I}))$ can be computed in $O(n(m+n)\log m)$ time.*

We conclude this section by noting that since the ordering σ (of active robots) is arbitrary, the algorithm can be extended to an online setting where R_i and (s_i, f_i) are given in an online manner (as long as each s_i, f_i given satisfies the revolving area property). Our algorithm is $O(1)$-competitive for this setting, i.e., the cost is $O(1)$ times the optimal cost of the offline problem.

5 Computing a Good Ordering

In the previous section, we proved that the total cost of the path ensemble Π is $O(1) \cdot \mathfrak{c}^*(\mathcal{I})$ irrespective of the order in which the robots moved. However, the order in which robots move has a significant impact on how the paths are edited in Stages (II) and (III). The increase in cost because of editing may vary between 0 and $O(nm)$ depending on the ordering (see [17] for a related argument). For a path ensemble Π computed by our algorithm, let $\Delta\mathfrak{c}(\Pi) = \mathfrak{c}(\Pi) - \mathfrak{c}(\Gamma)$, which we refer to as the *marginal cost* of Π, where Γ is the path ensemble computed in Stage (I). For a permutation σ of $[n]$, let Π_σ be the path ensemble computed by the algorithm if robots were moved in the order determined by σ. Set $\Delta\mathfrak{c}(\sigma) \coloneqq \Delta\mathfrak{c}(\Pi_\sigma)$. Finally, set $\Delta\mathfrak{c}^*(\mathcal{I}) = \min_\sigma \Delta\mathfrak{c}(\sigma)$, where the minimum is taken over all permutations of $[n]$.

Adapting the construction in [17], we can show that the problem of determining whether $\Delta\mathfrak{c}^*(\mathcal{I}) \leqslant L$, for some $L \geqslant 0$, is NP-hard. We present an approximation algorithm for computing a good ordering σ such that $\Delta\mathfrak{c}(\sigma) = O(\log n \log\log n)\Delta\mathfrak{c}^*(\mathcal{I})$.

Our main observation is that $\Delta\mathfrak{c}(\sigma)$, the marginal cost of an ordering σ, is decomposable, in the sense made precise below. For a pair $i \neq j$, we define $w_{ij} \geqslant 0$ to be the contribution of the pair R_i, R_j to the marginal cost of an ordering σ, assuming $i \prec_\sigma j$, i.e., how much the shortest path γ_i has to be modified because of γ_j and vice-versa assuming R_i is active before R_j. Note that if $i \prec_\sigma j$ then R_i (resp. R_j) is at f_i (resp s_j) when R_j (resp R_i) is active. There are two components of w^σ_{ij}: (i) R_i (resp. R_j) enters the core C_{s_j} (resp. C_{f_i}) in γ_i (resp. γ_j), (ii) retraction motion of R_j (resp. R_i) when R_i (resp. R_j) enters the buffer disc B_{s_j} (resp. B_{f_i}).

Let ϕ_{ij} (resp. ϕ_{ji}) be the arc of ΔC_{s_j} (resp. ΔC_{f_i}) with which $\gamma_i \cap C_{s_j}$ (resp. $\gamma_j \cap C_{f_i}$) is replaced with. Then $\alpha_{ij} = \ell(\phi_{ij} + \ell(\phi_{ji}) - \ell(\gamma_i \cap C_{s_j}) - \ell(\gamma_j \cap C_{f_i})$ is the contribution of (i) to w_{ij}. For (ii), we define $\rho^<_{ij}$ (resp. $\rho^>_{ij}$) be the retraction map of R_j because of R_i when R_i is active before (resp. after) R_j. Then $w_{ij} = \alpha_{ij} + \ell(\rho^<_{ij}(\bar{\gamma}_i)) + \ell(\rho^>_{ij}(\bar{\gamma}_j))$. From the previous two components, we have $\Delta\mathfrak{c}(\sigma) = \sum_{i,j:i\prec_\sigma j} w_{ij}$.

We now reduce the problem of computing an optimal ordering to instance of *weighted feedback-arc-set* (FAS) problem. Given a directed graph with weights on the edges, $G = (V, E), w : E \to \mathbb{R}_{\geqslant 0}$, a feedback arc set F is a subset of edges of G whose removal makes G a directed acyclic graph. The weight of $F, w(F)$, is $\sum_{e \in F} w(e)$. The FAS problem asks to compute an FAS of the smallest weight. It is known to be NP-complete.

Given an MRMP-RA instance $\mathcal{I} = (\mathcal{W}, \mathbf{s}, \mathbf{f}, \mathcal{A})$, we first compute Γ as in stage (I) of the algorithm. Next, for each pair $i, j \in [n]$, we construct a directed graph as follows. $G = (V, E)$ is a complete directed graph with $V = [n]$, one representing each robot, $E = \{i \to j : 1 \leqslant i \neq j \leqslant n\}$, $w(i \to j) = w_{ij}$. It can be shown that each feedback arc set F of G induces an ordering σ_F on $[n]$, and vice versa. Furthermore, $w(F) = \Delta\mathfrak{c}(\sigma_F)$. Even et al. [8] have described a polynomial-time $O(\log n \log\log n)$-approximation algorithm for the FAS problem. By applying their algorithm to G, we obtain the following.

▶ **Theorem 18.** *Let $\mathcal{I} = (\mathcal{W}, \mathbf{s}, \mathbf{f}, \mathcal{A})$ be an instance of optimal MMP with revolving areas, and let m be the complexity of $\mathcal{W}$. Let the optimal order of execution of paths be σ^*. An ordering σ with $\Delta\mathfrak{c}(\sigma) = O(\log n \log\log n)\Delta\mathfrak{c}(\sigma^*)$ can be computed in polynomial time in n and m.*

References

1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *IEEE Trans Autom. Sci. Eng.*, 12(4):1309–1317, 2015.

2 Bahareh Banyassady, Mark de Berg, Karl Bringmann, Kevin Buchin, Henning Fernau, Dan Halperin, Irina Kostitsyna, Yoshio Okamoto, and Stijn Slot. Unlabeled Multi-Robot Motion Planning with Tighter Separation Bounds. In *38th International Symposium on Computational Geometry (SoCG)*, 2022.

3 Eric Berberich, Dan Halperin, Michael Kerber, and Roza Pogalnikova. Deconstructing approximate offsets. *Discret. Comput. Geom.*, 48(4):964–989, 2012.

4 Josh Brunner, Lily Chung, Erik D. Demaine, Dylan H. Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff. 1 X 1 rush hour with fixed blocks is pspace-complete. In *10th International Conference on Fun with Algorithms*, volume 157, pages 7:1–7:14, 2021.

5 Danny Z Chen and Haitao Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms*, 11(4):1–46, 2015.

6 Dror Dayan, Kiril Solovey, Marco Pavone, and Dan Halperin. Near-optimal multi-robot motion planning with finite sampling. In *IEEE International Conference on Robotics and Automation*, pages 9190–9196, 2021.

7 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019.

8 Guy Even, J Seffi Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

9 Tzvika Geft and Dan Halperin. On the complexity of a family of decoupled multi-robot motion planning problems, 2021. `arXiv:2104.07011`.

10 Tzvika Geft and Dan Halperin. Refined hardness of distance-optimal multi-agent path finding. In *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 481–488, 2022.

11 John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

12 Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

13 C O'Ddnlaing and CK Yap. A retraction method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1985.

14 Oren Salzman. Sampling-based robot motion planning. *Commun. ACM*, 62(10):54–63, 2019.

15 Jacob T Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983.

16 Rahul Shome, Kiril Solovey, Andrew Dobson, Dan Halperin, and Kostas E. Bekris. dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning. *Auton. Robots*, 44(3-4):443–467, 2020.

17 Israela Solomon and Dan Halperin. Motion planning for multiple unit-ball robots in $\mathbb{R}^d$. In *Workshop on the Algorithmic Foundations of Robotics, WAFR*, pages 799–816, 2018.

18 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *Int. J. Robotics Res.*, 35(14):1750–1759, 2016.

19 Kiril Solovey, Lucas Janson, Edward Schmerling, Emilio Frazzoli, and Marco Pavone. Revisiting the asymptotic optimality of RRT. In *2020 IEEE International Conference on Robotics and Automation*, pages 2189–2195. IEEE, 2020.

20 Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion planning for unlabeled discs with optimality guarantees. In *Robotics: Science and Systems*, 2015.

21 Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proc. 12th International Symposium on Combinatorial Search*, pages 151–159, 2019.

22 Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Auton. Robots*, 37(4):401–415, 2014.

23 Vijay V Vazirani. *Approximation Algorithms*. Springer, 2001.

24 Chee-Keng Yap. An O($n \log n$) algorithm for the voronoi diagram of a set of simple curve segments. *Discret. Comput. Geom.*, 2:365–393, 1987.

A Appendix

A.1 Running-time Analysis

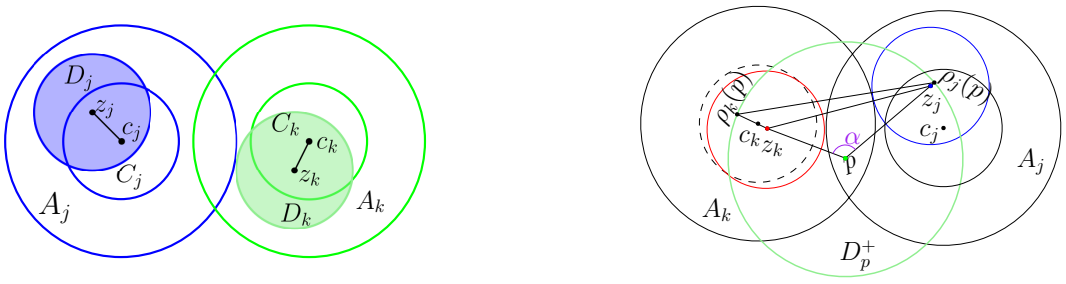
The algorithm has three stages. In the first stage, we compute the free space $\mathcal{F}$ with respect to one robot, which takes $O(m \log m)$ time, by computing the Voronoi of $\mathcal{W}$, see the algorithm of [24], and see [3] for details. In the same stage, we compute a set of shortest paths Γ for n discs, using the algorithm of [5], taking $O(mn \log m)$ time in total over all robots. Each path $\gamma_i \in \Gamma$ has complexity $O(m)$. In stage two of the algorithm, γ_i is modified to avoid the core of any occupied revolving area, increasing the complexity of each curve to $O(m+n)$. In stage three of the algorithm, the deformed paths $\bar{\gamma}_i$ are again edited to include retraction maps in which non-active robots may move within their revolving area. It suffices to bound the number of breakpoints in the final path π_j that correspond to retracted maps. Let ξ be such a breakpoint on π_j, which is $\rho_{ij}(\bar{\gamma}_i(t))$ for some $t \in [i-1, i]$. There are two cases: (i) the preimage of ξ on $\bar{\gamma}_i$ is a breakpoint of $\bar{\gamma}_i$, or (ii) $\|\xi - z_j\| = 2$ (i.e., $\bar{\gamma}_i$ forces R_j to move within the revolving area). We charge both of these breakpoints to $\bar{\gamma}_i$. Since the preimage of ξ lies in the buffer disk of R_j, using a packing argument similar to the proof of Lemma 16 below, we can show that $O(m+n)$ breakpoints are charged to $\bar{\gamma}_i$. Therefore, the total complexity of all paths in Π is only $O(n(m+n))$.

A.2 Missing Proofs

▶ **Lemma 1.** *$\mathcal{I}$ has a weakly monotone path ensemble with a cost of d if and only if $\mathcal{I}$ has a monotone path ensemble.*

Proof. Let A be a revolving area in $\mathcal{W}$. We first note that without any loss of generality, in any path ensemble of $\mathcal{I}$ a robot may either be contained in A at some point or never intersect A at all.

Let Π be a feasible path ensemble with $\mathfrak{c}(\Pi) = d$. We fix a robot R_i and examine the motion that occurs during its active interval τ_i. We claim that any motion of a robot $R_j, j \neq i$ during τ_i is *redundant*, i.e., if R_j does not move during τ_i then R_i can still perform the same motion. This suffices in order to conclude that Π can be made monotone. Observe that during the execution of Π no revolving area A can become congested, as otherwise the two robots that are simultaneously in A will have to take a path that is longer than the shortest path that ignores other robots. Therefore, whenever R_i is inside a revolving area A, it is the only robot in A, and any motion by other robots is redundant. Whenever R_i is not contained in any revolving area, all other robots must be contained in revolving areas, by definition. Hence, any motion by other robots at such point in time is also redundant. So overall, R_i may travel along its whole path without other robots moving. For the other direction, in a monotone path ensemble it also holds that no revolving area may become congested (as otherwise robots move simultaneously). Therefore, any revolving area that some robot R_i intersects during its motion must not contain other robots. For any gadget g that R_i needs to traverse, this allows R_i to take some shortest path through g. Therefore, R_i is able to take the shortest path that ignores other robots overall. Hence, a path ensemble with a cost of d exists. ◀



(a) **(b)**

Figure 7 (a) Illustration of Lemma 8. (b) Case 2 of Lemma 9, The angle $\alpha \geqslant \pi/3$.

▶ **Lemma 8.** *For any $j \neq k$ and $z_j, z_k \in \mathbf{s} \cup \mathbf{f}$, the minimum distance between the line segments $c_j z_j$ and $c_k z_k$ is at least 2, i.e., $\min_{\substack{y_j \in c_j z_j, \\ y_k \in c_k z_k}} \|y_j - y_k\| \geqslant 2$.*

Proof. Let y_j, y_k be the closest pair of points on the segments $c_j z_j$ and $c_k z_k$. Note that $z_j c_j$ and $z_k c_k$ are disjoint since $z_j c_j \in C_j$ and $z_k c_k \in C_k$ and these cores do not intersect (cf Lemma 7). This implies that either y_j or y_k is an endpoint of the respective segment.

Assume without loss of generality that y_j is an endpoint of $z_j c_j$. By Lemma 7, $\|c_j - z_k\|, \|c_k - z_j\| \geqslant 3$. Let y'_k be the endpoint of $c_z z_k$ at distance 3 from y_j ($y'_k = z_k$ if $y_j = c_j$ and $y'_k = c_k$ otherwise). Since $z_k \in C_k$, $\|y_k - y'_k\| \leqslant 1$, Then $\|y_j - y_k\| \geqslant \|y_j - y'_k\| - \|y'_k - y_k\| \geqslant 3 - 1 = 2$. ◀

▶ **Lemma 9.** *For any $j \neq k$, R_j and R_k do not collide during the interval T.*

Proof. In view of the above discussion, we assume $j, k \neq i$. The claim is equivalent to showing that $\|\rho_j(\pi_i(t)) - \rho_k(\pi_i(t))\| \geqslant 2$ for every $t \in T$. Let $p = \pi_i(t)$. There are two cases:

Case 1: $\rho_j(p) = z_j$ or $\rho_k(p) = z_k$. Without loss of generality, assume that $\rho_j(p) = z_j$. By construction, $\rho_k(p) \in C_k$, therefore by Lemma 7(iii), $\|\rho_j(p) - \rho_k(p)\| = \|z_j - \rho_k(p)\| \geqslant 2$.

Case 2: $\rho_j(p) \neq z_j$ and $\rho_k(p) \neq z_k$. Recall D_p^+ is the disc of radius 2 centered at p, and $x_{j,1}, x_{j,2}$ are the intersection points of the core of j and δD_p^+. In this case, $z_j, z_k \in D_p^+$.

We consider the triangle formed by the retraction points $\rho_k(p)$, $\rho_j(p)$ and p. We show that $\angle\rho_k(p)p\rho_j(p) \geqslant \pi/3$. We will first define a point $f_j(p)$ based on the current retraction type of j.

If the retraction of j is type sector, then z_j lies within the sector $S(p)$, let $f_j(p) = z_j$. Otherwise, the retraction is of type intersection and without loss of generality we assume $\rho_j(p)$ is $x_{j,1}(p)$. In this case, consider the segments $px_{j,1}(p)$ and z_jc_j. These two segments must intersect, as $c_j \in \mathcal{S}(p)$ and $z_j \notin \mathcal{S}(p)$. We let $f_j(p)$ be the intersection point of segments. Note that in either case, $f_j(p)$ lies on segment $p\rho_j(p)$. We analogously define $f_k(p)$. See Figure 7 for an example where $f_j(p) = z_j$ and $f_j(p) = z_k$.

By definition, $f_j(p) \in z_jc_j$ and $f_k(p) \in z_kc_k$ and Lemma 8 implies that $\|f_j(p) - f_k(p)\| \geqslant 2$. Additionally, $f_j(p) \in D_p^+$, so $\|p - f_j(p)\| \leqslant 2$ (similarly $\|p - f_k(p)\| \leqslant 2$). Let α be the angle $\angle f_k(p)pf_j(p)$. Since $\|p - f_j(p)\|, \|p - f_k(p)\| \leqslant 2$, and $\|f_j(p) - f_k(p)\| \geqslant 2$, $\alpha \geqslant \pi/3$.

Now consider the triangle formed by the retraction points and p. By construction, $\angle f_k(p)pf_j(p) = \angle\rho_k(p)p\rho_j(p)$. The distance between p and each retraction point is 2: $|p\rho_j(p)| = |p\rho_k(p)| = 2$. This implies the other two angles in the triangle are equal ($\angle p\rho_k(p)\rho_j(p) = \angle p\rho_j(p)\rho_k(p)$). Since $\angle\rho_k(p)p\rho_j(p) = \alpha \geqslant \pi/3$, $\rho_j(p)\rho_k(p)$ is the longest edge of the triangle $\triangle p\rho_j(p)\rho_k(p)$. The other two sides have length 2, so $\|\rho_j(p) - \rho_k(p)\| \geqslant 2$, as desired. ◀

Cost of $\overline{\Gamma}$. Stage (II) of the algorithm deforms Γ to $\overline{\Gamma}$. Path $\gamma_i \neq \overline{\gamma}_i$ only if $\gamma_i \cap \text{ int } C_j \neq \emptyset$ for some $j \neq i$, otherwise $\ell(\gamma_i) = \ell(\overline{\gamma}_i)$. Suppose $\gamma_i \cap C_j \neq \emptyset$ for some $j \neq i$. Then in $\overline{\gamma}_i$, $\gamma_i \cap C_j$ is replaced with the shorter arc σ of ∂C_z, determined by the first and last endpoints, say p and q, of $\gamma_i \cap \partial C_j$. Therefore, $\ell(\sigma) \leqslant 2\sin^{-1}\left(\frac{\|p-q\|}{2}\right) \leqslant 2\ell(C_j \cap \gamma_i)$. Hence, $\ell(\bar{\gamma}_i) \leqslant 2\ell(\gamma_i)$ and we obtain: $\mathrm{c}(\bar{\Gamma}) \leqslant 2\mathrm{c}(\Gamma)$.

▶ **Lemma 16.** *For an instance $\mathcal{I}$ of optimal MRMP with revolving areas, let $\Pi(\mathcal{I})$ be the path ensemble returned by the algorithm. Then $\mathrm{c}(\Pi(\mathcal{I})) = O(1) \cdot \mathrm{c}^*(\mathcal{I})$.*

Proof. Set $\Pi = \Pi(\mathcal{I})$. We already argued that $\mathrm{c}(\overline{\Gamma}) = O(\mathrm{c}^*(\mathcal{I}))$, where $\overline{\Gamma}$ is the path ensemble computed in stage II of the algorithm. We thus need to prove $\mathrm{c}(\Pi) = O(\mathrm{c}(\overline{\Gamma}))$. For a pair $1 \leqslant i, j \leqslant n$, let $\pi_{ij} = \pi_i[j-1, j]$. By construction, $\ell(\pi_{ii}) = \ell(\overline{\gamma}_i)$. For a fixed i,

$$\ell(\pi_i) = \sum_{j=1}^{n} \ell(\pi_{ij}) = \ell(\overline{\gamma}_i) + \sum_{j \neq i} \ell(\pi_{ij}) = \ell(\overline{\gamma}_i) + \sum_{j \neq i} O(\ell(\pi_j[\Delta_{ji}])).$$

Where the last equality follows from Corollary 15. Hence,

$$\mathrm{c}(\Pi) = \sum_{j=1}^{n} \ell(\pi_i) = \sum_{i=1}^{n} \ell(\overline{\gamma}_i) + \sum_{i=1}^{n} \sum_{j \neq i} O(\ell(\pi_j[\Delta_{ji}])) = \mathrm{c}(\overline{\Gamma}) + \sum_{i=1}^{n} \sum_{j \neq i} O(\ell(\pi_j[\Delta_{ji}])).$$

By definition of Δ_{ji}, $\Delta_{ji} \subseteq [j-1, j]$ and $\pi_j[\Delta_{ji}] \subseteq B_{z_i}$. Fix a point $x \in \mathbb{R}^2$. Consider a disk D of radius 4 centered at x. If $x \in B_z$ for some $z \in \mathbf{s} \cup \mathbf{f}$, then $C_z \subseteq D$. Since cores are pairwise-disjoint (cf Lemma 7(i)), D can contain at most 16 core disks and any $t \in [j-1, j]$ lies in $O(1)$ Δ_{ji}'s. Therefore,

$$\sum_{i \neq j} O(\ell(\pi_j[\Delta_{ji}])) = O(\ell(\pi_j[j-1, j])) = O(\ell(\overline{\gamma}_j)).$$

Plugging this back in we obtain: $\mathscr{c}(\Pi) = \mathscr{c}(\overline{\Gamma}) + \sum_{j=1}^{n} O(\mathscr{c}(\overline{\gamma}_j)) = O(\mathscr{c}(\overline{\Gamma}))$. ◀

Nested Active-Time Scheduling

Nairen Cao
Georgetown University, Washington D.C., USA

Jeremy T. Fineman
Georgetown University, Washington D.C., USA

Shi Li
University at Buffalo, NY, USA

Julián Mestre
The University of Sydney, Australia

Katina Russell
Georgetown University, Washington D.C., USA

Seeun William Umboh
The University of Sydney, Australia

Abstract

The **active-time scheduling problem** considers the problem of scheduling preemptible jobs with windows (release times and deadlines) on a parallel machine that can schedule up to g jobs during each timestep. The goal in the active-time problem is to minimize the number of active steps, i.e., timesteps in which at least one job is scheduled. In this way, the active time models parallel scheduling when there is a fixed cost for turning the machine on at each discrete step.

This paper presents a 9/5-approximation algorithm for a special case of the active-time scheduling problem in which job windows are laminar (nested). This result improves on the previous best 2-approximation for the general case.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling algorithms, Active time, Approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.36

Related Version *Previous Version*: https://doi.org/10.1145/3490148.3538554

1 Introduction

The active-time scheduling is the problem of scheduling jobs with windows on a parallel machine so as to minimize the number of timesteps during which machine is on, or active.

In the active-time problem [2], we are given as input a set J of n jobs, where each job $j \in J$ has an associated processing time p_j, release time r_j, and deadline $d_j \geq r_j + p_j$, all integers. The jobs are scheduled on a parallel machine that can execute up to g jobs during each step, where g is a positive integer specified as part of the input. The input thus comprises the jobs with their processing times p_j, release times r_j, and deadlines d_j, as well as the machine parameter g, all of which are integers. Time is organized into discrete (integer) steps or slots, and preemption is allowed but only at slot boundaries. We call the time interval $[r_j, d_j)$ the job j's **window**. Each job j must be fully scheduled within its window, i.e., each job must be assigned to p_j timesteps, where each timestep t to which the job is assigned satisfies $r_j \leq t < d_j$. Moreover, at most g jobs can be scheduled at any step.

We say that a timestep t is **active** if the schedule assigns at least one job to step t. The goal in the **active-time scheduling problem** is to find a schedule with minimum number of active steps that schedule all jobs within their windows.

We assume throughout that the instance is feasible. Testing feasibility (and producing a schedule) is an easy exercise applying max flow. In fact, this flow-based feasibility test generalizes to any given subset of active timesteps (see, e.g., Appendix A.1 of [10]). The active-time scheduling problem thus boils down to figuring out which slots should be activated.

Problem History

Chang, Gabow, and Khuller [2] introduce the active-time problem and show that the problem can be solved optimally in polynomial time when the processing times are all one. They also investigate various generalizations of the problem. For arbitrary integer processing times, Chang, Khuller, and Mukherjee [3] give two approximation algorithms. First, they give a rather complex rounding of the natural linear program (LP) that yields a 2-approximation. They also show that the integrality gap of the natural LP is 2, so the rounding is tight. Second, they show that, any **minimal feasible solution** yields a 3-approximation. A minimal feasible solution is a set of active slots such that (i) scheduling the jobs on those slots is feasible, and (ii) deactivating any slot would render the schedule infeasible. Consequently, a simple greedy algorithm (choose an arbitrary active slot and deactivate it if the resulting set of slots is still feasible) is a 3-approximation for the problem. Kumar and Khuller [10] give a greedy 2-approximation algorithm following the same general strategy of deactivating slots until reaching a minimal feasible solution, but they choose slots more carefully. They also exhibit inputs on which their algorithm achieves no better than a $2 - 1/g$ approximation, so the analysis is effectively tight.

A key challenge for improving the approximation ratio for the general active-time problem is that the natural linear program has an integrality gap of at least $2 - O(1/g)$, which converges to 2 as $g \to \infty$. There is also no clear avenue to improve the 2-approximation obtained by Kumar and Khuller's [10] greedy approach. Călinescu and Wang [6] suggest a stronger LP formulation that they conjecture has lower integrality gap, but they only show that the gap is at least 5/3 – whether it can lead to better approximations for general instances remains unknown. Recently, Davies et al.[7] study the active time problem in the batch scheduling model.

Nested active-time scheduling

To push past the barriers for the general version of the problem, this paper instead considers a special case of the active-time problem in which the job windows are laminar (nested). That is, for each pair of jobs i, j, either the intervals $[r_i, d_i)$ and $[r_j, d_j)$ are disjoint (meaning either $d_i \leq r_j$ or $d_j \leq r_i$)), or one of the intervals is fully contained in the other (i.e., either $r_i \leq r_j < d_j \leq d_i$ or $r_j \leq r_i < d_i \leq d_j$).

Our main result is a 9/5-approximation algorithm for the active-time problem with laminar job windows. Since the simple example exhibiting the integrality gap [3] of 2 for the natural LP is a nested instance, a different LP formulation is needed. Our algorithm starts by solving a stronger linear program (LP) for the problem to produce a fractional solution, then performing a new rounding process over the tree of job windows. The algorithm itself is not overly complex, but the analysis is not at all straightforward.

Restricting our attention to nested windows gives us two advantages. First, assuming laminar windows allows us to augment the LP to obtain a smaller integrality gap than for the general case. Notably, our LP includes an additional "ceiling constraint," which represents a stronger lower bound on the number of slots required in each window to the volume of jobs therein. It is not clear how to take advantage of this same constraint in the general version

of the problem. As exhibited by our algorithm the integrality gap of our LP on the nested version of the problem is at most 9/5, which provides a separation between the nested and general versions of the problem. Secondly, the rounding process itself is inherently tied to the fact that nested windows form a tree. Even ignoring the issue of the larger integrality gap for the general case, it is not clear how to perform a similar rounding for general windows.

In Section 5, we compare our strengthened LP formulation to that proposed by Călinescu and Wang's [6] and show that both formulations exhibit an integrality gap of at least 3/2 for nested instances. In Section 6, we show that the nested active time problem is NP complete.

Related work

The objective of the active-time problem is motivated by an application to energy minimization. In this context, the machine can be turned off when no jobs are being executed and it takes the same amount of energy to run regardless of how many jobs are running – but it has a fixed capacity g of how many jobs it can process per active time slot. Energy-aware scheduling is an active area of research [1, 8] that is motivated by the pressing need of modern data centers whose large energy footprint accounts for most of their running costs [13].

There are many variations and generalizations of the basic setup studied in this paper. Below we consider two of its most closely related variants. The reader is referred to the excellent survey by Chau and Li [4] for more related results such as online algorithms for active-time scheduling.

A natural generalization of the basic setup studied in this paper is to have, instead of a single interval, a collection of intervals where each job can be scheduled. Chang *et al.* [2] show that this generalization is NP-hard when $g \geq 3$ even when jobs are unit-length, but that it can be solved in polynomial time when $g = 2$. Furthermore, the problem admits an H_g-approximation for general g via Wolsey's submodular cover framework [14].

Another related model is the busy-time problem where jobs cannot be preempted and we have parallel machines. This problem is much harder as even testing feasibility for a fixed number of machines is NP-hard. Indeed, the best approximation algorithm for minimizing the number of machines needed when $g = 1$ is the $O\left(\sqrt{\frac{\log n}{\log\log n}}\right)$-approximation of Chuzhoy *et al.* [5]. Koehler and Khuller [9] show that it is possible minimize the number of machines use and simultaneously achieve a $O(1)$-approximation on the busy-time objective for instances with uniform processing time; for general instances with arbitrary processing times they can approximate the number of machines by a $O\left(\log \frac{p_{\max}}{p_{\min}}\right)$ factor while keeping the constant approximation bound for the busy-time objective. Liu and Tang [11] studies a generalized busy-time problem on heterogeneous machines and they show an $O(1)$-approximation algorithm in the offline setting and an O(max/min job length ratio)-competitive algorithm in the online setting.

2 Preliminaries

For an integer p, We use $[p]$ to represent integers from $\{1, 2, ..., p\}$. Given an instance of the nested active time problem, we define its tree T as follows. Each tree node i is associated with an interval $K(i)$ such that $K(i) = [r_j, d_j)$ for some $j \in J$. If there are several jobs with the same interval, we only create a single tree node. A tree node i' is a child of i if $K(i') \subsetneq K(i)$ and no other node interval is strictly between $K(i)$ and $K(i')$, i.e, there is no node i'' such that $K(i') \subsetneq K(i'') \subsetneq K(i)$. The descendants and ancestors of a node i are denoted $Des(i)$ and $Anc(i)$, respectively. Note that both $Des(i)$ and $Anc(i)$ include i itself.

To exclude x, we use $Des^+(i) = Des(i)\backslash\{i\}$ and $Anc^+(i) = Anc(i)\backslash\{i\}$. We define $\mathsf{par}(i)$ to be the parent node of i. W.l.o.g we can assume T is indeed a tree (instead of a forest) since otherwise the instance can be broken into several independent ones.

We assume that the tree contains m nodes and each node is associated with an unique id in $[m]$. Now, each job j's interval is associated with a node in the tree. For a job j, define $k(j)$ to be the tree node i with $K(k(j)) = [r_j, d_j)$; we say j belongs to the node i if $i = k(j)$. For jobs j_1 and j_2, if $r_{j_1} = r_{j_2}$ and $d_{j_1} = d_{j_2}$, then $k(j_1) = k(j_2)$. Given a node i and a job subset $J' \subseteq J$, $J'(i) = \{j \in J' \mid k(j) = i\}$ is the set of jobs in J' belonging to i. Note that at least one job belongs to each node. Define the length of a node i, which is denoted as $L(i)$, as the $|K(i)| - \sum_{i':\mathsf{par}(i')=i} |K(i')|$, i.e, the number of time slots in the interval $K(i)$, but not in $K(i')$ for any child node i' of i.

For simplicity, we use the following shorthand. Given a function or vector f and a set S, if f outputs reals, then $f(S) = \sum_{e\in S} f(e)$ or $f(S) = \sum_{e\in S} f_e$. If f outputs subsets, then $f(S) = \bigcup_{e\in S} f(e)$ or $f(S) = \bigcup_{e\in S} f_e$.

We say that a node i is **rigid** if a feasible solution must open the entire interval $K(i)$. For our rounding algorithm, it will be convenient if the tree is **canonical**.

▶ **Definition 1** (Canonical trees). *A tree is* canonical *if it is a binary tree and each leaf node is rigid.*

First, we transform an arbitrary tree to a binary tree. If a parent node i contains several children nodes $i_1, i_2, ..., i_t$, we will create several virtual nodes so that each node contains at most 2 children. Each virtual node's interval is the union of its children's intervals. There are no jobs associated with the virtual nodes and the length of a virtual node i' satisfying $L(i) = 0$. Notice that this transformation adds at most t virtual nodes for a node with t children. In total, this transformation only adds m virtual nodes to a tree that had m nodes originally. In the resulting tree, only internal nodes can be virtual so each leaf node must have at least one job associated with it.

We perform one final transformation to make each leaf node rigid. For a leaf node i, let $j \in J(i)$ be a job in i with the longest processing time. If $p_j = L(i)$, then we leave i and the jobs therein unchanged. Otherwise, we can assume that j is scheduled in the first p_j steps of i because j is the longest job in the leaf node and all jobs in the leaf node could choose the leaf's interval to fit in. We transform the instance by creating a virtual child node i' of the leaf i with interval corresponding to the first p_j steps of $K(i)$, and we reduce j's window to match i''s. Notice this transformation does not change our solution for the original tree.

3 Algorithm

3.1 Linear Program

The linear program is LP (1), which is given in Figure 1(a). In the LP, $x(i)$ denotes the number of time slots opened in node i, excluding times slots in i's children. $y(i, j)$ denotes the amount of job j that is scheduled in node i. In the LP, OPT_i denotes the smallest number of slots to schedule the jobs in $J(Des(i))$.

The objective is to minimize $\sum_{i\in[m]} x(i)$. (2) ensures that every job j is scheduled in at least p_j time slots. (3) ensures that the total number of jobs scheduled in $x(i)$ is at most $g \cdot x(i)$, for each node $i \in [m]$. (4) requires that the number of open time slots in a node $x(i)$ is at most the interval length $L(i)$ of node i. (5) says that we could give at most $x(i)$ time slots for a job j. (6) restricts that for each job $j \in J$, j can only be put into nodes in $Des(K(j))$. (7) and (8) are the key constraints that makes the LP stronger. They are clearly valid; moreover, checking if $\mathrm{OPT}_i \geq 2$ ($\mathrm{OPT}_i \geq 3$) can be done easily.

$$\textbf{min} \quad \sum_{i\in[m]} x(i) \quad \textbf{s.t.} \tag{1}$$

$$\sum_{i\in Des(K(j))} y(i,j) \geq p_j, \quad \forall j \tag{2}$$

$$\sum_{j\in J(Anc(i))} y(i,j) \leq g\cdot x(i), \quad \forall i \tag{3}$$

$$x(i) \leq L(i), \quad \forall i \tag{4}$$

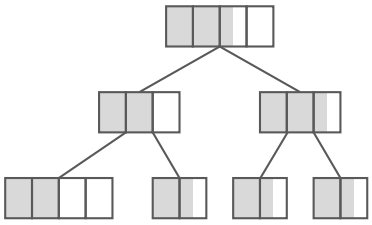
$$y(i,j) \leq x(i), \quad \forall i,j \tag{5}$$

$$y(i,j) = 0, \quad \forall i,j \notin J(Anc(i)) \tag{6}$$

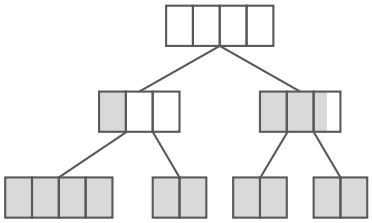
$$\sum_{i'\in Des(i)} x(i') \geq 2, \quad \forall i, \mathrm{OPT}_i \geq 2 \tag{7}$$

$$\sum_{i'\in Des(i)} x(i') \geq 3, \quad \forall i, \mathrm{OPT}_i \geq 3 \tag{8}$$

(a) Linear program for active time scheduling. By default we restrict $i \in [m]$ and $j \in J$.



(b) The open slots from an LP solution.



(c) The open slots after performing the LP transformation.

Figure 1 (a) is the linear program for active time schuduling. (b) and (c) are an example of a tree before and after running the LP transformation in Lemma 2. The dark slots represent slots have jobs scheduled in them, and the white slots are closed.

For simplicity, given a node set $V \subset [m]$ and $J' \subset J$, $y(V,J') = \sum_{i\in V, j\in J'} y(i,j)$; if either V or J' is a singleton, we can replace it by the unique element in it. Let $x(S) = \sum_{i\in S} x(i)$ for every $S \subseteq [m]$.

After running the LP and getting a solution (x,y), we will perform a transformation on the solution.

3.2 Transformation of LP Solution

▶ **Lemma 2.** *Given a feasible LP solution, we can efficiently output another feasible LP solution such that for any pair of nodes i_1, i_2 such that $i_2 \in Des^+(i_1)$, if $x(i_2) < L(i_2)$, then $x(i_1) = 0$.*

Proof. Suppose there are nodes i_1, i_2 with $i_2 \in Des^+(i_1)$ and $x(i_2) < L(i_2)$ and $x(i_1) > 0$. Then let $\theta = \min\{L(i_2) - x(i_2), x(i_1)\} > 0$. We can move θ fractional open slots from i_1 to i_2. For every job j, we move $\frac{\theta}{x(i_1)} y(i_1,j)$ fractional assignment of j from i_1 to i_2. More specifically, let $(x',y') = (x,y)$ initially. We decrease $x'(i_1)$ by θ and increase $x'(i_2)$ by θ. For every $j \in J$, we decrease $y'(i_1,j)$ to $\frac{x'(i_1)}{x(i_1)} y(i_1,j)$ and increase $y'(i_2,j)$ to $y'(i_2,j) + \frac{\theta}{x(i_1)} y(i_1,j)$. Notice that every job j that can be assigned to i_1 can also be assigned to i_2. It is not hard to show that all constraints remain satisfied by the new solution (x',y'). Finally we update $(x,y) \leftarrow (x',y')$.

Notice that after the operation, we have either $x(i_1) = 0$ or $x(i_2) = L(i_2)$. By repeating the procedure a polynomial number of times, we can find a solution (x,y) satisfying the property of the lemma. ◀

An example of of the LP transformation is shown in Figure 1(b) and 1(c). Lemma 2 implies that for any i with $x(i) > 0$, all of its strict descendants are fully open. We let I be the set of topmost nodes i with $x(i) > 0$; those are the nodes i with $x(i) > 0$ but all its strict ancestors i' have $x(i') = 0$.

▷ Claim 3. The following properties hold for I:

(3a) No node in I is a strict ancestor of another node in I.
(3b) $Des(I)$ contains all leaves.
(3c) Every $i \in I$ has $x(i) > 0$.
(3d) For any $i \in I$ and $i' \in Des^+(i)$, we have $x(i') = L(i')$.
(3e) For any $i \in I$ and $i' \in Anc^+(i)$, we have $x(i') = 0$.

We make one modification to the tree that does not change the instance. For every $i \in Anc^+(I)$, we can assume that i has exactly two children: if i has only one child, we remove it from the tree and connect its parent directly to its children. This does not change the instance since $x(i) = 0$.

We also want to mention for any node i such that $x(Des(i)) \in (1, 2)$, i has one child i', which is a leaf with $x(i') = L(i') = 1$ because of the rigidity of the leaf node.

3.3 Rounding Algorithm to Obtain an Integral Vector $\tilde{x} \in \{0, 1\}^{[m]}$

The rounding algorithm that constructs our integral $\tilde{x}$ is given in Algorithm 1.

Algorithm 1 Rounding Algorithm.

let $\tilde{x}(i) \leftarrow \lfloor x(i) \rfloor, \forall i \in I$ and $\tilde{x}(i) \leftarrow x(i), \forall i \in [m] \setminus I$.
for every node $i \in Anc(I)$ from bottom to top
  while $\frac{9x(Des(i))}{5} \geq \tilde{x}(Des(i)) + 1$
    if $\exists i' \in Des(i)$ with $\tilde{x}(i') < x(i')$ **then**
      choose such an i' arbitrarily
      let $\tilde{x}(i') \leftarrow \lceil x(i') \rceil$
    else
      break

Clearly, the number of open slots is at most $\frac{9x([m])}{5}$.

▶ **Lemma 4.** *After running the Algorithm 1,* $\tilde{x}([m]) \leq \frac{9x([m])}{5}$.

4 Feasibility of $\tilde{x}$

In this section, we show that the rounded time slot $\tilde{x}$ is a feasible solution.

4.1 A Necessary and Sufficient Condition

First we will give an if-and-only-if condition. From now on, for any set $J' \subseteq J$, we define $J'(i) = J(i) \cap J'$ for every $i \in [m]$ and $J'(S) = J(S) \cap J'$ for every $S \subseteq [m]$.

▶ **Lemma 5.** *Given an integer solution $\tilde{x}$ for the LP, $\tilde{x}$ is feasible if and only if for every subset $J' \subseteq J$ of jobs, we have*

$$\sum_{i \in [m]} \min\{|J'(Anc(i))|, g\} \cdot \tilde{x}(i) \geq p(J'). \tag{9}$$

Proof. The only if part is easy to see. Any node i can hold at most $\min\{|J'(Anc(i))|, g\} \cdot \tilde{x}(i)$ volume of jobs in J'. All the jobs in J' should be assigned. If $\tilde{x}$ is feasible, then $\sum_{i \in [m]} \min\{|J'(Anc(i))|, g\} \cdot \tilde{x}(i) \geq p(J')$.

Now we prove the if part, by considering the contra-positive of the statement and applying the maximum-flow-minimum cut theorem. Assume $\tilde{x}$ is not feasible. We construct a 4-layer directed network $H = (V_H, E_H)$, where the nodes from left to right are $\{s\}, J, [m]$ and $\{t\}$. There is an edge from s to every $j \in J$ with capacity p_j, an edge from every $j \in J$ to every $i \in Des(k(j))$ with capacity $\tilde{x}(i)$, and an edge from every $i \in [m]$ to t with capacity $g \cdot \tilde{x}(i)$. For a subsets $V' \subseteq V_H$ and a node $v \in V_H \setminus V'$, we use $E_H(V', v)$ to denote the set of edges from V' to v.

As $\tilde{x}$ is not feasible, there is a s-t cut in the network with capacity less than $p(J)$. Let (A, B) be the cut: $s \in A, t \in B$ and $A \uplus B = V_H$. Its cut value, which is $p(B \cap J) + \sum_{i \in B \cap [m]} |E_H(A \cap J, i)| \cdot \tilde{x}(i) + g \cdot \tilde{x}(A \cap [m])$, is less than $p(J)$. This is equivalent to $\sum_{i \in B \cap [m]} |E_H(A \cap J, i)| \cdot \tilde{x}(i) + g \cdot \tilde{x}(A \cap [m]) < p(A \cap J)$. Let $J' = A \cap J$. Then the contribution of a node $i \in [m]$ to the left-side is either $|E_H(J', i)| \cdot \tilde{x}(i)$ (if $i \in B$), or $g \cdot \tilde{x}(i)$ (if $i \in A$), which is lower bounded by $\min\{|E_H(J', i)|, g\} \cdot \tilde{x}(i)$. Noticing $|E_H(J', i)| = |J'(Anc(i))|$, we have $\sum_{i \in [m]} \min\{|J'(Anc(i))|, g\} \cdot \tilde{x}(i) < p(J')$. This finishes the proof of the if part. ◀

▶ **Lemma 6.** *In Lemma 5, it is sufficient to consider the sets $J' \subseteq J$ satisfying the following property:*

(6a) $p_j > \tilde{x}\Big(\big\{i \in Des(K(j)) : |J'(Anc(i))| \le g\big\}\Big), \forall j \in J'.$

Proof. Suppose J' does not satisfy the property. Then for some $j \in J'$ we have $\tilde{x}\Big(\big\{i \in Des(K(j)) : |J'(Anc(i))| \le g\big\}\Big) \ge p_j$. Then removing j from J' will decrease the left side of (9) by $\tilde{x}\Big(\big\{i \in Des(K(j)) : |J'(Anc(i))| \le g\big\}\Big)$, and the right side by p_j. Thus, the inequality (9) for J' will be implied by the inequality for $J' \setminus \{j\}$. ◀

Once we have the if-and-only-if condition for the feasibility, the main lemma we need to prove is the following:

▶ **Theorem 7.** *For every subset $J' \subseteq J$ of jobs satisfying the property in Lemma 6, we have* (9).

The rest of the section is dedicated to the proof of Theorem 7. From now on we fix a subset $J' \subseteq J$ satisfying the property of Lemma 6. Our goal is to prove (9).

For notational convenience, let $u_i = \min\{|J'(Anc(i))|, g\}$ and $w_i = u_i x(i)$ and $\tilde{w}_i = u_i \tilde{x}(i)$ for every $i \in [m]$. Thus, (9) is simply written as $p(J') \le \tilde{w}([m])$. Recall that $y(V, J') = \sum_{i \in V, j \in J'} y(i, j)$ for a given $V \subseteq [m], J' \subseteq J$. We have $p(J') = y([m], J') = y(Des(I), J')$ and $\tilde{w}([m]) = \tilde{w}(Des(I))$. Thus, we need to prove

$$y(Des(I), J') \le \tilde{w}(Des(I)), \tag{10}$$

for every $J' \subseteq J$ satisfying the property in Lemma 6.

The following simple claim will be used multiple times:

▷ Claim 8. For every $i \in [m]$, we have $y(i, J') \le w_i = u_i x(i)$.

Proof. If $u_i = g$, we use (3) in the LP. If $u_i < g$, then we use (5) and (6). ◁

4.2 Construction of Triples

For nodes $i \in [m] \setminus I$, we have $\tilde{x}(i) = x(i)$. For nodes $i \in I$ with $x(Des(i)) \notin (1, \frac{10}{9})$, we have $\tilde{x}(i) = \lceil x(i) \rceil$ since $\frac{9x(Des(i))}{5} \ge \lceil x(Des(i)) \rceil$. Thus, for these nodes i, Claim 8 implies $y(i, J') \le \tilde{w}_i$. The critical nodes are those $i \in I$ with $x(Des(i)) \in (1, \frac{10}{9})$.

With this in mind, we classify nodes in I into two types: a node $i \in I$ is of

- type-B if $x(Des(i)) \in \{1\} \cup [\frac{4}{3}, \infty)$, and
- type-C if $x(Des(i)) \in (1, \frac{4}{3})$.

In the definition, we use $\frac{4}{3}$ instead of $\frac{10}{9}$ to create some buffers. Furthermore, a type-C node $i \in I$ is of

- type-C_1 if $\tilde{x}(Des(i)) = 1$, and
- type-C_2 if $\tilde{x}(Des(i)) = 2$.

At most 2 type-C nodes. Before going to the triples, we first solve the case at most 2 type-C nodes are in I. At the same time, if 1 type-B node exists, then all type-C nodes are type-C_2.

▶ **Lemma 9.** *If there are at most 2 type-C nodes and at least 1 type-B node in I, then all type-C are type-C_2.*

Proof. Let i_1 and i_2(if exists) be the type-C node and i_3 be the type-B node. Notice that $\frac{9}{5}x(i_3) - \lceil x(i_3) \rceil \geq 0.4$. We have $\frac{9}{5}(x(i_1)+x(i_3)) \geq x(i_1)+0.8+\lceil x(i_3) \rceil + 0.4 \geq \lceil x(i_1) \rceil + \lceil x(i_3) \rceil$ and $\frac{9}{5}(x(i_1)+x(i_2)+x(i_3)) \geq x(i_1)+0.8+x(i_2)+0.8+\lceil x(i_3) \rceil + 0.4 \geq \lceil x(i_1) \rceil + \lceil x(i_2) \rceil + \lceil x(i_3) \rceil$. In either case, algorithm 1 can afford to round up all type-C nodes. ◀

Based on Lemma 9, if at least one type-B node is in I, then we already rounded up all type-C nodes. Assume that there is no type-B node in I, then we have $x([m]) \leq 2 \times 4/3 = 8/3$. Due to the LP constraint (7) and (8), $x(i)$ are integer for all $i \in [m]$ and this contradicts to the assumption that no type-B node is in I.

More than 2 type-C nodes. When we have at least 3 type-C nodes in I, we want to create disjoint triples of type-C nodes. Each triple contains 1 type-C_1 node, and 2 type-C_2 nodes. Moreover, all the type-C_1 nodes are contained in these triples. Later for each triple (i_1, i_2, i_3) we constructed, we shall prove $y(Des(\{i_1, i_2, i_3\}), J') \leq \tilde{w}(Des(\{i_1, i_2, i_3\}))$. This will prove (10).

The construction of triples are given in Algorithm 2. Notice that this is not a part of our algorithm for solving the active time scheduling problem; it is only used in the analysis. If we have a type-C_1 node i_1 and a type-C_2 node i_2 as brothers, then we say (i_1, i_2) is a C_1C_2-brother-pair. In our triples, we make sure that we do not break C_1C_2-brother-pairs: for such a pair (i_1, i_2), there must be some C_2-node i_3 such that (i_1, i_2, i_3) is a triple we constructed.

Algorithm 2 Construction of Triples.

1: triples $\leftarrow \emptyset$, set all type-C_1 nodes as uncovered, and all type-C_2 nodes as unused.
2: **for** every node $i \in Anc(I)$ with $|Des(i) \cap I| \geq 3$ from bottom to top
3: **while** $\exists$ an uncovered type-C_1 node $i_1 \in Des(i)$
4: choose two unused type-C_2 nodes $i_2, i_3 \in Des(i)$, without breaking C_1C_2-brother-pairs ▷ See Lemma 10 and Lemma 11
5: add (i_1, i_2, i_3) to triples, claim i_1 is covered, and i_2 and i_3 are used.

▶ **Lemma 10.** *In Step 4 of Algorithm 2, there are at least two unused type-C_2 nodes in $Des(i)$.*

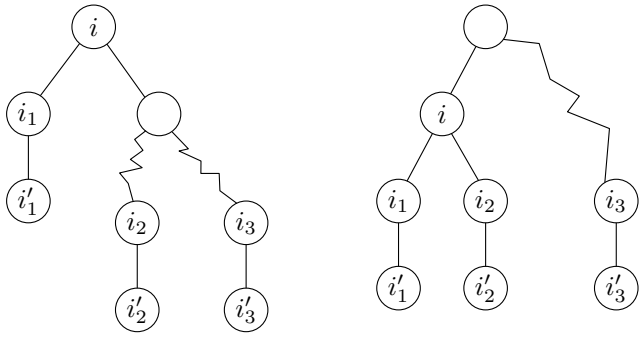


Figure 2 The two cases in Lemma 11 and 12.

Proof. Let n_1 and n_2 be the number of type-C_1 and type-C_2 nodes in $Des(i) \cap I$ respectively. Let $n' = n_1 + n_2$. We shall prove $n_2 \geq 2n_1$, which implies that we will not run out of type-C_2 nodes and proves the lemma.

First, we consider the case where there is no type-B node in $Des(i) \cap I$. Then $n' = |Des(i) \cap I| \geq 3$. Moreover, $n_1 + 2n_2 \geq \lfloor \frac{9n'}{5} \rfloor$ or $n_1 = 0$ by our rounding algorithm in Algorithm 1. In the latter case we clearly have $n_2 \geq 2n_1$. So, we assume the former. Then $n_2 \geq \lfloor \frac{4n'}{5} \rfloor$. Notice that $\lfloor \frac{4n'}{5} \rfloor \geq \frac{4n'}{5} - \frac{4}{5}$ and thus $\lfloor \frac{4n'}{5} \rfloor \geq \frac{2n'}{3}$ whenever $n' \geq 6$. One can check that when $n' \in \{3, 4, 5\}$ we have $\lfloor \frac{4n'}{5} \rfloor \geq \frac{2n'}{3}$. Therefore if $n' \geq 3$, we have $\lfloor \frac{4n'}{5} \rfloor \geq \frac{2n'}{3}$. So $n_2 \geq 2n_1$.

Now we consider the other case where there is at least one type-B node in $Des(i) \cap I$. If $n_1 = 0$, then $n_2 \geq 2n_1$ and thus we assume $n_1 > 0$. Then we have $n_1 + 2n_2 \geq \lfloor \frac{9n'}{5} + \frac{2}{5} \rfloor$, as the type B node i^* have $\frac{9x(Des(i^*))}{5} \geq \lceil x(Des(i^*)) \rceil + \frac{2}{5}$. This implies $n_2 \geq \lfloor \frac{4n'}{5} + \frac{2}{5} \rfloor$. Then $n_2 \geq \frac{2n'}{3}$ as $\lfloor \frac{4n'}{5} + \frac{2}{5} \rfloor \geq \frac{2n'}{3}$ for every integer $n' \geq 0$. Thus, $n_2 \geq 2n_1$. ◀

4.3 Proof of (10) Using the Triples

In this section, we prove (10) by using the constructed triples. First, we show that they satisfy some good properties:

▶ **Lemma 11.** *For every $(i_1, i_2, i_3) \in \mathsf{triples}$, one of the following two conditions hold:*

(11a) *$i_2, i_3 \in Des^+(\mathsf{par}(i_1))$.*

(11b) *i_1 and i_2 are brothers, and $i_3 \in Des^+(\mathsf{par}(\mathsf{par}(i_1)))$.*

Proof. Consider any type-C_1 node i_1. Let i^* and i' be the parent and brother of i_1 respectively.

First consider the case that $i' \notin I$. By (3a) and (3b), we have $|Des(i^*) \cap I| \geq 3$. By Algorithm 2 and Lemma 10, i' will be covered when we are at the iteration $i = i^*$ in Algorithm 2. So, the first property holds.

Then consider the other case that $i' \in I$. Then i' can not be of type-B since otherwise i_1 should be of type-C_2. i' can not be of type-C_1 since we could open 3 slots in $Des(i^*)$. Therefore i' must be of type-C_2 and (i_1, i') is a $\mathsf{C}_1\mathsf{C}_2$-brother-pair. In this case the second property holds. ◀

See Figure 2 for the two cases obtained in Lemma 11, which will be used again in the proof of the following Lemma:

▶ **Lemma 12.** *For every* $(i_1, i_2, i_3) \in \mathsf{triples}$*, we have*

$$y(Des(\{i_1, i_2, i_3\}), J') \le \tilde{w}(Des(\{i_1, i_2, i_3\})).$$

Proof. Recall that i_1 is of type C_1 and i_2 and i_3 are of type-C_2. Let i'_1, i'_2 and i'_3 be the children of i_1, i_2 and i_3 respectively. By Lemma 11, either a or b holds.

We first assume a. Let $i = \mathsf{par}(i_1)$. See Figure 2(a) for all illustration of nodes used in this case.

By (7) in the LP, $J(Des(i_1))$ can be scheduled in the one slot in i'_1 since $x(Des(i_1)) < 2$. That is, all the jobs in the set has size 1 and there are at most g of them. So, we have

$$y(Des(i_1), J'(Des(i_1))) \le |J'(Des(i_1))| \le u_{i'_1} x(i'_1). \tag{11}$$

$$\begin{aligned} y(Des(i_1), J'(Anc^+(i_1))) &\le \frac{10}{9} \min\{|J'(Anc^+(i_1))|, g\} \\ &\le (\tilde{x}(i_2) - x(i_2) + \tilde{x}(i_3) - x(i_3)) \min\{|J'(Anc^+(i_1))|, g\} \\ &\le (\tilde{x}(i_2) - x(i_2)) u_{i_2} + (\tilde{x}(i_3) - x(i_3)) u_{i_3} \end{aligned} \tag{12}$$

$$y(Des(\{i_2, i_3\}), J') \le u_{i_2} x(i_2) + u_{i'_2} x(i'_2) + u_{i_3} x(i_3) + u_{i'_3} x(i'_3) \tag{13}$$

The first inequality of (11) is by that all jobs in $J'(Des(i_1))$ have size 1, and the second one is by $u_{i'_1} = \min\{|J'(Anc(i'_1))|, g\} \ge |J'(Des(i_1))|$ and $x(i'_1) = 1$. The first inequality of (12) is by that $x(Des(i_1)) < \frac{10}{9}$. The second one follows from $\tilde{x}(i_2) - x(i_2) \ge \frac{2}{3}$ and $\tilde{x}(i_3) - x(i_3) \ge \frac{2}{3}$. The last one used that every job in $J'(Anc^+(i_1))$ can be assigned to i_2 and i_3. (13) is by Claim 8.

Adding (11), (12) and (13), we have

$$\begin{aligned} y(Des(\{i_1, i_2, i_3\}), J') &\le u_{i'_1} x(i'_1) + u_{i_2} \tilde{x}(i_2) + u_{i'_2} x(i'_2) + u_{i_3} \tilde{x}(i_3) + u_{i'_3} x(i'_3) \\ &= \tilde{w}(Des(\{i_1, i_2, i_3\})) \end{aligned}$$

Now we consider the case that b holds. Let $i = \mathsf{par}(i_1) = \mathsf{par}(i_2)$. See Figure 2(b) for illustration of the nodes.

First, if $u_{i_2} = g$, then we have $(\tilde{x}(i_2) - x(i_2)) u_{i_2} \ge (x(i_1) - \tilde{x}(i_1)) u_{i_1}$ as $\tilde{x}(i_2) - x(i_2) \ge \frac{2}{3}$ and $x(i_1) - \tilde{x}(i_1) < \frac{1}{9}$. This is $\tilde{w}(i_1) + \tilde{w}(i_2) \ge w(i_1) + w(i_2)$. Thus, $y(Des(\{i_1, i_2, i_3\}), J') \le w(Des(\{i_1, i_2, i_3\})) \le \tilde{w}(Des(\{i_1, i_2, i_3\}))$ as $\forall i' \in \{i_3, i'_1, i'_2, i'_3\}$ we have $w(i') \le \tilde{w}(i')$.

So assume $u_{i_2} < g$. As we assumed J' satisfies (6a), every job $j \in J'(Anc(i_2))$ has $p_j > 1$. All jobs in $J(i)$ have $p_j \le 2$ since otherwise we would have $x(Des(i)) \ge 3$ by LP constraint (2) and (5). Also all jobs in $J(i_2)$ have size 1. So all jobs in $J'(i)$ have size 2, and $J'(i_2) = \emptyset$.

Then we have

$$y(Des(i_1), J'(Des(i_1))) + |J'(i)| \le |J'(Des(i_1))| + |J'(i)| \le u_{i'_1} x(i'_1). \tag{14}$$

$$\begin{aligned} y(Des(i_1), J'(Anc^+(i))) &\le \frac{10}{9} |J'(Anc^+(i))| \le \big(\tilde{x}(i_2) - x(i_2) + \tilde{x}(i_3) - x(i_3)\big) |J'(Anc^+(i))| \\ &\le (\tilde{x}(i_2) - x(i_2)) |J'(Anc^+(i))| + (\tilde{x}(i_3) - x(i_3)) u_{i_3} \end{aligned} \tag{15}$$

$$\begin{aligned} y(Des(\{i_2, i_3\}), J' \setminus J'(i)) + |J'(i)| &\le |J'(Anc^+(i))| \cdot x(i_2) + u_{i'_2} x(i'_2) + u_{i_3} x(i_3) \\ &\quad + u_{i'_3} x(i'_3) + |J'(i)| \cdot \tilde{x}(i_2) \end{aligned} \tag{16}$$

The first inequality of (14) all jobs in $J'(Des(i_1))$ have size 1, and the second inequality is by that $|J'(Des(i_1))| + |J'(i)| \le g$ since otherwise $\mathrm{OPT}_i \ge 3$. The proof of (15) is similar to that of (12). Notice that every job in $J'(Anc^+(i))$ can be assigned to i_2 and i_3. The inequality in (16) used Claim 8 for i'_2, i_3 and i'_3.

$$\begin{aligned}
&\min \sum_{t\in\mathcal{T}} x(t) \\
\text{s.t.} \quad &\sum_{t\in[r_j,d_j)} y(t,j) \geq p_j && \forall j \in J \\
&\sum_{j\in J} y(t,j) \leq g \cdot x(t) && \forall t \in \mathcal{T} \\
&y(t,j) \leq x(t) && \forall t \in \mathcal{T}, \forall j \in J \\
&x(t) \leq 1 && \forall t \in \mathcal{T} \\
&\sum_{t\in[t_1,t_2)} x(t) \geq \left\lceil \frac{\sum_{j\in J} q_j(I)}{g} \right\rceil && I = [t_1,t_2), \forall t_1 \in \mathcal{T}, \forall t_2 \in \mathcal{T}
\end{aligned}$$

Figure 3 Călinescu and Wang's linear program for active time scheduling [6].

Adding (14), (15) and (16), we get

$$\begin{aligned}
&y(Des(i_1,i_2,i_3), J' \setminus J'(i)) + 2|J'(i)| \\
&\leq u_{i'_1} x(i'_1) + (|J'(Anc^+(i)| + |J'(i)|)\tilde{x}(i_2) + u_{i'_2} x(i'_2) + u_{i_3}\tilde{x}(i_3) + u_{i'_3} x(i'_3) \\
&= u_{i'_1} x(i'_1) + u_{i_2}\tilde{x}(i_2) + u_{i'_2} x(i'_2) + u_{i_3}\tilde{x}(i_3) + u_{i'_3} x(i'_3) = \tilde{w}(Des(i_1,i_2,i_3)).
\end{aligned}$$

Notice that $y(Des(i_1,i_2,i_3), J'(i)) = p(J'(i)) = 2|J'(i)|$, we have $y(Des(i_1,i_2,i_3), J') \leq \tilde{w}(Des(i_1,i_2,i_3))$. ◀

With Lemma 12, we can prove (10). First $\sum_{i\in *} y(Des(i), J') \leq \sum_{i\in *} \tilde{w}(Des(i))$, where $i \in *$ is over all nodes in the triples we constructed. For all the other nodes $i \in I$, we have $y(Des(i), J') \leq w(Des(i)) \leq \tilde{w}(Des(i))$. Therefore we have $y(Des(I), J') \leq \tilde{w}(Des(I))$, which is exactly (10). Combining with Lemma 4, we have following theorem,

▶ **Theorem 13.** *There exists a 1.8-approximation polynomial-time algorithm for the nested active-time problem.*

5 Integrality Gap

For the general (non-nested) version of the active scheduling problem, Călinescu and Wang [6] proposed a slightly stronger than our LP from Figure 1a and showed a non-nested instance where the integrality gap approaches 5/3 as $g \to \infty$. In this section, we show that their LP and our LP have an integrality gap of at least 3/2 on nested instances.

To define their LP, we need some notation. Let $\mathcal{T} = [\min_{j\in J} r_j, \max_{j\in J} d_j)$ denote the set of time steps between the earliest release time and the latest deadline. For an interval of time $I = [t_1, t_2)$ for some $t_1, t_2 \in \mathcal{T}$ and a job j, let $q_j(I)$ denote the minimum number of slots within I that job j needs to occupy in a feasible solution even if all slots outside of I were active and available to j. The variable $x(t)$ denotes the extent to which the slot t is active and $y(t,j)$ denotes the extent to which job j is assigned to slot t. See Figure 3.

▶ **Lemma 14.** *The linear program in Figure 3 has an integrality gap of at least* 3/2 *on nested instances.*

Proof. The integrality gap instance consists of one long job j_0 with processing time g and window $[0, 2g)$, and g groups of g jobs. For $0 \leq i < g$, the i-th group consists of g jobs with unit processing time and window $[2i, 2i+2)$.

Consider the following LP solution (x, y): open each slot $t \in \mathcal{T} = [0, 2g)$ to an extent of $x(t) = (g + 2)/2g$ for a total of $g + 2$. For each $0 \leq i < g$, the LP schedules $1/2$ unit of j_0 and $1/2$ unit of each job j in the i-th group in slots $2i$ and $2i + 1$; that is, $y(2i, j_0) = y(2i + 1, j_0) = 1/2$ and $y(2i, j) = y(2i + 1, j) = 1/2$ for each job j in group i.

We now argue that (x, y) satisfies the ceiling constraints of [6]'s LP; it is easy to check that the other constraints are satisfied. Consider an interval I. Since the long job j_0's window is $[0, 2g)$ and j_0 has length g, we have that $q_{j_0}(I) = 0$ if $|I| \leq g$ and $q_{j_0}(I) = |I| - g$ if $|I| > g$. This is because there are $2g - |I|$ slots outside of I. For a job j in group i, since it has unit length, we have $q_j(I) = 1$ if I contains its window $[2i, 2i + 2)$ and $q_j(I) = 0$ otherwise.

Combining the above, the LP constraint on interval I is

$$\sum_{t \in I} x(t) \geq \left\lceil \frac{\max\{0, |I| - g\} + g|\{i \mid I \supseteq [2i, 2i + 2)\}|}{g} \right\rceil .$$

The tightest constraints are when I is the union of windows of consecutive groups. Thus, it suffices to argue that these constraints are satisfied. Suppose $I = [2i', 2(i' + k - 1) + 2)$, i.e., it is the union of windows of k consecutive groups. Note that $|I| = 2k$

If $k \leq g/2$, then the LP constraint on I is $\sum_{t \in I} x(t) \geq k$. This is satisfied since $x_{2i} + x_{2i+1} = (g + 2)/g$ for each group i. If $k > g/2$, then the LP constraint on I is $\sum_{t \in I} x(t) \geq 1 + k$. This is also satisfied since $\sum_{t \in I} x(t) = k(g + 2)/g = k + 2k/g > k + 1$. Thus, (x, y) is a feasible solution to the LP that opens $g + 2$ slots fractionally.

We claim that any integral solution (x', y') needs to open at least $3g/2$ slots. Let k be the number of groups i such that y' schedules at least one unit of the long job j_0 in the window $[2i, 2i + 2)$. Consider the window $[2i, 2i + 2)$. Since there are g unit jobs that need to be scheduled in $[2i, 2i + 2)$, x' opens two slots in the window if y' schedules j_0 in the window and only one slot otherwise. Thus, y' opens exactly $g + k$ slots. Since each window $[2i, 2i + 2)$ has only two slots, and j_0 has length g, we have that $k \geq g/2$. Therefore, any integral solution needs to open at least $3g/2$ slots. Thus, the integrality gap of the LP is at least $\frac{3g}{2(g+2)}$ which converges to $3/2$ for large g. ◀

6 NP-Completeness

In this section, we show that the decision version of the nested active time problem is NP complete. Very recently, Sagnik and Manish [12] showed the general case is NP complete. Unfortunately, their proof uses crossing intervals (i.e., intervals that overlap but neither is included in the other). Our proof reduces the nested active time problem to a new problem that we call **prefix sum cover**, which is related to the classic set cover problem.

Prefix sum cover problem

For any pair of d-dimensional vectors $v = (v_1, v_2, ..., v_d), w = (w_1, w_2, ..., w_d) \in R^d$, we say $v \prec w$ if and only if for all $j \in [1, d]$, $\sum_{i \leq j} v_i \geq \sum_{i \leq j} w_i$. In the prefix sum cover problem, we are given n vectors $u_1, u_2, ...u_n \in N_+^d$, a target vector $v \in N^d$ and an integer number k, and we want to find k vectors $u_{l_1}, u_{l_2}, ..., u_{l_k}$ such that $\sum_{i \leq k} u_{l_i} \prec v$.

Moreover, for the purposes of our reduction, we consider a restricted version of the problem. Let W be the maximum scalar that appears in any of the vectors $u_1, ..., u_n$ and v. First, we require that both d and W be bounded by some polynomial of n. For a vector $w \in N^d$, let $[w]_j$ be its j-th dimension value. Second, for each $i \in [1, n]$, we require that $[u_i]_1 \geq [u_i]_2 \geq ... \geq [u_i]_d$, and $[v]_1 \geq [v]_2 \geq ... \geq [v]_d$, i.e., all vectors are non-decreasing. Lastly, we require that all vectors are non-negative and integral.

We show in Appendix 6 that prefix sum cover is NP complete even under these restriction.

▶ Remark. Notice that the prefix sum cover problem is almost the same as set cover problem except for the "order" relation. We can think of the set cover problem as requiring that each dimension of the sum vector is greater than the target vector, while in the prefix sum cover problem, the requirement is "prefix sum".

Reduction

Now we will reduce the prefix sum cover problem to the active time problem. Let $(\{u_1, u_2, \ldots, u_n\}, v, k)$ be the prefix sum cover instance. Our nested active time instance is defined by a set of jobs J and it uses $p = dW$ machines. Our instance is made up of three kinds of jobs:

- For each vector u_i, and each $w \in [2, W]$, we have $p - |\{j \in [1, d] \mid [u_i]_j \geq w\}|$ rigid unit length jobs, each with window consisting of a single slot $[(i-1)W + w - 1, (i-1)W + w]$.
- For each vector u_i, we also have $\sum_{j \leq d}[u_i]_j - d$ flexible unit jobs with window $[(i-1)W, iW]$.
- Finally, we have jobs that depend on the target vector. For each $j \in [1, d]$, we have a job with length $[v]_j$ and window $[0, nW]$.

We denote each of these sets of job with S_1 (rigid jobs), S_2 (flexible jobs associate with each u_i vector), and S_3 (jobs associated with the target vector).

Let us try to schedule this instance, starting with S_1. Since the jobs in S_1 are rigid, we must open all slots in $[(i-1)W + 1, iW]$. Notice that each of these slots has at least $p - d$ jobs in S_1, so each of these time slots has at most d unused machines after scheduling S_1.

Next we will try to fit jobs from S_2 into $[(i-1)W, iW]$. Observe that the total capacity in the window $[(i-1)W + 1, iW]$ is $p(W-1)$ and that the jobs from S_1 take up up $\sum_{w \in [2,W]}(p - |\{j \in [1, d] \mid [u_i]_j \geq w\}|)$ capacity. Further observe that

$$\sum_{w \in [2,W]} (p - |\{j \in [1, d] \mid [u_i]_j \geq w\}|) + \sum_{j \leq d} [u_i]_j - d = p(W-1).$$

Therefore, if we do not open the slot $[(i-1)W, (i-1)W + 1]$, then the jobs from S_1 and S_2 will use up all of the available capacity in the time window $[(i-1)W, iW]$. This is important, as it means that we cannot schedule any job from S_3 in this window.

We say that the time slots $[(i-1)W, (i-1)W + 1]$ for $i \in [n]$ are *special*. Since all non-special slots in $[0, nW]$ must be open, the problem boils down to opening as few special slots as possible to accommodate the jobs in S_3.

Suppose we open the special time slot $[(i-1)W, (i-1)W + 1]$. We claim that all jobs in S_2 will be assigned to the special time slot. Indeed, even after all S_2 jobs are assigned to this slot, there are still $p - (\sum_{j \leq d}[u_i]_j - d) \geq d$ unused machines in it, while we can only have at most d unused machines in each time slots in $[(i-1)W + 1, iW]$ after scheduling S_1.

Let *configuration* be a sequence $(z_1, z_2, ..., z_M)$, where z_i is the number of machines unused in time slot $[i-1, i]$. Thus, once we have chosen which special slots to open, we get the configuration which tells us how many machines are left unused in each time slot. In the remainder of this section, we give an if-and-only-if condition on whether a configuration can fit all jobs from S_3.

Assume the machines are numbered from 1 to p. For any given configuration, we can assume without loss of generality that if we have z_t unused capacity at time slot $[t-1, t]$ then machines 1 through z_t are unused; i.e, we always leaves smaller index machine unused. Let e_j be the number of empty time slots at machine j. We give an if-and-only-if condition for the feasibility based on the e_j values.

▶ **Lemma 15.** *Given a configuration, let e_j be the machine unused slot defined above and J' be a set of $q \leq p$ jobs with no release time and due time constraint. Let $l_1 \geq l_2 \geq ... \geq l_q$ be the lengths of the jobs in J. The configuration can fit all jobs in J' if and only if $\sum_{i \leq j} e_i \geq \sum_{i \leq j} l_i$ for all $j \in [1, q]$.*

Proof. To identify each job, when we say the i-th job, we refer the job with length l_i.

If part. Suppose $\sum_{i\le j} e_i \ge \sum_{i\le j} l_i$ for all $j \in [1,q]$. Now, we prove the following statement by induction on k: if $\sum_{i\le k} e_i \ge \sum_{i\le k} l_i$ for any k, then we can fit jobs $l_1, l_2, ..., l_k$ into the first k machines. The base case is $k = 1$: since $e_1 \ge l_1$, then we can fit l_1 into the first machine. Now we prove the inductive case. Suppose we can fit the first k jobs into the first k machines and we want to fit the first $k+1$ jobs into the first $k+1$ machines. Now, we first try to fit the first job to the first $k+1$ machines and then use our induction. We fit the first job in following sequence: we first use machine $k+1$ and if we cannot fit all of the first job in it, we use the machine k and repeat this process, i,e, fit the first job by using machine with decreasing index. Notice that we use at most e_1 time slots to fit the first job, thus the above process will finally fit the first job inside but use some time slot of machines from $k+1$ to 1. Now, let $e'_1, e'_2, e'_3 ..., e'_{k+1}$ be the machine unused time slot after we fit the first job inside. We know that $e'_j \ge e_{j+1}$ for $j \in [1,k]$ since a job cannot use the same time slot twice. Another point is if $e'_j < e_j$, i.e, we use some time slot of machine j, then $\sum_{i>j}(e_i - e'_i) = e_{j+1}$. Now, notice that we have jobs $l_2, l_3, ..., l_{k+1}$, and the new time slot is $e'_1, e'_2, ...e'_k$, if we can show $\sum_{i\le j} e'_i \ge \sum_{i\le j} l_{i+1}$ for all $j \le k$, then by induction, we can fit the second to the $(k+1)$-th jobs to the first k machines. If $e'_j = e_j$, then we can say $e'_{j'} = e_{j'}$ for all $j' \le j$, and thus $\sum_{i\le j} e'_i = \sum_{i\le j} e_i \ge \sum_{i\le j} l_i \ge \sum_{i\le j} l_{i+1}$. If $e'_j < e_j$, we have $l_1 = \sum_{i\le k}(e_i - e'_i) = e_{j+1} + \sum_{i\le j}(e_i - e'_i)$, thus

$$\sum_{i\le j+1} e_i \ge \sum_{i\le j+1} l_i = l_1 + \sum_{i\le j} l_{i+1} = e_{j+1} + \sum_{i\le j}(e_i - e'_i) + \sum_{i\le j} l_{i+1}$$
$$\Rightarrow \sum_{i\le j} e_i - \sum_{i\le j}(e_i - e'_i) \ge \sum_{i\le j} l_{i+1} \Rightarrow \sum_{i\le j} e'_i \ge \sum_{i\le j} l_{i+1}$$

In either case, we know that the remaining jobs could be fitted into machines from 1 to k, thus we could fit all jobs if for all $j \in [1,p]$, $\sum_{i\le j} e_j \ge \sum_{i\le j} l_j$.

Only if part. If for some $j \in [1,q]$, $\sum_{i\le j} e_i < \sum_{i\le j} l_i$, then the configuration is impossible to fit jobs $l_1, l_2, ..., l_j$. Consider a time slot $[t-1, t]$, when we fit a job inside, we always use the machine with smallest index, if we could fit $l_1, l_2, ..., l_j$ into the time slot, then we will use machines with index at most j. If at any time slot, we use a machine with index greater than j, then we know we already use all machines from 1 to j, however, we have only j jobs now. Thus, if we could fit the first j jobs into the time slot, we can use the first j machines to fit those jobs. However, for the first j machines, the total capacity $\sum_{i\le j} e_i$ is less than the length of all jobs, i.e, $\sum_{i\le j} l_i$. Therefore, it is not possible to fit the first j jobs into the configuration. ◀

Now we show how to apply it to the active time instance. We will set $J = S_3$ and $q = d$. For any interval $[(i-1)W, iW]$, let $e_{1,i}, e_{2,i}, ..., e_{d,i}$ be the unused time slot for machine 1 to d in this interval. If we close the special time slot $[(i-1)W, (i-1)W+1]$, then there is no capacity left so $e_{1,i} = e_{2,i} = ... = e_{d,i} = 0$. If we open it, then $e_{j,i} = [u_i]_j$, i.e. the j-th machine will hold exactly $[u_i]_j$ unused time slots in the interval. Now the problem becomes we want to open k special time slots such that the resulting configuration can fit all jobs from S_3. Lemma 15 implies that it is equivalent to choosing k vectors from $(e_{1,1}, \ldots, e_{d,1}) = u_1, ..., (e_{1,n}, \ldots, e_{d,n}) = u_n$ such that $\sum_{i\le j} e_i \ge \sum_{i\le j}[v]_i$ for every $j \in [1,d]$, which is exactly the definition of our prefix sum cover problem. Note that the ordering requirement comes from the fact we have ordering requirement in Lemma 15 and

the positiveness of u comes from the fact that the machine from 1 to d has 1 unused space at time slot $[(i-1)W, (i-1)W+1]$ if we open it. Since d, W are all polynomial, the interval length and the machine number p are also polynomial.

References

1 Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.
2 Jessica Chang, Harold N. Gabow, and Samir Khuller. A model for minimizing active processor time. *Algorithmica*, 70(3):368–405, 2014.
3 Jessica Chang, Samir Khuller, and Koyel Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. *J. of Scheduling*, 20(6):657–680, 2017.
4 Vincent Chau and Minming Li. *Active and Busy Time Scheduling Problem: A Survey*, pages 219–229. Springer International Publishing, 2020.
5 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *Proc. of the 45th Symposium on Foundations of Computer Science*, pages 81–90, 2004.
6 Gruia Călinescu and Kai Wang. A new lp rounding algorithm for the active time problem. *Journal of Scheduling*, pages 1–10, March 2021.
7 Sami Davies, Samir Khuller, and Shirley Zhang. Balancing flow time and energy consumption. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, pages 369–380, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3490148.3538582.
8 Sandy Irani and Kirk R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
9 Frederic Koehler and Samir Khuller. Busy time scheduling on a bounded number of machines (extended abstract). In *Proc. of the 15th International Symposium on Algorithms and Data Structures*, pages 521–532, 2017.
10 Saurabh Kumar and Samir Khuller. Brief announcement: A greedy 2 approximation for the active time problem. In *Proc. of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 347–349, 2018.
11 Mozhengfu Liu and Xueyan Tang. Analysis of busy-time scheduling on heterogeneous machines. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, pages 340–350, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3409964.3461795.
12 Sagnik Saha and Manish Purohit. Np-completeness of the active time scheduling problem. *arXiv preprint arXiv:2112.03255*, 2021.
13 Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. Trends in worldwide ICT electricity consumption from 2007 to 2012. *Computer Communications*, 50:64–76, 2014.
14 Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

A NP Completeness of the prefix sum cover problem

Proof. We will reduce set cover problem to the prefix sum cover problem. Recall $[v]_i$ is the i-th index value of vector v. Consider a set cover instance, U is the universe containing d elements, S contains n sets and k is the target integer, the set cover problem is to find at most k subsets from S such that the union of those sets is the universe U. We could use a vector $u_i \in N^d$ to represent each set of S, for each index $j \in [1, d]$, if the set contains the j-th element, then we set the j-th value of u_i to be 1. Otherwise, it will be 0. the target vector $v = \mathbf{1}^d$. Now the set cover problem is to find at most k vectors $u_{l_1}, u_{l_2}, ..., u_{l_k}$ from

$u_1, u_2, ..., u_n$ such that for each $j \in [1, d]$, $[\sum_{i \le k} u_{l_i}]_j \ge [v]_j$. For technique problem, we will add 0-th index to the vector and set it to be 0 for both u and v. This won't affect our solution and this index is only helps for dealing with 1-th index.

Next, we will transform all vectors $u_1, u_2, ..., u_n$ to new vectors $u'_1, u'_2, ..., u'_n$. Specifically, for each vector $u_i = ([u_i]_1, [u_i]_2, ..., [u_i]_d)$, the new vector $u'_i = ([u'_i]_1, [u'_i]_2, ..., [u'_i]_d)$, where $[u'_i]_j = [u_i]_j - [u_i]_{j-1} + 2 + d - j$ for all $j \in [1, d]$. Notice that $[u_i]_j$ is either 0 or 1, thus $[u'_i]_j \in [1, d+2]$. Now for the target vector v, we will set the new vector $v' = ([v']_1, [v']_2, ..., [v']_d)$ such that $[v']_j = [v]_j - [v]_{j-1} + 2k + k(d-j)$. Again, since $[v]_j, [v]_{j-1} \in [0, 1]$, we have $[v']_j \in [2k-1, kd+k+1]$. Thus, the maximum value in the new vectors is at most $kd+k+1$, which is at most polynomial in n and fits our requirement of prefix sum cover problem. Last, for the ordering requirement, we have $[u'_i]_j - [u'_i]_{j-1} = [u_i]_{j-2} - [u_i]_{j-1} + 1 \ge 0$ and $[v']_j - [v']_{j-1} = [v]_{j-2} - [v]_{j-1} + k \ge 0$. Now we want to show the following if-and-only-if for the reduction.

If part. If we have a solution $u_{l_1}, u_{l_2}, ..., u_{l_k}$ for the set cover problem. If the solution contains less than k vectors, we could add some vectors to the solution until k vectors, this doesn't change the solution, so we could assume we have k vectors in the solution. Now, we want to show, the new vector $u'_{l_1}, u'_{l_2}, ..., u'_{l_k}$ is a solution for the partial sum problem. From set cover problem, we know $[\sum_{i \le k} u_{l_i}]_j \ge [v]_j$, for $j \in [0, d]$. Our target is to show $\sum_{i' \le j} \sum_{i \le k} [u'_{l_i}]_{i'} \ge \sum_{i' \le j} [v']_{i'}$, for $j \in [1, d]$. Notice that

$$
\begin{aligned}
&\sum_{i' \le j} \sum_{i \le k} [u'_{l_i}]_{i'} - \sum_{i' \le j} [v']_{i'} \\
=&\sum_{i \le k} \sum_{i' \le j} ([u_{l_i}]_{i'} - [u_{l_i}]_{i'-1} + 2 + d - i') - \sum_{i' \le j} ([v]_{i'} - [v]_{i'-1} + 2k + k(d-j)) \\
=&\sum_{i \le k} ([u_{l_i}]_j + 2j + \frac{(2d-1-j)j}{2}) - ([v]_j + 2jk + \frac{(2d-1-j)jk}{2}) \\
=&\sum_{i \le k} [u_{l_i}]_j - [v]_j = [\sum_{i \le k} u_{l_i}]_j - [v]_j \ge 0
\end{aligned}
$$

Thus, the new vectors are the solution for the partial problem.

Only if. If we have a solution $u'_{l_1}, u'_{l_2}, ..., u'_{l_k}$ for the prefix sum cover problem. Again, if the solution contains less than k vectors, we add some vectors to the solution. Notice that all number in the vector are non-negative, thus, the new solution is still feasible. Now, based on the above equation, the vector $u_{l_1}, u_{l_2}, ..., u_{l_k}$ is a solution for the set cover problem. ◀

On Algorithmic Self-Assembly of Squares by Co-Transcriptional Folding

Szilárd Zsolt Fazekas
Akita University, Japan

Hwee Kim
Incheon National University, Republic of Korea

Ryuichi Matsuoka
The University of Electro-Communications, Tokyo, Japan

Shinnosuke Seki
The University of Electro-Communications, Tokyo, Japan

Hinano Takeuchi
The University of Electro-Communications, Tokyo, Japan

Abstract

Algorithms play a primary role in programming an orchestrated self-assembly of shapes into molecules. In this paper, we study the algorithmic self-assembly of squares by RNA co-transcriptional folding in its oritatami model. We formalize the square self-assembly problem in oritatami and propose a universal oritatami transcript made of 939 types of abstract molecules (beads) and of period 1294 that folds deterministically and co-transcriptionally at delay 3 and maximum arity into the $n \times n$ square modulo horizontal and vertical scaling factors for all sufficiently large n's after building a $\Theta(\log n)$ width "ruler" that measures n upon the seed of size $\Theta(\log n)$ on which n is encoded in binary.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Algorithmic molecular self-assembly, Co-transcriptional folding, Oritatami system, Self-assembly of squares

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.37

Supplementary Material *Software (Source Code)*: `https://github.com/rmatsuoka/oritatami_square`, archived at `swh:1:dir:016668d43c79b95a11aa013450ee472c2c7569cf`

Funding *Szilárd Zsolt Fazekas*: His research is supported by JSPS-KAKENHI Grant-in-Aid for Scientific Research (C) No. 19K11815.
Shinnosuke Seki: His research is supported in part by JSPS-KAKENHI Grant-in-Aid for Scientific Research (B) No. 20H04141 and (C) No. 20K11672.

Acknowledgements We would like to show our sincere gratitude towards Nicolas Schabanel and Ryuhei Uehara for their valuable comments in the development stage of the proposed square self-assembler. We would also like to thank the anonymous reviewers of ISAAC 2022 for their constructive comments, in particular for pointing out that mimicking the aTAM square construction's encoding of n in a base larger than 2 would not be feasible with a periodic transcript. We also want to thank Naoya Iwano and Yu Kihara for proofreading earlier drafts.

1 Introduction

RNA co-transcriptional folding (Figure 1) is a phenomenon in which the RNA single-stranded sequence (*transcript*) folds upon itself into a complex structure while being synthesized (*transcribed*) sequentially, that is, nucleotide by nucleotide, from its DNA template sequence. It plays a significant role in various computations in nature such as gene expression regulation [21] and RNA maturation [13, 18]. Geary, Rothemund, and Andersen have successfully programmed an artificial rectangular tile structure into a transcript, or more precisely into

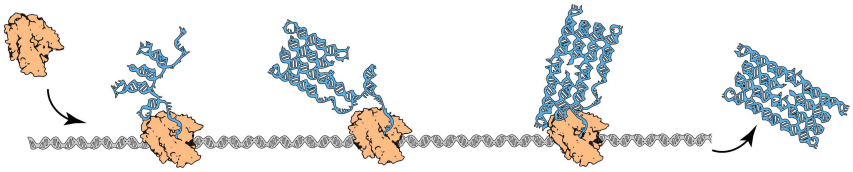


Figure 1 Co-transcriptional folding and RNA origami [8]. An RNA polymerase (colored in orange) scans a double-stranded DNA template (gray) from left to right and synthesizes the corresponding RNA sequence (blue) sequentially by mapping a DNA nucleotide read to the RNA nucleotide according to the rule `A` $\rightarrow$ `U`, `C` $\rightarrow$ `G`, `G` $\rightarrow$ `C`, and `T` $\rightarrow$ `A`. While being thus synthesized, the transcript folds upon itself into an intricate RNA structure.

its DNA template, in such a way that the transcript folds co-transcriptionally into the tile structure *in-vitro* [8]. This "RNA origami" architecture is spared the costly synthesis of RNA (it is much cheaper to synthesize DNA sequences commercially) and has been considerably automated, moduralized, and extended in size and structural diversity, for example, by the RNA Origami Automated Design Software (ROAD) [5]. Nevertheless, a programmed structure is still hardcoded in the sense that as soon as it is modified in size or shape, its transcript must be redesigned from scratch.

Algorithms and computation are fundamental in programming a transcript that folds co-transcriptionally into the shape in the class specified as input. Co-transcriptional folding has proven Turing universal in its abstract *oritatami* model [6]; therefore, any algorithm can be embedded in and guide co-transcriptional folding, though existing Turing universal oritatami systems [7, 16, 17] require much larger numbers of abstract nucleotides than 4 (`A`, `C`, `G`, `U`) as well as an intricate network of interactions among them, compared with the Watson-Crick complementarity `A`-`U` and `C`-`G`. It may however only be a matter of time before the Turing universal oritatami system is simplified enough for *in-vitro* implementation. This computational power has not been exploited yet for folding shapes co-transcriptionally. The oritatami shape builders [4, 9] still hardcode a given shape into a transcript. Masuda, Seki, and Ubukata proposed an oritatami transcript that can fold into an arbitrary finite portion of the Heighway dragon fractal H, known also as the paper folding sequence [12]. H is a sequence of L's and R's (left and right turns) and whether its i-th letter $H[i]$ is L or R can be computed by a 4-state deterministic finite automaton A, which reads the binary representation of i from its least significant bit and outputs L/R assigned to the state finally reached. The proposed transcript is periodic and its period consists of a binary counter to increment the counter value i by 1, a module to simulate A to compute $H[i]$, and a mechanical module to make a turn accordingly, all of which do their job while remembering i. It still hardcodes the bit-width k of the binary counter and cannot fold beyond $H[2^k-1]$ (it is open whether the fractal H can be folded in oritatami, c.f. [10]).

The algorithmic self-assembly of shapes by co-transcriptional folding should be studied more systematically. The assembly of arbitrary size squares is an appropriate first goal in shape assembly, due to both the ubiquity of the shape as building block for larger ones and the opportunity to relate the results to the rich record on square assembly in the DNA abstract Tile Assembly Model (aTAM) [1, 2, 3, 19], where it is considered to be a "benchmark problem" for complexity [15]. The $n \times n$ square S_n is the set of lattice points $\{(x, y) \mid 0 \leq x, y < n\}$. Unlike in the aTAM ([22]), it is challenging or even theoretically impossible to self-assemble a shape without scaling in oritatami [4, 9, 12]. Scaling S_n up by the constant c_w horizontally and by c_h vertically results in the rectangle $R_{c_w n \times c_h n} = \{(x, y) \mid 0 \leq x < c_w n, 0 \leq y < c_h n\}$ by mapping a point (i, j) in S_n to the rectangular region $\{(x, y) \mid c_w i \leq x < c_w(i + 1), c_h j \leq y < c_h(j + 1)\}$.

▶ **Definition 1** (Self-assembly of scaled-up square in oritatami). *A deterministic oritatami system* self-assembles the square at scale $c_w \times c_h$ *if for all sufficiently large* n*, starting from a seed that efficiently encodes* n*, its (unique) terminal assembly puts a bead in all rectangular regions of* $R_{c_w n \times c_h n}$*, each of which corresponds to a point in* S_n*, but no bead outside.*

The square self-assembly problem in oritatami naturally arises from Definition 1 as the problem of efficiently programming a deterministic oritatami system that self-assembles the square at as small scale factors as possible. It can be solved by using the compiler from a Turedo to a delay-3 deterministic oritatami system [16]. Turedo is a self-avoiding 2D Turing machine over the triangular grid; that is, its head cannot go back to any (hex) cell already visited (for its computational power, see also [14]). It can intrinsically simulate the square self-assembly system in the aTAM by Rothemund and Winfree [19], which consists of a constant number of tile types exclusively for counting in binary upon an $O(\log n)$-size initial assembly in order to build the $\lceil \log n \rceil \times (n - \lceil \log n \rceil)$ rectangle, for self-assembling a diagonal of S_n "cooperatively," and for filling, or even base conversion as a pre-process to let this aTAM system launch from the optimal $O(\log n / \log \log n)$-size initial assembly [1].

Nevertheless, it is worthwhile to propose an oritatami square self-assembler. Firstly, the oritatami system compiled from Turedo requires almost twice as many bead types as our proposed system (1735 compared with 939) and the period of its transcript is much longer due to the constant hidden in its asymptotic notation $\Theta(|Q|^6 \log |Q|)$, where Q is the alphabet of a simulated Turedo whereas the period of the proposed transcript is 394. The period in particular is directly translated to the size of the circular DNA template, that is, its synthesis cost. In addition, since every period folds into a regular hexagon, the resulting square would be at scale $\Theta\left(|Q|^3 \sqrt{\log |Q|}\right)$ and too bumpy to ignore in practice.

More importantly, the proposed transcript addresses the problem of how to run more than one computation in parallel upon co-transcriptional folding, which is inherently sequential. Solutions to fundamental challenges in oritatami have been almost always given as simple motifs or modules and seem not too far from solutions given by nature. For instance, a pseudoknotted linear structural motif in oritatami called a *glider* (see Figure 2) provided computations with a solid scaffold [7, 11], whereas a tandem of several pseudoknots is stacked coaxially into a long double-helix-like domain in order to support, for example, the tobacco mosaic virus [20]. Gliders can carry 1-bit in the direction of travel and this 1-bit-carrying capability was used, for instance, to wire half-adders in the oritatami binary counter, which is the first system implemented in oritatami [6] and later endowed with the capability to expand the bit-width at an overflow [11]. The counter folds its periodic transcript upon a seed that encodes an initial count in k bits in a zigzag manner so as to arrange half-adders into a 2D array with k half-adders along every zig and zag. The square self-assembler is based on the infinite binary counter and utilizes the overflow detectability rather for making a transition from Phase 1, in which it counts in binary to build a thin rectangle of height $c_h n$, to Phase 2, where it folds along this ruler into zigzags. In order to track a diagonal for halting, it displaces the half-adders along the diagonal by using a novel *elastic glider* module (Section 3). Elastic gliders are functionally upward compatible with gliders. They can shrink while carrying 1-bit. Furthermore, elastic gliders enable the square self-assembler to create and absorb offsets to propagate a 1-bit signal along the diagonal.

The system we propose does not strictly assemble a square in the sense of Definition 1: the shape assembled in Phase 2 is indeed a square satisfying the definition, but we also have the ruler shape from Phase 1 next to it. The square in Phase 2 is "thick" (as defined in aTAM square assembly [2]), that is, both its height and width are linear in n with constant ratio 16, whereas the Phase 1 rectangle results from the binary counting, hence it is thin with width logarithmic in n. In Section 5 we discuss the implications of our construction with regards to the side lengths in more detail.

We invite the interested reader to run the proposed square self-assembler on Schabanel's Simple OS Simulator available for free at:

`http://perso.ens-lyon.fr/nicolas.schabanel/OSsimulator/`

The OS specifications can be downloaded from the link below. We also included a readable listing of the rule set and the transcript in separate files.

`https://github.com/rmatsuoka/oritatami_square`

2 Preliminaries

Let Σ be a finite alphabet of types of abstract molecules, or *beads*. A bead of type $a \in \Sigma$ is called an a-bead. By Σ^* and Σ^ω, we denote the set of finite sequences of beads and that of one-way infinite sequences of beads, respectively, both including the empty sequence λ. Let $w = b_1 b_2 \cdots b_n \in \Sigma^*$ be a sequence of length n for some $n \geq 0$ and bead types $b_1, \ldots, b_n \in \Sigma$. The *length* of w is denoted by $|w|$, that is, $|w| = n$. For two indices, i, j with $1 \leq i \leq j \leq n$, we let $w[i..j]$ refer to the subsequence $b_i b_{i+1} \cdots b_{j-1} b_j$; if $i = j$, then $w[i..i]$ is simplified as $w[i]$. For $k \geq 1$, $w[1..k]$ is called a *prefix* of w.

In oritatami, a transcript, which is a directed sequence of beads, folds over the triangular grid graph $\mathbb{T} = (V, E)$ co-transcriptionally. A directed path $P = p_1 p_2 \cdots p_n$ in $\mathbb{T}$ is a sequence of *pairwise-distinct* points $p_1, p_2, \ldots, p_n \in V$ such that $\{p_i, p_{i+1}\} \in E$ for all $1 \leq i < n$. Its i-th point p_i is referred to as $P[i]$. Now we are ready to abstract RNA single-stranded structures in the name of conformation. A *conformation* C (over Σ) is a triple (P, w, H) of a directed path P in $\mathbb{T}$, a sequence of bead types $w \in \Sigma^*$ which is as long as P, and a set of (hydrogen) bonds $H \subseteq \{\{i, j\} \mid 1 \leq i, i+2 \leq j, \{P[i], P[j]\} \in E\}$. This is to be interpreted as the sequence w being folded along the path P in such a manner that its i-th bead $w[i]$ is placed at the i-th point $P[i]$ and the i-th and j-th beads are bonded (by a hydrogen-bond-based interaction) if and only if $\{i, j\} \in H$. The condition $i+2 \leq j$ represents the topological restriction that two consecutive beads along the path cannot be bonded. The energy of a conformation $C = (P, w, H)$, denoted by $\Delta G(C)$, is defined to be $-|H|$; the more bonds a conformation has, the more stable it gets. A *rule set* $R \subseteq \Sigma \times \Sigma$ is a symmetric relation over Σ, that is, for all bead types $a, b \in \Sigma$, $(a, b) \in R$ implies $(b, a) \in R$. A bond $\{i, j\} \in H$ is *valid with respect to* R, or simply R-valid, if $(w[i], w[j]) \in R$. This conformation C is *R-valid* if all of its bonds are R-valid. By $\mathcal{C}(\Sigma)$, we denote the set of all conformations over Σ; its argument Σ is omitted whenever Σ is clear from the context.

Conformations grow by an operation called elongation. Given a rule set R and an R-valid conformation $C_1 = (P, w, H)$, we say that another conformation C_2 is an elongation of C_1 by a bead $b \in \Sigma$, written as $C_1 \xrightarrow{R}_b C_2$, if $C_2 = (Pp, wb, H \cup H')$ for some point $p \in V$ not along the path P and set $H' \subseteq \{\{i, |w|+1\} \mid 1 \leq i \leq |w|, \{P[i], p\} \in E, (w[i], b) \in R\}$ of bonds formed by the b-bead; this set H' can be empty. Note that C_2 is also R-valid. This operation is recursively extended to the elongation by a finite sequence of beads as: for any conformation C, $C \xrightarrow{R}{}^*_\lambda C$; and for a finite sequence of beads $w \in \Sigma^*$ and a bead $b \in \Sigma$, a conformation C_1 is elongated to a conformation C_2 by wb, written as $C_1 \xrightarrow{R}{}^*_{wb} C_2$, if there is a conformation C' that satisfies $C_1 \xrightarrow{R}{}^*_w C'$ and $C' \xrightarrow{R}_b C_2$.

An *oritatami system* Ξ is a tuple $(\Sigma, R, \delta, \sigma, w)$, where Σ and R are defined as above, δ is a positive integer called *delay*, $w \in \Sigma^* \cup \Sigma^\omega$ is a (possibly infinite) *transcript*, and σ is an R-valid initial conformation called a *seed*, which can be considered not as a part of the system but as an input to the system. The definition usually includes a parameter called arity, which

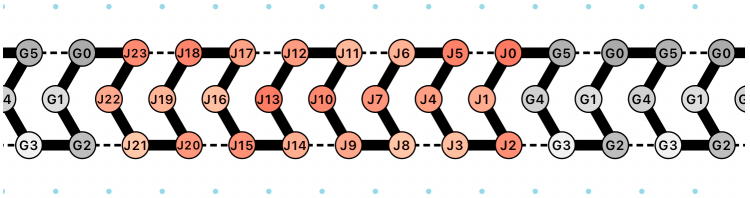


Figure 2 Glider motif in oritatami at delay 3. Here it folds co-transcriptionally from right to left.

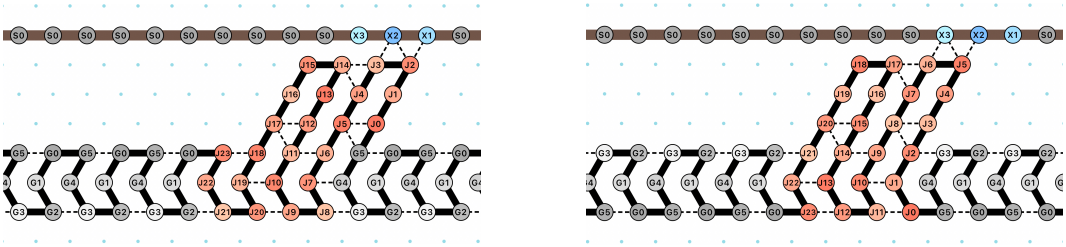


Figure 3 Elastic glider in the shrunk conformations.

is often assumed to be inexhaustible, that is, 6, which is our case as well. We opted to omit the parameter here to simplify the formalism. The transcript is to be synthesized one bead at a step. At the i-th step it has been already stabilized up to $w[i-1]$ as an elongation C_{i-1} of the seed by $w[1..i-1]$ and only the fragment $w[i..i+\delta-1]$ of the most nascent δ-beads - beyond which w has not been transcribed yet - can jiggle to minimize energy collaboratively. The set $\mathcal{F}(\Xi)$ of conformations *foldable* by the system Ξ is recursively defined as: the seed σ is in $\mathcal{F}(\Xi)$; and provided that elongation C_i of Σ by the prefix $w[1\dots i]$ be foldable (i.e., $C_0 = \sigma$), its further elongation C_{i+1} by the next bead $w[i+1]$ is foldable if

$$C_{i+1} \in \underset{C \in \mathcal{C} s.t. C_i \xrightarrow{R}_{w[i+1]} C}{\arg\min} \min \left\{ \Delta G(C') \mid C \xrightarrow{R}^{*}_{w[i+2\dots i+k]} C', k \le \delta, C' \in \mathcal{C} \right\}. \tag{1}$$

Then we say that the bead $w[i+1]$ and its bonds are *stabilized* according to C_{i+1} and they will not move or break any more. A conformation foldable by Ξ is *terminal* if none of its elongations is foldable by Ξ. The oritatami system Ξ is *deterministic* if for all $i \ge 0$, there exists at most one C_{i+1} that satisfies (1). A deterministic oritatami system folds into a unique terminal conformation. All the systems presented in this paper are deterministic and we set $\delta = 3$.

In the next section, we illustrate the dynamics (1) by introducing a novel oritatami motif of practical use called an *elastic glider*.

3 Elastic Glider

The square self-assembler which we shall describe in Section 4 has to remember where the diagonal is while counting in binary. As done in the existing oritatami binary counters [6, 11], its periodic transcript folds in a zigzag manner into a 2D array of half-adders such that in each zig (←) the carry-out of a half-adder is wired to the carry-in of the half-adder transcribed next by a self-standing motif called a *glider*, which folds at delay 3 as shown in Figure 2 in the direction indicated by its arrow-like shape (leftward in the figure, for instance). The capability of gliders to carry 1 bit is thus already used up and transferring more bits in the direction of folding is yet to be achieved in the theory of oritatami systems.

The square self-assembler overcomes this limit by displacing all and only half-adders on the diagonal together with some preceding and succeeding modules using a novel *elastic glider* module, colored in orange in Figures 2 and 3. The elastic glider also works at delay 3 and is upward compatible with the standard glider in the sense that besides the conventional "stretched-out" glider conformation (Figure 2), it can shrink in the two ways shown in Figure 3 below the tandem of beads `X3-X2-X1`.

Let us see how an elastic glider shrinks as a demonstration of oritatami dynamics (1) at $\delta = 3$. Initially, the nascent fragment of its first δ beads, `J0-J1-J2`, is attracted by `G3` but pulled upward more strongly by `X2-X1` with two bonds (Fig. 3, left); in fact, folding this fragment in any other way results in at most one bond. As a result, `J0` is stabilized to the northeast of `G5`. These two bonds between `J2` and `X2-X1` also stabilize `J1` and `J2` as shown in Figure 3 (left). The bond `J3-X2` guides the transcript leftward and the fragment `J3-J4-J5` folds back along `J0-J1-J2` just stabilized due to the original glider rule (`J5`, `J0`) as well as a new rule (`J5`, `G5`), which does no harm in the usual glider conformation as when `J5` is transcribed after `J2` is stabilized, `G5` is "too far." The fragment `J6-J7-J8` could fold back along `J3-J4-J5` just stabilized but rather folds downward along `G3-G4-G5` due to the new rules (`J7`, `G4`) and (`J8`, `G3`); observe that beads paired here are too far from each other to bind in the glider conformation. The succeeding `J9-J10-J11` folds as in the usual glider. This is the functional unit of this module and it can shorten the glider by 2. Arbitrarily many, say k, functional units can be repeated and the resulting elastic glider, of $12k$ beads length, yields the offset $2k$ (the case of $k = 2$ is illustrated in Figures 2 and 3); let us call this the *elastic glider for offset* $2k$. Note that the second and subsequent units do not need to be suspended from the ceiling for shrinking because they can rather rely on their shrunk predecessor as `J14` binds to `J3-J4`. With the preceding glider being vertically flipped (`G5` at the bottom), the first unit `J0-J1-`$\cdots$`-J11` folds rather as shown in Figure 3 (right). Compare Figure 3 (left) with (right) and observe that they yield the same offset and the last bead of every unit is at the same height as the preceding `G5`; this means that this module can shrink independently of whether 0 or 1 is propagated.

Offsets can be not only created but absorbed by elastic gliders. Observe in Figure 3 how critical it is to place the tandem `X3-X2-X1` precisely as illustrated relative to the preceding glider. Starting too early (right) or too late (left), this elastic glider would stretch out to the normal glider. This property enables the square self-assembler as shown in Figure 8 to shift half-adders considerably (by 6 points) by sandwiching them by elastic gliders *Pre*, *Post*, D, and *spc* as well as to have this D push the succeeding transcript up to the next *Pre* slightly (by 2 points) to propagate a signal to track the diagonal.

4 Algorithmic Oritatami Square Self-Assembler

The infinite binary counter [11] serves as a basis of the proposed square self-assembler. The transcript of the counter is made of four glider-based modules F (formatter), L (left carriage return: CR), H (half-adder), and R (right CR) which are transcribed in this order periodically infinitely many times; that is, it can be represented as $(FLHR)^*$. All the modules can fold into a glider of even length, which puts the first and last beads of their transcript on the same side (top or bottom; see `J0` and `J23` in Figure 2). We shall explain shortly how this property enables gliders to wire half-adders. Below a seed on which the count is initialized, the transcript folds into zigs ($\leftarrow$) and zags ($\rightarrow$) as:

Figure 4 (Left) The glider-like conformation `Rg` that R takes during zigs and zags; (Right) The right-CR conformation of R `Rcr`.

$$
\begin{array}{llllll}
 & & \text{Seed} & & & \\
\hline
 & L\circlearrowleft & FRHLFRH\leftarrow & \cdots & \leftarrow LFRHLF & \circlearrowright \\
 & & HRFLHRF\rightarrow & \cdots & \rightarrow LHRFLH & \\
 & L\circlearrowleft & FRHLFRH\leftarrow & \cdots & \leftarrow LFRHLF & \circlearrowright R \\
 & & HRFLHRF\rightarrow & \cdots & \rightarrow LHRFLH & \\
L\circlearrowleft & FRH\ L\leftarrow & FRHLFRH\leftarrow & \cdots & \leftarrow LFRHLF & \circlearrowright R
\end{array}
\tag{2}
$$

Each period $HRFL$ folds into a glider-like linear conformation (see Figure 4 for such a conformations that an instance of corresponding right-CR module R takes in the proposed square self-assembler) except at the left end of the counter where an instance of L takes a special CR conformation $\circlearrowleft$ to guide the transcript into the next zag or at the right end where an instance of R does a carriage-return alike. The instance of L behaves thus differently there due to a special configuration of beads exposed below the instance of L above, which has also folded into the left CR conformation. This is also the case for R; the corresponding module of the same name in the proposed square self-assembler admits the two conformations in Figure 4. Observe that a period $HRFL$ in a zag is aligned vertically with another period in the preceding zig so as for their R's are one above the other; see those colored in blue and red in (2). These periods collaboratively function as one half-adder in a way explained shortly so that we collectively call them a *functional unit*. The binary counter thus co-transcriptionally arranges functional units into a 2D array.

Among the two instances of H and those of F in a functional unit, that of H in a zig and that of F below actually play a functional role whereas the other H and F are idled; let us call these active ones its *zig* H and *zag* F, respectively (we shall refer to other modules, say X, of a functional unit of the square self-assembler as the *zig* X or *zag* X). The zig H actually serves as a half-adder. It receives a 1-bit carry-in c_{in} propagated through the zig from the previous H and another 1-bit b from the zag F of the functional unit above, and carries out $c_{\text{in}} \wedge b$ to the next H and outputs c_{in} `xor` b to the zag F below, which formats the output for the sake of the zig H of the functional unit which will be transcribed later below. Instance of the corresponding formatting module F_H of the square self-assembler take one of the two conformations `FH0` and `FH1` in Figure 6 in order to format the outputs 0 and 1 from above, respectively, in a zag. In a zig, they rather fold into the glider-like conformation `FHg` in Figure 7. Other modules also fold into their own glider-like conformation in a zig so that a

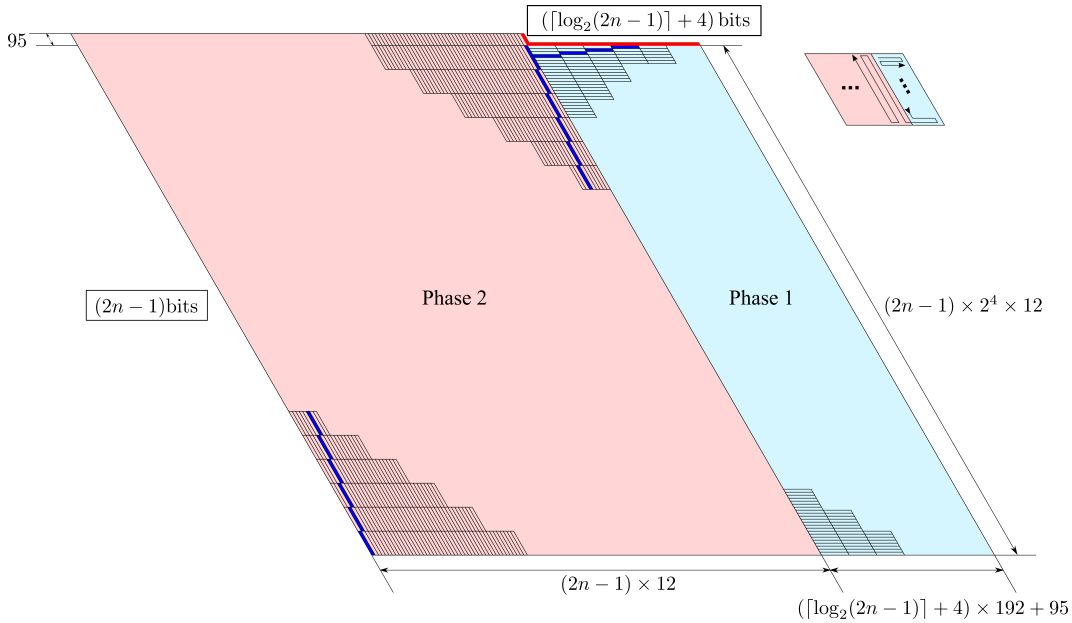


Figure 5 Propagation of a diagonal signal through the 2D array of $(\lceil \log(2n-1) \rceil + 4) \times 16(2n-1)$ functional units in Phase 1 and the 2D array of $(2n-1) \times (2n-1)$ functional units in Phase 2. Thin rectangles of size 192×12 in Phase 1 or 12×192 in Phase 2 are a functional unit and the signal propagates through those colored in dark blue. The seed is colored in red.

carry propagates from the zig H's of a functional unit to that of the next functional unit as a position of the last bead of each module (top for no-carry, bottom for carry). Thus, at an overflow, a zig ends at the bottom so that the CR signal is too far for L being transcribed to do a carriage-return; this instance of L and the succeeding H, R, and F, all fold into gliders to expand the counter bit-width by 1. The square self-assembler does not need bit-width expansion. As overviewed in Figure 5, it rather exploits this overflow detectability rather to have the instance of L guide the transcript from Phase 1, in which a tall rectangle of height $c_h n$ has self-assembled upon an $O(\log n)$-size seed (see Figure 14) by thus counting in binary, to Phase 2, where the zigzag counting continues after the count is reset to 0 along the left long side of the rectangle, by taking the 90-degree right-turn conformation; see the bottom-right instance of L, colored in teal, in Figure 10. Since the overflow till the next right CR, the transcript proceeds along the left CRs northwestward, which have all the instances

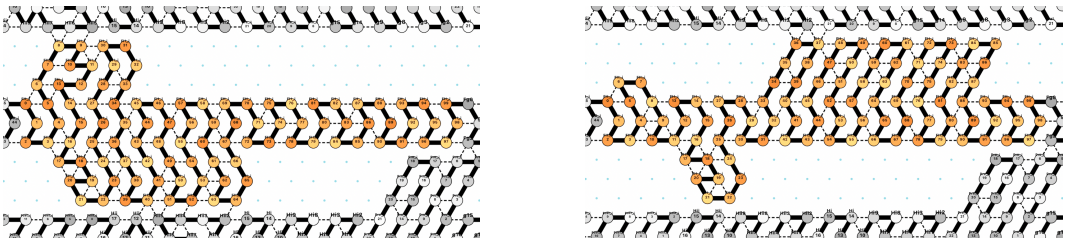


Figure 6 The two conformations `FH0` and `FH1` that an instance of the module F_H can take in a zag. Until the first right CR after the phase transition, all the instances of F_H take `FH0`; in this way, the binary counter is reset to 0 and thus guarantees that it will not be overflown before the square is completed.

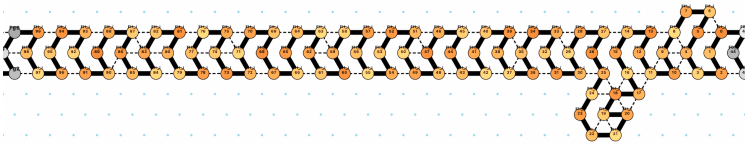


Figure 7 The glider-like conformation `FHg` into which all the instances of F_H folds in a zig.

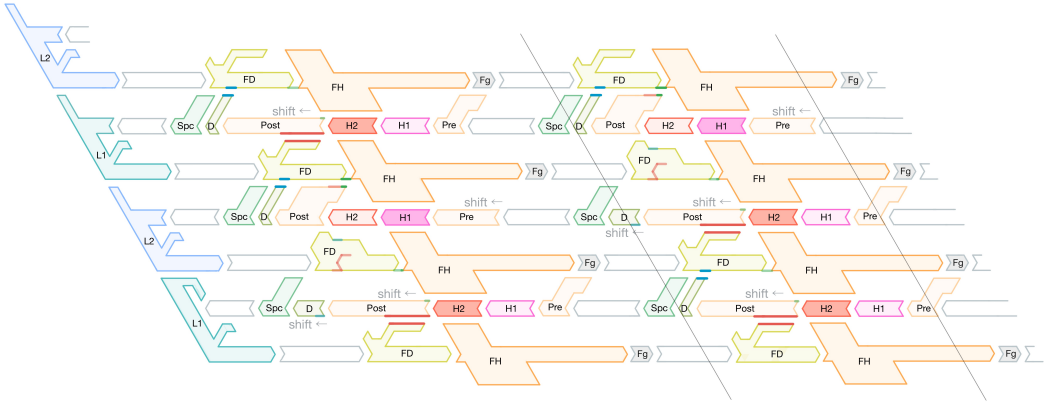


Figure 8 Diagonal signal propagation by elastic gliders *Pre*, *Post*, *D*, and *Spc*. Only along the diagonal, *Pre* is stretched and shrinks the succeeding *Post*. The shrunk *Post* folds F_D below differently, and stretches *D* further below to push the next *Pre* forward so that it cannot shrink.

of F_H take `FH0`; this means that the binary counter is reset to 0 at the beginning of Phase 2 and guarantees that it does not overflow before the diagonal reaches the opposite corner, where the system is supposed to halt.

In order to halt, the self-assembler tracks the left-down diagonal ($\swarrow$) of the square by displacing a functional unit on the diagonal leftward by 2 points and by having this shift be propagated diagonally to displace the next functional unit on the diagonal alike (see Figure 8). Reaching the left end of the counter, this displacement folds an instance of L into a special left-CR conformation shown in Figure 9 (Right), which will remind the transcript in Phase 2 of where the diagonal is; compare the conformations of the yellow module, F_D, to the left of L in Figure 9, depending on whether it is (Left) away from the diagonal or

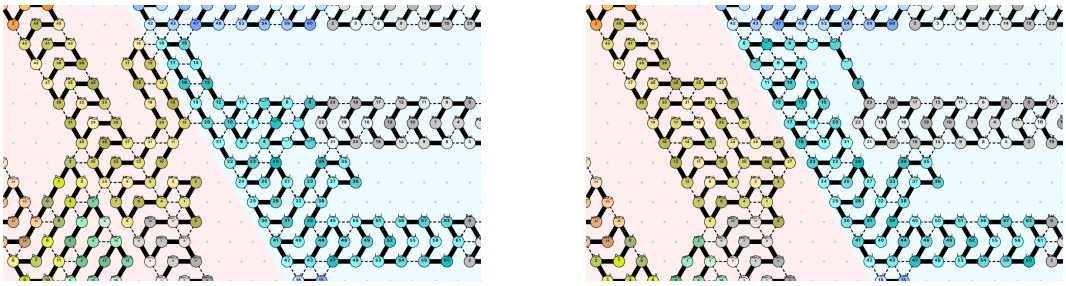


Figure 9 The two carriage-return (CR) conformations of L_1 (colored in teal). Below an instance of L_2 (blue) in a CR conformation, an instance of L_1 does a carriage-return in the way shown left unless the preceding glider is pushed leftward (by the diagonal signal), where it does a carriage-return as shown right in order to remind an instance of F_D (yellow), if transcribed to its left, of where the diagonal is. An instance of L_2 always takes the left conformation for carriage-return.

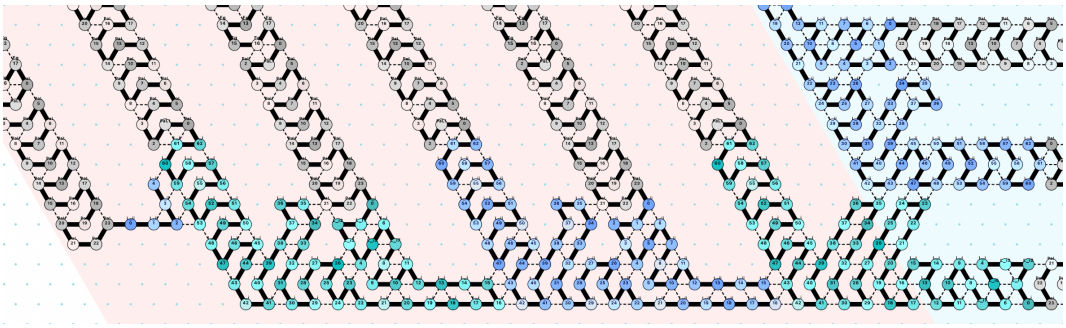


Figure 10 Phase transition by L_1 (colored in teal) and halting by L_2 (blue). The bottom right instance of L_1 starts folding at the bottom, that is, with a carry due to the counter overflow. In this environment, the instance of L_1 takes the phase transition conformation `Lpt`. Afterward, the system keeps counting in binary in Phase 2 until the diagonal signal reaches the bottom edge and pushes an instance of L_2 transcribed there further downward to trap the transcript for halting.

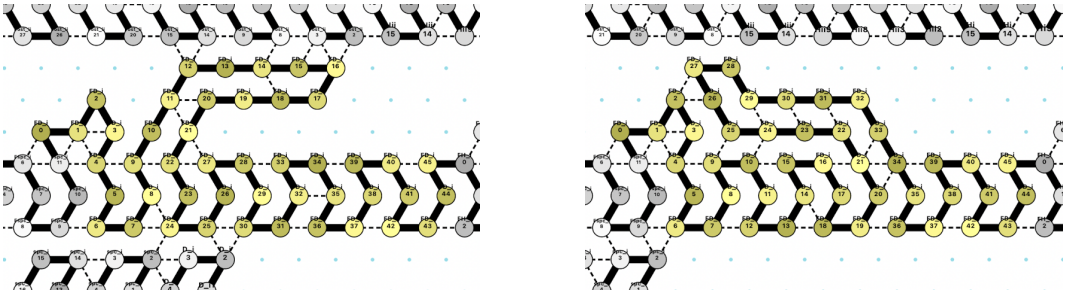


Figure 11 Two conformations `FDf` and `FDt` that F_D takes in a zag when being (Left) away from the diagonal and (Right) on the diagonal, respectively. In a zig, F_D always folds into `FDt` or upside-down.

(Right) on it (see also Figure 11). Afterward, the diagonal signal propagates in Phase 2 until it reaches the left end (at the bottom in figures), where an instance of L traps the transcript inside the region enclosed by the previous zag and zig to halt the system.

As described so far, L must play the two extra functional roles at the left edge of the counter in this square self-assembler: of letting the diagonal signal pass through the phase boundary and of stopping it at the end of Phase 2. It is highly non-trivial to propagate a diagonal signal differently in Phase 1 and in Phase 2, and hence, we could not help but rather introduce two types of L. We indeed replace every other instance of L along the transcript with its "doppelgänger," which is implemented like L but from bead types not used in L. The period is thus doubled, and one period now contributes to two functional units. Observe in (2) that every zig or zag involves an even number of L's as suggested in (2). Therefore,

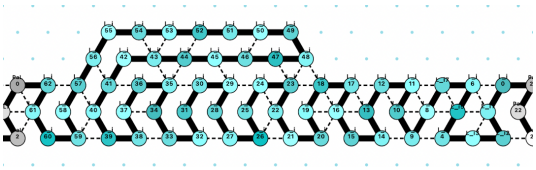


Figure 12 The glider-like conformation `Lg` that L_1 and L_2 always fold into in the middle of zigs and zags.

Table 1 The half-period of the transcript of the square self-assembler at modular level.

	Spacer
Pre:	Elastic glider for offset 4
H_1, H_2:	Half-adder and its doppelgänger (Figure 13)
Post:	Elastic glider for offset 6
D:	Diagonal tracker (Elastic glider for offset 2)
Spc:	Elastic glider for offset 4
	Spacer
R:	Right turner (Figure 4)
	Spacer
F_D:	Formatter for D (Figure 11)
F_H:	Formatter for H (Figures 6 and 7)
F_g:	(Conventional) glider
	Spacer
L_1 or L_2:	Left turner and its doppelgänger (Figures 9, 10, and 12)

along the left end of the counter thus modified, the two types of left-turners, say L_1 and L_2 (colored in teal and in blue, respectively), occur alternately one below another in the CR conformation. Two instances of L_1 still get one above the other inside the counter and so do those of L_2, where introducing an "intra-modular" rule to fold an instance of L_1 in some specific manner may cause an "inter-modular" interference and vice versa. Since this problem cannot be solved no matter how many types of doppelgängers are available, we rather keep a zig away from both the zags above and below by 3 points. Note that both L_1 and L_2 fold into the glider-like conformation in Figure 12 in the middle of zigs and zags.

Based on this modified counter, we now complete the design of the square self-assembler. Its transcript is obtained by further inserting elastic gliders for diagonal signal propagation, which shall be described in detail in Section 4.1, as well as spacers into the half-period as described in Table 1. The spacers, which always fold into a glider, make sure that the ratio of the height of one zigzag, which is 12 due to the space between zigs and zags, to the length of the linear structure into which the half-period folds is a power of 2; here 16 is the smallest possible ratio to accommodate all the functional modules. As a result, each of the two half-periods of a functional unit folds into a glider-like linear structure of width $12 \times 16 = 192$ and of height 6 except at the left and right end of the counter. Though these two structures are not perfectly aligned vertically, it is more convenient for our argument to assume that a functional unit folds into a rectangle of width 192 and height 12 as abstracted in Figure 5. Indeed, this assumption does not impair any mathematical rigor because it is just up to us which bead to designate as an origin of a half-period.

4.1 Diagonal signal propagation for halt

Figure 8 illustrates how instances of the modules *Pre*, H_1, H_2, *Post*, D, F_D, and F_g collaborate for propagating a diagonal signal from top right to bottom left through Phases 1 and 2.

In a zig, the half-period of the functional unit along the diagonal is pushed 2 points leftward by the previous half-period, or more precisely, by its zig D. Its zig *Pre* thus gets too far from an instance of F_g above to be suspended for shrinking. It hence rather stretches out and amplifies the offset from 2 to 6. The resulting offset 6 pushes H_1H_2 leftward but H_1 is interlocked to the counter circuit in substitution for H_2. The succeeding instance of *Post* fully absorbs the offset by shrinking. The glider conformation of *Post* is provided with a

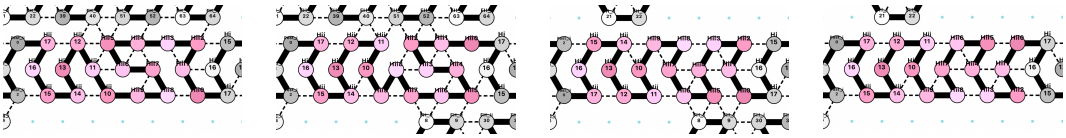


Figure 13 The four conformations of H_1 and of H_2: from left, `H0+c`, `H0+n`, `H1+c`, and `H1+n`. A carry-in c_{in} is given as whether the previous glider ends at the top (no-carry) or bottom (carry) whereas the input 1-bit b is fed from above as whether an instance of F_H just above ($b = 0$) or far above ($b = 1$).

sequence of bead types along both sides that attracts an instance of F_D below in the next zag so as for the instance to take the conformation `FDf` as shown in Figure 11 (Left). Both of these "signals" are however convoluted inside when an instance of *Post* shrinks and let an instance of F_D below fold freely into `FDt` shown in Figure 11 (Right). The zig D of the functional unit below stretches out if this zag F_D takes `FDt`, or it shrinks otherwise. These conformations result in the offset 2 and it is propagated leftward through the zig up to the next *Pre*, which is the zig *Pre* of the next functional unit on the diagonal. The signal has successfully propagated along the diagonal. Reaching the left end of the counter instead and moreover an instance of L_1 is being transcribed, the offset 2 has the instance of L_1 fold into `Lcrd`, one of the two possible CR conformations of L_1, as shown in Figure 9. If an instance of L_2 is being transcribed there instead, the offset guides the transcript inside the enclosed region to halt the system, as shown in Figure 10.

Note that one half-period is provided with two half-adder modules H_1 and H_2, which are a doppelgänger of each other, made of disjoint bead type sets. Both of them are indispensable as long as the diagonal signal is propagated as just described due to the slide of the whole half-period. Without any offset, H_2 interlocks with the instance of F_H above and H_1 is idle. The offset 2 must be amplified to let H_1 interlock instead because half-adder modules have not been implemented so compactly yet [6, 11, 12]. These half-adder modules are implemented so as to admit the four conformations of width 6 shown in Figure 13. The offset 2 is hence amplified to 6 by *Pre*. If we implemented D as an elastic glider with offset 6, then *Pre* would be unnecessary. However, L_1 cannot handle offset 6, so we amplify the offset temporarily by *Pre* and then absorb it by *Post*, so the offset 6 never reaches L_1.

Last but not the least, the module *Spc*, which is an elastic glider for offset 4, makes sure that all the zigs and zags are exactly the same length even with all these expansion and contraction mechanisms by shrinking in a zig and expanding in a zag. It turned out that for this purpose it suffices for its instances to always shrink in a zig and stretch out into the glider in a zag. This module guarantees more strongly that all the half-periods except those in a zig along the diagonal or the sub-diagonal fold the same in width; those in a zig along the sub-diagonal are longer than usual by 2 but this is immediately cancelled out by the succeeding half-period, which is on the diagonal and shorter by 2.

4.2 Seed, Phase Boundary, and Scaling

One functional unit is of width 12 and of height 12×2^4 in Phase 2 so that it spans 16 zigzags, that is, functional units in Phase 1. See Figure 5. We set the initial value of the counter as $c_0 = \left(2^{\lceil \log(2n-1) \rceil} - (2n-1)\right) \times 2^4$ and encode its binary representation $b_{k-1}b_{k-2}\cdots b_1 b_0$ with $k = \lceil \log(2n-1) \rceil + 4$ and $b_0, b_1, \ldots, b_{k-1} \in \{0, 1\}$ upon the seed as this value is encoded below the zag; see Figure 14. Starting from this seed, the transcript folds into $(2n-1) \times 2^4$ zigzags until the counter overflows (we shall explain why not n but $2n-1$ shortly). Recall

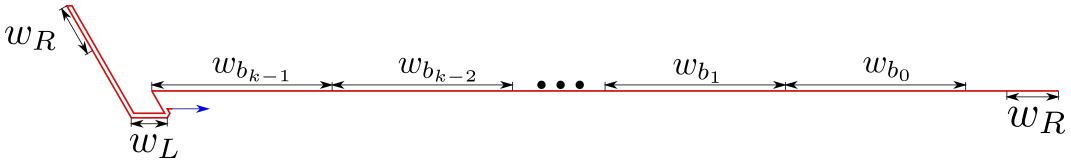


Figure 14 Encoding of an integer represented in binary as $b_{k-1}b_{k-2}\cdots b_1b_0$ upon the seed. w_L and w_R are the configurations of beads exposed at the bottom of the left CR configuration(s) of L_1 and that of R, respectively.

that in its CR conformation, L_1 lets the diagonal signal pass through while L_2 stops it. Thus, the diagonal signal must be launched properly so as to hit an instance of L_1 at the left edge of the counter in Phase 1. More restrictively, it must hit the 4th CR from above of the 16 zigzags that is spanned by a functional unit of Phase 2 so that the zag F_D can read the signal. Thus, the very first left-CR of the counting must be done by an instance of L_2. The final zig in Phase 1 ends with an instance of L_1, which detects the overflow and takes the phase transition conformation `Lpt` as shown in Figure 10. In Phase 2, the diagonal signal must hit the left CR of L_2 to halt. Since phase transition has been made by L_1, the number of zigzags in Phase 2 must be odd. To make sure that happens, we need an even number of functional units below the one which starts the diagonal signal of Phase 2. Therefore, we bundle every 32 zigzags in Phase 1 from the bottom, map the corresponding 2 functional units (one period) in Phase 2 to one lattice point in the $n \times n$ square S_n, and have the diagonal signal hit the 20th CR from above of such a bundle. We bundle the first 16 actual zigzags with 16 imaginary zigzags represented by the seed and hence the diagonal signal hits to its "20th" CR (actually 4th) by launching the signal from the 4th most significant bit of c_0 upon the seed.

We could, in fact, start the diagonal signal of Phase 2 directly from the seed making the construction perhaps somewhat simpler by avoiding signal passing between the phases. However, this would not preclude having to combine the binary counting with diagonal signal propagation as we only have one transcript which has to take care of both phases. Moreover, the present construction is "ready" for generalization: if one can implement a slowdown of the diagonal signal, they then can start it from the top-right corner of Phase 1 and have the whole assembly conform to Definition 1.

5 Conclusions

As explained earlier, the size of one functional unit is 192×12 and this means that the difference between the width and height of the Phase 2 rectangle is linear in n with a factor $2 \cdot (192 - 12)$. By this it is clear that Phase 1 cannot make up for the difference if we want to maintain the efficiency of having a $o(n)$ sized seed in our current construction.

The rectangle that the proposed system folds into is too thick to be self-assemblable by merely counting in binary, unless we trade seed size for scale and program size complexity by simply using a binary counter with a linear-sized seed and a suitably implemented halting module, which terminates on overflow. However, it is not thick enough yet even with Phase 2, as its aspect width-to-height ratio is asymptotically 16. The periods have been lengthened by spacers in order to make the aspect ratio of the rectangle be a power of 2, so that we can sync the phase bottoms. It should be constructive and resource-friendly to utilize the idle lot, where spacers are, for example, by embedding a yet-to-be-implemented mechanism to decelerate the diagonal signal exponentially, which would allow a Phase 1 like construction

throughout. Even if one could find a module which slows the diagonal signal by a fixed amount, inserting multiple copies of it might be enough to sync the vertical and horizontal propagation speeds, which would make the Phase 2 rectangle closer to, or exactly a square.

Finally, perhaps the proposed system could be programmed more efficiently by encoding n in a larger base than 2 and implementing a mechanism to decode it into its binary representation, as done for optimal program-size self-assembly of squares in aTAM [1]. The aTAM method uses tiles tailored for each n, so one would need a transcript with a prefix which decodes the large base encoding of n and a succeeding periodic part which assembles the square as proposed here. Therefore, instead of a purely periodic transcript, the optimal system would require an ultimately periodic one. At this moment it is not clear whether such a transcript is feasible.

References

1 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001)*, pages 740–748. ACM, 2001.

2 Gagan Aggarwal, Qi Cheng, Michael H. Goldwasser, Ming-Yang Kao, Pablo Moisset de Espanes, and Robert T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005.

3 Florent Becker, Eric Rémila, and Nicolas Schabanel. Time optimal self-assembly for 2D and 3D shapes: The case of squares and cubes. In *Proceedings of the 14th International Meeting on DNA Computing (DNA 14)*, volume 5347 of *LNCS*, pages 144–155. Springer, 2008.

4 Erik D. Demaine, Jacob Hendricks, Meagan Olsen, Matthew J. Patitz, Trent A. Rogers, Nicolas Schabanel, Shinnosuke Seki, and Hadley Thomas. Know when to fold 'em: Self-assembly of shapes by folding in oritatami. In *Proceedings of the 24th International Conference on DNA Computing and Molecular Programming (DNA 24)*, volume 11145 of *LNCS*, pages 19–36. Springer, 2018.

5 Cody Geary, Guido Grossi, Ewan K. S. McRae, Paul W. K. Rothemund, and Ebbe S. Andersen. RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds. *Nature Chemistry*, 13:549–558, 2021.

6 Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Programming biomolecules that fold greedily during transcription. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *LIPIcs*, pages 43:1–43:14, 2016.

7 Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Proving the turing universality of oritatami co-transcriptional folding. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *LIPIcs*, pages 23:1–23:13, 2018.

8 Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA structures. *Science*, 345(6198):799–804, 2014.

9 Yo-Sub Han and Hwee Kim. Construction of geometric structure by oritatami system. In *Proceedings of the 24th International Conference on DNA Computing and Molecular Programming (DNA 24)*, volume 11145 of *LNCS*, pages 173–188, 2018.

10 Yo-Sub Han and Hwee Kim. Impossibility of strict assembly of infinite fractals by oritatami. *Natural Computing*, 20(4):691–701, 2021.

11 Kohei Maruyama and Shinnosuke Seki. Counting infinitely by oritatami co-transcriptional folding. *Natural Computing*, 20(2):329–340, 2021.

12 Yusei Masuda, Shinnosuke Seki, and Yuki Ubukata. Towards the algorithmic molecular self-assembly of fractals by cotranscriptional folding. In *Proceedings of the 23rd International Conference on Implementation and Application of Automata (CIAA 2018)*, volume 10977 of *LNCS*, pages 261–273, 2018.

13 Evan C. Merkhofer, Peter Hu, and Tracy L. Johnson. Introduction to cotranscriptional RNA folding. In *Methods in Moleclar Biology*, volume 1126, pages 83–96. Springer, 2014.

14 Samuel Nalin and Guillaume Theyssier. On turedo hierarchies and intrinsic universality. In *Proceedings of the 28th International Conference on DNA Computing and Molecular Programming (DNA 28)*, volume 238 of *LIPIcs*, pages 6:1–6:18, 2022.

15 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

16 Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki, and Guillaume Theyssier. Oritatami systems assemble shapes no less complex than tile assembly model (aTAM). In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *LIPIcs*, pages 51:1–51:23, 2022.

17 Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki, and Yuki Ubukata. Simple intrinsic simulation of cellular automata in oritatami molecular folding model. In *Proceedings of the 14th Latin American Symposium on Theoretical Informaitcs (LATIN 2020)*, volume 12118 of *LNCS*, pages 425–436, 2020.

18 Roberto Perales and David Bentley. "Co-transcriptionality" – the transcription elongation complex as a nexus for nuclear transactions. *Molecular Cell*, 36(2):178–191, 2009.

19 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstracts). In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000)*, pages 459–468. ACM, 2000.

20 Alex van Belkum, Jan Pieter Abrahams, Gornelis W. A. Pleij, and Leender Bosch. Five pseudoknots are present at the 204 nucleotides long 3' noncoding region of tobacco mosaic virus RNA. *Nucleic Acids Research*, 13(1):7673–7686, 1985.

21 Kyle E. Watters, Eric J. Strobel, Angela M. Yu, John T. Lis, and Julius B. Lucks. Cotranscriptional folding of a riboswitch at nucleotide resolution. *Nature Structural and Molecular Biology*, 23(12):1124–1131, 2016.

22 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

On Constrained Intersection Representations of Graphs and Digraphs

Ferdinando Cicalese
Department of Computer Science, University of Verona, Italy

Clément Dallard
Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, Orléans, France

Martin Milanič
FAMNIT and IAM, University of Primorska, Koper, Slovenia

Abstract

We study the problem of determining minimal directed intersection representations of DAGs in a model introduced by [Kostochka, Liu, Machado, and Milenkovic, ISIT2019]: vertices are assigned color sets, two vertices are connected by an arc if and only if they share at least one color and the tail vertex has a strictly smaller color set than the head, and the goal is to minimize the total number of colors. We show that the problem is polynomially solvable in the class of triangle-free and Hamiltonian DAGs and also disclose the relationship of this problem with several other models of intersection representations of graphs and digraphs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph theory

Keywords and phrases Directed intersection representation, intersection number

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.38

Funding This work is supported in part by the Slovenian Research Agency (I0-0035, research program P1-0285, research projects J1-3001, J1-3002, J1-3003, J1-4008, J1-9110, N1-0102, and N1-0160); and the GNCS group of the Italian "Istituto Nazionale di Alta Matematica - INdAM".

Acknowledgements We would like to thank Andrea Caucchiolo for several insightful discussions we had on some of the results contained in this paper.

1 Introduction

Problem definition and state of the art. Given a digraph $D = (V, A)$, a *directed intersection representation* of D is a pair (U, φ) where U is a finite set of *colors* and φ is a *proper coloring* of D, that is, a mapping assigning to each vertex $v \in V$ a set $\varphi(v) \subseteq U$ such that for any two vertices $u, v \in V$, it holds that

$$(u, v) \in A \quad \text{if and only if} \quad \varphi(u) \cap \varphi(v) \neq \emptyset \text{ and } |\varphi(u)| < |\varphi(v)|\,. \tag{1}$$

The *cardinality* of a directed intersection representation (U, φ) of D is defined as the number of colors, that is, $|U|$. Note that if (U, φ) is a directed intersection representation of a digraph D and $W = (v_1, \ldots, v_k)$ is a walk in D, then $|\varphi(v_1)| < \ldots < |\varphi(v_k)|$, which implies that D is acyclic (that is, a DAG). On the other hand, Kostochka et al. [9] showed that every DAG admits a directed intersection representation. They initiated a study of the following invariant of DAGs. The *directed intersection number* of a DAG $D = (V, A)$ – denoted by $DIN(D)$ – is the smallest cardinality of a directed intersection representation of D. In [9, 11], the authors focused on characterizing the extremal values of DIN. They showed that:

- for every DAG D with n vertices, it holds that $DIN(D) \leq \frac{5n^2}{8} + \mathcal{O}(n)$.[1]
- for every n there is a DAG D with n vertices such that $DIN(D) \geq \frac{9n^2}{16}(1 - o(1))$.

In [1, 2] Caucchiolo and Cicalese studied the computational complexity of determining $DIN(D)$. They showed that the problem of computing $DIN(D)$ is NP-hard even when D is an arborescence (a tree with all the edges oriented away from the root). Moreover, for general DAGs and any $\epsilon > 0$, the problem does not admit an $n^{1-\epsilon}$ approximation unless $\mathsf{P} = \mathsf{NP}$. Conversely, for the case of arborescences the problem is shown to be in APX, and in [2] the authors also provide an asymptotic fully polynomial time approximation scheme.

The main result. In this paper we continue the quest for islands of tractability in the complexity landscape for the problem of computing directed intersection representations of DAGs initiated in [1, 2]. We focus on a class of graphs that, in the analysis of [9], appears to include the instances that are the most demanding in terms of the *DIN* value. In fact, in [9], a key point in the construction of the lower bound on the extremal value of $DIN(D)$ is to consider DAGs that are both Hamiltonian and triangle-free.[2] Here we show that for every D in the class of Hamiltonian and triangle-free DAGs, computing the value of $DIN(D)$ is a tractable problem solvable in time $\mathcal{O}(n^3)$.

A key element for obtaining such a result is the fact that for any Hamiltonian and triangle-free DAG D it is possible to define a demand function b on the vertices such that the value of $DIN(D)$ can be exactly characterized in terms of the value of a maximum b-matching in the underlying graph of D, a parameter that can be computed in polynomial time [8, 14].

Additional results, related problems, and literature. The proof of the above result leads us to introduce several generalizations and variants of intersection representations of graphs and digraphs (defined in Section 2.1). We believe that these variants might be of independent interest and for which we are able to prove interesting results on their relations in terms of minimal intersection representations (summarized in Figure 1).

It is known that any finite undirected graph G admits an intersection representation given by a family of finite sets associated to its vertices, such that two vertices are adjacent if and only if their associated sets intersect. The minimum cardinality of the ground set of such a family is referred to as the *intersection number* of the graph G and denoted by $IN(G)$. Erdős, Goodman and Pósa [6] showed that $IN(G)$ equals the minimum number of cliques needed to cover the edges of G, i.e., the size of a minimum edge clique cover of G. Determining this value was proved to be NP-hard in [13] (see also [10]). By [7, 12], both problems are not approximable within a factor of $|V|^{\epsilon}$ for some $\epsilon > 0$ unless $\mathsf{P} = \mathsf{NP}$. Applications of intersection representations and clique covers are found in areas as diverse as computational geometry, matrix factorization, compiler optimization, applied statistics, resource allocations, etc; see, e.g., the survey papers [15, 16, 17], and the comprehensive introduction of [3].

In this paper several new variants are considered which contribute to this rich literature and in particular to the approach of [17] of studying constrained versions of intersection representation and their applications.

The following practical scenario can be modelled by a constrained variant of the intersection number of undirected graphs, which is considered in the series of reductions leading to the proof of our main result.

---

[1] The precise bound given in [11] is $\frac{5n^2}{8} - \frac{3n}{4} + 1$.

[2] Liu et al. [11] leave as an open problem to show that for every n the maximum value of $DIN(D)$ among the DAGs with n vertices is attained by one that is also Hamiltonian.

There is a shared resource (for example, a wireless communication channel) and a set of participants who want to use the resource. However, no two participants are willing to share the resource at the same time unless they need to do it for accomplishing a common task. In the wireless communication example, you might imagine that using the channel at the same time means to be possibly eavesdropped while you do need to share temporally at least once with whomever you want to communicate with. We say that two participants are *compatible* (with each other) if they need to accomplish a common task. We assume that the resource can be used for an arbitrary number of time periods, where in each time period only a subset of pairwise compatible participants can share the resource. Furthermore, if a set S of pairwise compatible participants shares the resource in a certain time period, then the common task of any pair of participants in S can be carried out in this period. Since the use of the resource is expensive, our goal is to design a schedule of assigning the participants to time slots for using the resource that minimizes the total number of temporal slots, such that all pairs of participants can accomplish their tasks without ever using a resource together with an incompatible participant. This problem can be modeled as the problem of computing the intersection number of the compatibility graph. If we also assume that every participant requests to take part in at least a certain number of slots, we obtain the ℓ-constrained intersection number, where $\ell(v)$ is the desired lower bound on the number of slots for participant v (see Section 2.1 for definitions).

2 Notations and definitions

We denote by $\mathbb{N}$ the set of all positive integers and by $\mathbb{Z}_+$ the set of all nonnegative integers. All graphs in this paper will be finite and simple, but may be directed or undirected. We will use the term *graph* to refer to an undirected graph and the term *digraph* to refer to a directed graph.

Definitions for graphs. A *graph* is a pair $G = (V, E)$ where $V = V(G)$ is a finite set of *vertices* and $E = E(G)$ is a set of 2-element subsets of V called *edges*. Two vertices u and v in a graph $G = (V, E)$ are *adjacent* if $\{u, v\} \in E$. A vertex in a graph is *universal* if it is adjacent to all other vertices. A graph is said to be *nontrivial* if it contains more than one vertex. A *vertex cover* in G is a set C of vertices such that every edge has at least one endpoint in C. Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of G. A b-matching of G is a function $x : E \to \mathbb{Z}_+$ such that for each vertex v it holds that $\sum_{e \in E_v} x(e) \leq b(v)$, where E_v denotes the set of edges incident with v. A maximum weight b-matching of G is a b-matching of G such that the total weight $\sum_{e \in E} x(e)$ is maximum among all b-matchings of G. We use $\nu(G, b)$ to denote the total weight of a maximum weight b-matching of G.

Definitions for digraphs. A *digraph* is a pair $D = (V, A)$ where $V = V(D)$ is a finite set of *vertices* and $A = A(D)$ is a set of ordered pairs of vertices called *arcs*. Given an arc $a = (u, v) \in A$, we call u the *tail* of a and v its *head*. Two vertices u and v in a digraph $D = (V, A)$ are *adjacent* if $(u, v) \in A$ or $(v, u) \in A$. A *walk* in a digraph D is a sequence $(v_1, \ldots, v_k)$ of vertices of D such that $(v_i, v_{i+1}) \in A$ for all $i \in \{1, \ldots, k-1\}$. A *path* in D is a walk in which all vertices are pairwise distinct. Given a positive integer k, a *cycle of length* k in D is a path $(v_1, \ldots, v_k)$ such that $(v_k, v_1) \in A$. Note that a cycle of length one consists of a vertex v at which D has a loop (that is, an arc of the form (v, v)), and a cycle of length two consists of a pair of vertices u, v such that both arcs (u, v) and (v, u) exist. A digraph is *acyclic* if it contains no cycles; a directed acyclic graph is referred to as

a *DAG*. A digraph is *Hamiltonian* if it has a path containing every vertex. For a digraph $D = (V, A)$, the *underlying graph of* D is the undirected graph $U(D) = (V, E)$ in which two distinct vertices are adjacent if and only if they are adjacent in D.

Common definitions for graphs and digraphs. Let G be a graph or a digraph. An *independent set* in G is a set of pairwise nonadjacent vertices. Let f be a function from $V(G)$ to $\mathbb{Z}_+$. Given a set $S \subseteq V(G)$, we denote by $f(S)$ the value $\sum_{v \in S} f(v)$. Let $\alpha(G, f)$ denote the maximum value of $f(S)$ over all independent sets S in G. We say that G is *bipartite* if it admits a *bipartition*, that is, a partition of its vertex set into two (possibly empty) independent sets. More generally, G is *triangle-free* if it does not contain three pairwise adjacent vertices. Given a vertex $u \in V(G)$, the *degree* of u in G, denoted by $\deg_G(u)$ (or simply by $\deg(u)$ when the graph or digraph is clear from the context), is the number of vertices $v \in V(G)$ such that u and v are adjacent in G. Note that if D is a digraph without cycles of length one or two (in particular, if D is a DAG), then the vertex degrees are the same in D and in the underlying graph $U(D)$.

A *partially ordered set* (or: a *poset*) is a pair $(V, \preceq)$ where V is a finite set and $\preceq$ is a binary relation on V that is reflexive, antisymmetric, and transitive. We write $u \prec v$ if $u \preceq v$ and $u \neq v$. A poset element $v \in V$ is *minimal* if there is no element $u \in V \setminus \{v\}$ such that $u \prec v$.

2.1 Intersection representations of graphs and digraphs

In this section we introduce several definitions of intersection representation for graphs and digraphs. These variants will be used in the proof of our main result which will consist of a sequence of reductions from one variant to another.

A **weak directed intersection representation of a digraph** $D = (V, A)$ is a pair (U, φ) where U is a finite set of *colors* and φ is a *weak proper coloring* of D, that is, a mapping assigning to each vertex $v \in V$ a set $\varphi(v) \subseteq U$ such that for any two distinct vertices $u, v \in V$, it holds that $(u, v) \in A$ if and only if $\varphi(u) \cap \varphi(v) \neq \emptyset$ and $|\varphi(u)| \leq |\varphi(v)|$. Note that, with respect to the definition of a directed intersection representation given in Equation (1), the constraint on the cardinality is expressed by a weak inequality. Furthermore, if (U, φ) is a weak directed intersection representation of a digraph D and $W = (v_1, \ldots, v_k)$ is a walk in D, then it may happen that $|\varphi(v_1)| = \ldots = |\varphi(v_k)|$, in which case $(v_k, v_{k-1}, \ldots, v_1)$ is also a walk in D. In particular, D may contain cycles in which for each arc (u, v) the digraph D also contains the oppositely oriented arc (v, u). If G is a graph and D is the digraph obtained from G by replacing each edge with a pair of oppositely oriented arcs, then $DIN(D, \leq)$ equals the minimum cardinality of a set U such that G admits an intersection representation over the set U such that vertices in each component of G are assigned sets with the same cardinality (see, e.g., [4, 5, 19]).

The *cardinality* of a weak directed intersection representation (U, φ) of D is defined as the number of colors, that is, $|U|$. It is not hard to see that every DAG admits a weak directed intersection representation, e.g., (i) use a distinct color for each arc; (ii) fix a topological sorting (using the fact that it is a DAG); (iii) add to each vertex a distinct set of colors so that the cardinality of the color set of the vertices strictly increase along the chosen topological sorting[3]. The *weak directed intersection number* of a DAG $D = (V, A)$, denoted

[3] Note that the coloring defined by such procedure is also a (non-weak) directed intersection representation of the DAG. In fact, for a DAG, every weak directed intersection representation is also a directed intersection representation (see Lemma 7). Note, however, that the converse is not true as there are directed intersection representation of a DAG where non-adjacent vertices have intersecting color set of the same cardinality that are not weak directed intersection representation of the same graph.

by $DIN(D, \leq)$, is the smallest cardinality of a weak directed intersection representation of D. The weak directed intersection number has been previously considered in [20], where it was showed that the problem of computing $DIN(D, \leq)$ is NP-hard when D is an arbitrary DAG but polynomially solvable if D is an arborescence, which is in contrast with the NP-hardness of computing $DIN(D)$ for arborescences.

An **intersection representation of an undirected graph** $G = (V, E)$ is a pair (U, φ) where U is a finite set and φ is a mapping assigning to each vertex $v \in V$ a set $\varphi(v) \subseteq U$ such that for any two distinct vertices $u, v \in V$, it holds that $\{u, v\} \in E$ if and only if $\varphi(u) \cap \varphi(v) \neq \emptyset$. The *cardinality* of an intersection representation (U, φ) of G is defined as the cardinality of U. The *intersection number* of an undirected graph G, denoted by $IN(G)$, is the smallest cardinality of an intersection representation of G.

A *partially ordered graph* is a pair $(G, \preceq)$, where $G = (V, E)$ is an undirected graph and $\preceq$ is a partial order on the vertex set of G (that is, $(V, \preceq)$ is a poset). An **intersection representation of a partially ordered graph** $(G, \preceq)$ is a pair (U, φ) where U is a finite set of *colors* and φ is a *proper coloring* of $(G, \preceq)$, that is, a mapping assigning to each vertex $v \in V$ a set $\varphi(v) \subseteq U$ such that for any two distinct vertices $u, v \in V$, it holds that

- $\{u, v\} \in E$ if and only if $\varphi(u) \cap \varphi(v) \neq \emptyset$;
- if $u \prec v$ then $|\varphi(u)| < |\varphi(v)|$.

The *cardinality* of an intersection representation (U, φ) of a partially ordered graph $(G, \preceq)$ is defined as the cardinality of U. The *intersection number* of a partially ordered graph $(G, \preceq)$, denoted by $IN(G, \preceq)$, is the smallest cardinality of an intersection representation of $(G, \preceq)$.

Let $G = (V, E)$ be a graph and let $\ell : V \to \mathbb{Z}_+$ be a demand function on the vertices of G. An **ℓ-constrained intersection representation of an undirected graph** G is an intersection representation (U, φ) of G such that $|\varphi(v)| \geq \ell(v)$ for all $v \in V$. The *ℓ-constrained intersection number* of G, denoted by $IN(G, \ell)$, is the smallest cardinality of an ℓ-constrained intersection representation of G.

One can view the ℓ-constrained intersection number of a graph G and the intersection number of a partially ordered graph $(G, \preceq)$ as constrained variants of the intersection number of the graph G, placing it in a general framework proposed by Roberts in [16] along with numerous other variants of the intersection number studied in the literature.

When discussing algorithms on partially ordered graphs, we assume that a partially ordered graph $(G, \preceq)$ is represented with the adjacency lists of the graph G and an arbitrary DAG D on the same vertex set such that $u \preceq v$ if and only if there exists a directed u, v-path from u to v in D.

3 The DIN of triangle-free Hamiltonian DAGs

In this section, we present our main result about the polynomial computation of $DIN(D)$ (including a corresponding optimal directed intersection representation (U, φ) for D) for a triangle-free Hamiltonian DAG D. We prove the following theorem.

▶ **Theorem 1.** *Let $D = (V, A)$ be a triangle-free DAG with a Hamiltonian path $P = (v_1, \ldots, v_n)$. Let $w : V \to \mathbb{Z}_+$ be the vertex weight function on D defined recursively along P as follows: $w(v_1) = \deg(v_1)$, and for all $i \in \{2, \ldots, n\}$, we set $w(v_i) = \max\{w(v_{i-1}) + 1, \deg(v_i)\}$. Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of D defined by setting $b(v) = w(v) - \deg(v)$ for all $v \in V$ and let G be the underlying graph of D. Then $DIN(D) = |A| + b(V) - \nu(G, b)$ and a directed intersection representation (U, φ) of D with minimum cardinality can be computed in time $\mathcal{O}(|V|^3)$.*

3.1 Proof of Theorem 1

The proof of Theorem 1 involves several intermediate steps that allow us to state relationships among the variants of intersection representations introduced above. Here, we present these steps and the relationships they involve among variants of *IN* and *DIN*, and defer the proofs to the following subsections.

We first show that a weak directed intersection representation is equivalent to an intersection representation for any DAG that is Hamiltonian.

▶ **Lemma 2.** *Let $D = (V, A)$ be a Hamiltonian DAG. Then, $DIN(D) = DIN(D, \leq)$.*

We now show that computing a weak directed representation of a DAG D is equivalent to computing an intersection representation of a partially ordered graph $(G, \preceq)$, where G is the underlying graph of D and the partial order $\preceq$ is defined by the reachability relation in D.

▶ **Lemma 3.** *Let $D = (V, A)$ be a DAG. Let $G = (V, E)$ be the underlying graph of D and let $\preceq$ be the partial order on V defined by setting $u \preceq v$ if and only if there exists a u, v-path in D. Then, $DIN(D, \leq) = IN(G, \preceq)$.*

Then, we show that for a triangle-free graph G, and any given partial order $\preceq$ on the vertices of G, the computation of a weak intersection representation for the partially ordered graph $(G, \preceq)$ can be reduced to the computation of an ℓ-constrained intersection representation of G for a well defined and polynomially computable demand function ℓ.

▶ **Theorem 4.** *Let $(G, \preceq)$ be a partially ordered graph such that $G = (V, E)$ is triangle-free. Let M be the set of minimal elements in the poset $(V, \preceq)$. Let $\ell : V \to \mathbb{Z}_+$ be a demand function on the vertices of G defined by*

$$\ell(v) = \begin{cases} \deg(v) & \text{if } v \in M\,, \\ \max\left\{1 + \max_{u : u \prec v} \ell(u), \deg(v)\right\} & \text{if } v \notin M\,. \end{cases} \tag{2}$$

Then $IN(G, \preceq) = IN(G, \ell)$.

Figure 1 summarizes the relationships established by the above steps.

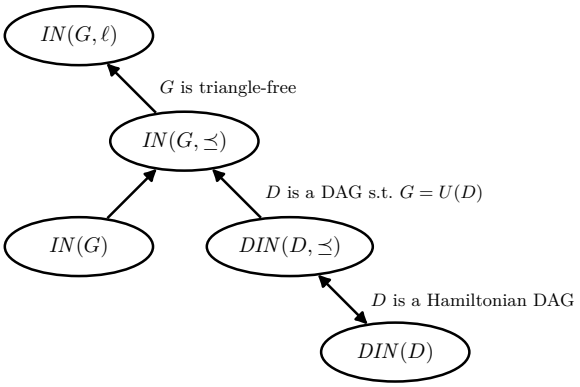


Figure 1 The relationships among the different types of intersection representations considered in this paper. The arc $Prob(a) \to Prob(b)$ is to be read "$Prob(a)$ is a special case of $Prob(b)$". A label on an arc specifies the restriction on the class of instances for which the relation is proved to hold.

Finally, we show that computing the ℓ-constrained intersection representation of a triangle-free graph G can be attained in polynomial time.

▶ **Theorem 5.** *Let $G = (V, E)$ be a triangle-free graph and $\ell : V \to \mathbb{Z}_+$ be a demand function on the vertices of G. Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of G defined by setting $b(v) = \max\{\ell(v) - \deg(v), 0\}$ for all $v \in V$. Then $IN(G, \ell) = |E| + b(V) - \nu(G, b)$ and an ℓ-constrained intersection representation (U, φ) of G with minimum cardinality can be computed in polynomial time, namely in time $\mathcal{O}(\min\{B|V|^2, |E|^2 \log |V| \log B\})$, where $B = \max_{v \in V} b(v)$.*

By following the above reductions in the opposite order, this final result extends to the computation of the directed intersection number of triangle-free Hamiltonian DAGs, i.e., it implies Theorem 1; we recall its statement and give a proof.

▶ **Theorem 1.** *Let $D = (V, A)$ be a triangle-free DAG with a Hamiltonian path $P = (v_1, \ldots, v_n)$. Let $w : V \to \mathbb{Z}_+$ be the vertex weight function on D defined recursively along P as follows: $w(v_1) = \deg(v_1)$, and for all $i \in \{2, \ldots, n\}$, we set $w(v_i) = \max\{w(v_{i-1}) + 1, \deg(v_i)\}$. Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of D defined by setting $b(v) = w(v) - \deg(v)$ for all $v \in V$ and let G be the underlying graph of D. Then $DIN(D) = |A| + b(V) - \nu(G, b)$ and a directed intersection representation (U, φ) of D with minimum cardinality can be computed in time $\mathcal{O}(|V|^3)$.*

Proof. Since D is a Hamiltonian DAG, Lemma 2 implies that $DIN(D) = DIN(D, \leq)$. Let $G = (V, E)$ denote the underlying graph of D and $\preceq$ the partial order on V such that $u \preceq v$ if and only if there exists a u, v-path in D. By Lemma 3, we obtain that $DIN(D, \leq) = IN(G, \preceq)$. We define the demand function $\ell : V \to \mathbb{Z}_+$ as in Theorem 4, which, by Theorem 4, implies that $IN(G, \preceq) = IN(G, \ell)$. Finally, Theorem 5 guarantees that $IN(G, \ell) = |E| + b(V) - \nu(G, b)$ and thus that $DIN(D) = |A| + b(V) - \nu(G, b)$, as claimed. ◀

As a corollary of Theorem 5, we also obtain a characterization of the intersection number of a partially ordered triangle-free graph.

▶ **Corollary 6.** *Let $(G, \preceq)$ be a partially ordered graph such that $G = (V, E)$ is triangle-free. Let M be the set of minimal elements in the poset $(V, \preceq)$. Let $w : V \to \mathbb{Z}_+$ be the vertex weight function on G defined by*

$$w(v) = \begin{cases} \deg(v) & \text{if } v \in M\,, \\ \max\left\{1 + \max\limits_{u : u \prec v} w(u), \deg(v)\right\} & \text{if } v \notin M\,. \end{cases} \tag{3}$$

Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of G defined by setting $b(v) = w(v) - \deg(v)$ for all $v \in V$. Then $IN(G, \preceq) = |E| + b(V) - \nu(G, b)$ and an intersection representation (U, φ) of $(G, \preceq)$ with minimum cardinality can be computed in time $\mathcal{O}(|V|^3)$.

3.2 On the relationships between DIN and weak DIN of DAGs

We first show that for arbitrary DAGs, the weak directed intersection number is always an upper bound on the directed intersection number.

▶ **Lemma 7.** *Let $D = (V, A)$ be a DAG. Then, $DIN(D) \leq DIN(D, \leq)$.*

Proof. The claim will follow from showing that any weak directed intersection representation of D is a directed intersection representation of D.

Let (U, φ) be a weak directed intersection representation of D. Then, $(u, v) \in A$ if and only if $\varphi(u) \cap \varphi(v) \neq \emptyset$ and $|\varphi(u)| \leq |\varphi(v)|$. Moreover, since D is a DAG, whenever $(u, v) \in A$, there is no arc (v, u), hence the inequality between the color sets' cardinalities must be

strict, i.e., $|\varphi(u)| < |\varphi(v)|$. If $(u,v) \notin A$, then either $\varphi(u) \cap \varphi(v) = \emptyset$ or $|\varphi(u)| > |\varphi(v)|$, which implies that either $\varphi(u) \cap \varphi(v) = \emptyset$ or $|\varphi(u)| \geq |\varphi(v)|$. Hence, (U, φ) is also a directed intersection representation of D. ◀

▶ **Lemma 2.** *Let $D = (V, A)$ be a Hamiltonian DAG. Then, $DIN(D) = DIN(D, \leq)$.*

Proof. By Lemma 7, it suffices to show that if D is Hamiltonian, then any directed intersection representation of D is a weak directed intersection representation of D.

Let (U, φ) be a directed intersection representation of D. Clearly, we have that for each $(u,v) \in A$ the fact that $\varphi(u) \cap \varphi(v) \neq \emptyset$ and $|\varphi(u)| < |\varphi(v)|$ implies that

$$\varphi(u) \cap \varphi(v) \neq \emptyset \text{ and } |\varphi(u)| \leq |\varphi(v)|. \tag{4}$$

Moreover, the existence of a Hamiltonian path in D implies that for each pair of vertices u, v we have $|\varphi(u)| \neq |\varphi(v)|$. Therefore, if $(u,v) \notin A$, since (U, φ) is a directed intersection representation, at least one of the following must hold: (i) $|\varphi(u)| \geq |\varphi(v)|$, and hence $|\varphi(u)| > |\varphi(v)|$; (ii) $\varphi(u) \cap \varphi(v) = \emptyset$. The fact that (4) holds for each arc (u,v) and (i) or (ii) holds whenever $(u,v) \notin A$ implies that (U, φ) is a weak directed intersection representation of D. ◀

3.3 A relationship between weak directed intersection representations of DAGs and intersection representations of partially ordered graphs

▶ **Lemma 3.** *Let $D = (V, A)$ be a DAG. Let $G = (V, E)$ be the underlying graph of D and let $\preceq$ be the partial order on V defined by setting $u \preceq v$ if and only if there exists a u,v-path in D. Then, $DIN(D, \leq) = IN(G, \preceq)$.*

Proof. Let (U, φ) be a weak directed intersection representation of D.

▷ Claim 8. Fix distinct vertices u, v. If there is a u,v-path then $|\varphi(u)| < |\varphi(v)|$.

Proof. Let (u,v) be an arc of D. Then, we have $|\varphi(u)| \leq |\varphi(v)|$ and $\varphi(u) \cap \varphi(v) \neq \emptyset$. The latter, together with $(v,u) \notin A$ (since D is a DAG), implies $|\varphi(u)| < |\varphi(v)|$. The claim now follows by transitivity, repeatedly using the above argument on the arcs of a u,v-path. ◁

Let us show that (U, φ) satisfies the two properties defining an intersection representation of $(G, \preceq)$. Let u, v be a pair of distinct vertices in G.

1. Assume that $u \prec v$. By the definition of $\prec$ there is a u,v-path in D, hence by the claim above we have $|\varphi(u)| < |\varphi(v)|$.
2. Let $\{u,v\} \in E$. Then, either $(u,v) \in A$ or $(v,u) \in A$, and in either case, $\varphi(u) \cap \varphi(v) \neq \emptyset$. Similarly, if $\varphi(u) \cap \varphi(v) \neq \emptyset$, then either (u,v) or (v,u) is an arc of D, which implies that $\{u,v\} \in E$.

Let us now assume that (U, φ) is an intersection representation of $(G, \preceq)$. Then, for each pair of distinct vertices u, v of D we have the following.

- If $(u,v) \in A$ then: (i) $\{u,v\} \in E$; and (ii) $u \prec v$. Hence, from (i) and (ii) respectively, $\varphi(u) \cap \varphi(v) \neq \emptyset$, and $|\varphi(u)| < |\varphi(v)|$.
- Assume now that: (i) $\varphi(u) \cap \varphi(v) \neq \emptyset$, and (ii) $|\varphi(u)| < |\varphi(v)|$. From (i) we have $\{u,v\} \in E$. Since G is the underlying graph of D and D is a DAG, it follows that exactly one of the arcs $(u,v), (v,u)$ is in A. However, we cannot have $(v,u) \in A$, for otherwise $v \prec u$, which, together with (U, φ) being an intersection representation of $(G, \preceq)$, would contradict the hypothesis $|\varphi(u)| < |\varphi(v)|$. Therefore, we must have $(u,v) \in A$.

The two items imply that (U, φ) is a weak directed intersection representation of D and conclude the proof. ◀

3.4 From partially ordered triangle-free graphs to ℓ-constrained triangle-free graphs

▶ **Theorem 4.** *Let $(G, \preceq)$ be a partially ordered graph such that $G = (V, E)$ is triangle-free. Let M be the set of minimal elements in the poset $(V, \preceq)$. Let $\ell : V \to \mathbb{Z}_+$ be a demand function on the vertices of G defined by*

$$\ell(v) = \begin{cases} \deg(v) & \text{if } v \in M\,, \\ \max\left\{1 + \max\limits_{u:u\prec v} \ell(u), \deg(v)\right\} & \text{if } v \notin M\,. \end{cases} \tag{2}$$

Then $IN(G, \preceq) = IN(G, \ell)$.

Proof. The claim will follow from showing that any intersection representation of $(G, \preceq)$ is an ℓ-constrained intersection representation of G, and that there exists an ℓ-constrained intersection representation of G with minimum cardinality that is also an intersection representation of $(G, \preceq)$.

Let (U, φ) be an intersection representation of $(G, \preceq)$. To show that (U, φ) is an ℓ-constrained intersection representation of G, we need to show that for each vertex $v \in V$, we have $|\varphi(v)| \geq \ell(v)$. By the definition of an intersection representation of a partially ordered graph, we have that

$$|\varphi(v)| > |\varphi(u)| \text{ for each } u \prec v. \tag{5}$$

Moreover, because of the triangle-free condition we have that the neighborhood of v is an independent set. Again, by the definition of intersection representation, this implies that the corresponding color sets are pairwise disjoint. On the other hand, each one of these color sets has a nonempty intersection with $\varphi(v)$. It follows that $|\varphi(v)| \geq \deg(v)$. This, together with Equation (5), implies that $|\varphi(v)| \geq \ell(v)$. Thus, (U, φ) is an ℓ-constrained intersection representation of G.

Assume now that (U, φ) is an ℓ-constrained intersection representation of G such that $|U| = IN(G, \ell)$. For each edge $e = \{u, v\} \in E$, we have $\varphi(u) \cap \varphi(v) \neq \emptyset$; in particular, there exists a color $c_e \in \varphi(u) \cap \varphi(v)$. We show next that for any two distinct edges $e, e' \in E$, we have $c_e \neq c_{e'}$. Suppose for a contradiction that there exist two different edges $e = \{u, v\} \in E$ and $e' = \{x, y\} \in E$ such that $c_e = c_{e'}$. Since $e \neq e'$ and G does not contain loops or duplicated edges, the set $\{u, v, x, y\}$ has cardinality at least 3. Therefore, there is a color shared by at least three vertices, and, since the coloring is proper, this implies that any such three vertices form a triangle, contradicting the hypothesis that G is triangle-free.

Let us write $U_E = \{c_e : e \in E\}$ and for all $v \in V$, denote $\varphi'(v) = \varphi(v) \setminus U_E$. Then $|\varphi'(v)| = |\varphi(v)| - \deg(v)$. Since the representation is ℓ-constrained, we have for any vertex $v \in V$ that $|\varphi'(v)| = |\varphi(v)| - \deg(v) \geq \ell(v) - \deg(v) \geq 0$. For each $v \in V$ such that $|\varphi'(v)| > \ell(v) - \deg(v)$, we select an arbitrary set $X_v \subseteq \varphi'(v)$ such that $|X_v| = \ell(v) - \deg(v)$. Then, we define, for each $v \in V$

$$\psi(v) = \begin{cases} (\varphi(v) \cap U_E) \cup X_v, & \text{if } |\varphi'(v)| > \ell(v) - \deg(v)\,, \\ \varphi(v), & \text{otherwise.} \end{cases}$$

Similarly as above, let us denote $\psi'(v) = \psi(v) \setminus U_E$ for all $v \in V$. By construction, we have $|\psi'(v)| = \ell(v) - \deg(v)$ for all $v \in V$ and consequently $|\psi(v)| = |\psi'(v)| + \deg(v) = \ell(v)$ for all $v \in V$. Note that the recursive definition of the function ℓ implies that $\ell(u) < \ell(v)$ whenever $u \prec v$, and hence $|\psi(u)| < |\psi(v)|$ whenever $u \prec v$. Next, we show that (U, ψ) is

an intersection representation of $(G, \preceq)$. To this end, it remains to show that $\{u, v\} \in E$ if and only if $\psi(u) \cap \psi(v) \neq \emptyset$. If $\{u, v\} = e \in E$ then $c_e \in \varphi(u) \cap \varphi(v) \cap U_E \subseteq \psi(u) \cap \psi(v)$ since we obtained the mapping ψ from φ by only removing colors not in U_E. If $\{u, v\} \notin E$ then $\psi(u) \cap \psi(v) = \emptyset$ follows directly from the relations $\psi(u) \subseteq \varphi(u)$ and $\psi(v) \subseteq \varphi(v)$, using the fact that $\varphi(u) \cap \varphi(v) = \emptyset$. This shows that (U, ψ) is an intersection representation of $(G, \preceq)$. Furthermore, since $|\psi(v)| = \ell(v)$ for all $v \in V$ and $|U| = IN(G, \ell)$, the pair (U, ψ) is an ℓ-constrained intersection representation of G with minimum cardinality. This completes the proof. ◀

3.5 The ℓ-constrained intersection number of triangle-free graphs

▶ **Theorem 5.** *Let $G = (V, E)$ be a triangle-free graph and $\ell : V \to \mathbb{Z}_+$ be a demand function on the vertices of G. Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of G defined by setting $b(v) = \max\{\ell(v) - \deg(v), 0\}$ for all $v \in V$. Then $IN(G, \ell) = |E| + b(V) - \nu(G, b)$ and an ℓ-constrained intersection representation (U, φ) of G with minimum cardinality can be computed in polynomial time, namely in time $\mathcal{O}(\min\{B|V|^2, |E|^2 \log |V| \log B\})$, where $B = \max_{v \in V} b(v)$.*

Proof. First, we prove that $|E| + b(V) - \nu(G, b)$ is an upper bound on the ℓ-constrained intersection number of the graph G by constructing an ℓ-constrained intersection representation (U, φ) with cardinality at most $|E| + b(V) - \nu(G, b)$. We associate to each edge $e \in E$ a unique color c_e and define $U_E = \{c_e : e \in E\}$. Let $x : E \to \mathbb{Z}_+$ be a maximum weight b-matching in G. Then $\sum_{e \in E} x(e) = \nu(G, b)$. We denote by E_v the set of edges in G incident with a vertex v. To each edge e of G, we associate another set C_e of $x(e)$ new colors, and to each vertex v of G, we associate a set C_v of $b(v) - \sum_{e \in E_v} x(e)$ new colors, so that the sets U_E, C_e, $e \in E$, and C_v, $v \in V$, are pairwise disjoint. Note that this construction is indeed possible: for each vertex v, the value of $b(v) - \sum_{e \in E_v} x(e)$ is a non-negative integer since x is a b-matching in G. We define

$$U = U_E \cup \left(\bigcup_{e \in E} C_e \right) \cup \left(\bigcup_{v \in V} C_v \right) \quad \text{and} \quad \varphi(v) = \{c_e : e \in E_v\} \cup C_v \cup \left(\bigcup_{e \in E_v} C_e \right). \quad (6)$$

Next, we show that (U, φ) is an ℓ-constrained intersection representation of G. Clearly $\varphi(v) \subseteq U$ for all $v \in V$. Furthermore, for each vertex $v \in V$, by definition it holds that $b(v) + \deg(v) \geq \ell(v)$, and we have

$$\begin{aligned} |\varphi(v)| = |E_v| + |C_v| + \sum_{e \in E_v} |C_e| &= \deg(v) + b(v) - \sum_{e \in E_v} x(e) + \sum_{e \in E_v} x(e) \\ &= \deg(v) + b(v) \geq \ell(v). \end{aligned} \quad (7)$$

Therefore, the color sets assigned to the vertices by φ satisfy the lower bounds defined by the demand function ℓ.

We now prove that (U, φ) is an intersection representation of G, by showing that for any distinct vertices u and v of G, it holds that $(u, v) \in E$ if and only if $\varphi(v_i) \cap \varphi(v_j) \neq \emptyset$.

For this, assume first that $e = \{u, v\} \in E$. Then $e \in E_u \cap E_v$ and thus $c_e \in \varphi(u) \cap \varphi(v)$, implying that $\varphi(u) \cap \varphi(v) \neq \emptyset$. For the converse direction, assume that $\varphi(u) \cap \varphi(v) \neq \emptyset$. Let $c \in \varphi(u) \cap \varphi(v)$. Because for any two vertices $u, v \in V$, we have that $C_u \subseteq \varphi(v)$ if and only if $u = v$, the color c cannot belong to any set C_z for $z \in V$. Therefore, since the sets U_E, and C_e, $e \in E$, are pairwise disjoint, the color c must belong to either U_E or to some set C_e for $e \in E$. Assume first that $c \in U_E$. Then $c = c_e$ for some $e \in E$, which implies that

$e \in E_u \cap E_v$ and thus $e = \{u, v\}$, i.e., $\{u, v\}$ is an edge of E. Assume now that $c \in C_e$ for some $e \in E$. Since $c \in \varphi(u) \cap \varphi(v)$, we infer that $e \in E_u$ and $e \in E_v$, which again implies that $\{u, v\}$ is an edge of G. This concludes the proof that φ is a proper coloring of (G, φ).

It remains to show that $|U| = |E| + b(V) - \nu(G, b)$. Using the definition of U we infer that, as claimed, its cardinality is

$$\begin{aligned} |U| &= |U_E| + \sum_{e \in E} |C_e| + \sum_{v \in V} |C_v| = |E| + \sum_{e \in E} x_e + \sum_{v \in V} \left(b(v) - \sum_{e \in E_v} x(e) \right) \\ &= |E| + \nu(G, b) + \sum_{v \in V} b(v) - \sum_{v \in V} \sum_{e \in E_v} x(e) = |E| + \nu(G, b) + b(V) - 2 \sum_{e \in E} x(e) \\ &= |E| + \nu(G, b) + b(V) - 2\nu(G, b) = |E| + b(V) - \nu(G, b). \end{aligned}$$

Second, we prove that $|E| + b(V) - \nu(G, b)$ is a lower bound for the ℓ-constrained intersection number of G. Let (U, φ) be an arbitrary ℓ-constrained intersection representation of G. We need to show that $|U| \geq |E| + b(V) - \nu(G, b)$. For each edge $e = \{u, v\} \in E$, we have $\varphi(u) \cap \varphi(v) \neq \emptyset$; in particular, there exists a color $c_e \in \varphi(u) \cap \varphi(v)$. We show next that for any two distinct edges $e, e' \in E$, we have $c_e \neq c_{e'}$. Suppose for a contradiction that there exist two different edges $e = \{u, v\} \in E$ and $e' = \{x, y\} \in E$ such that $c_e = c_{e'}$. Since $e \neq e'$ and G does not contain loops or duplicated edges, the set $\{u, v, x, y\}$ has cardinality at least 3. Therefore, there is a color shared by at least three vertices, and, since the coloring is proper, this implies that any such three vertices form a triangle, contradicting the hypothesis that G is triangle-free.

Because of the triangle-free condition we have that the neighborhood of v is an independent set. By the definition of i*se* , this implies that the corresponding color sets are pairwise disjoint. On the other hand, each one of these color sets has a nonempty intersection with $\varphi(v)$. It follows that $|\varphi(v)| \geq \deg(v)$. By the assumption that (U, φ) is an ℓ-constrained intersection representation of G we also have $|\varphi| \geq \ell(v)$, hence $|\varphi| \geq \max\{\ell(v), \deg(v)\}$.

For all $v \in V$, let us denote $\varphi'(v) = \varphi(v) \setminus U_E$. Then $|\varphi'(v)| = |\varphi(v)| - \deg(v) \geq 0$. It follows that for any vertex $v \in V$, we have that

$$|\varphi'(v)| = |\varphi(v)| - \deg(v) \geq \max\{0, \ell(v) - \deg(v)\} = b(v).$$

▷ Claim 9. We may assume without loss of generality that $|\varphi'(v)| = b(v)$ for all $v \in V$.

Proof. Suppose that this is not the case. Then, we choose for each $v \in V$ such that $|\varphi'(v)| > b(v)$, an arbitrary set $X_v \subseteq \varphi'(v)$ such that $|X_v| = b(v)$ and define, for all $v \in V$

$$\psi(v) = \begin{cases} (\varphi(v) \cap U_E) \cup X_v, & \text{if } |\varphi'(v)| > b(v)\,, \\ \varphi(v), & \text{otherwise.} \end{cases}$$

Similarly as above, let us denote $\psi'(v) = \psi(v) \setminus U_E$ for all $v \in V$. By construction, we have $|\psi'(v)| = b(v)$ for all $v \in V$ and consequently $|\psi(v)| = |\psi'(v)| + \deg(v) \geq \ell(v)$ for all $v \in V$. Hence, ψ satisfies the ℓ-constraints on the vertices. To see that (U, ψ) is indeed an ℓ-constrained intersection representation of G, it remains to show that $\{u, v\} \in E$ if and only if $\psi(u) \cap \psi(v) \neq \emptyset$. If $\{u, v\} = e \in E$ then $c_e \in \varphi(u) \cap \varphi(v) \cap U_E \subseteq \psi(u) \cap \psi(v)$ since we obtained the mapping ψ from φ by only removing colors not in U_E. If $\{u, v\} \notin E$ then $\psi(u) \cap \psi(v) = \emptyset$ follows directly from the relations $\psi(u) \subseteq \varphi(u)$ and $\psi(v) \subseteq \varphi(v)$, using the fact that $\varphi(u) \cap \varphi(v) = \emptyset$. This shows that (U, ψ) is an intersection representation of G and completes the proof of the claim. ◁

Let $U_E = \{c_e : e \in E\}$. Then $|U_E| = |E|$. To complete the proof of the lower bound, we need to show that $|U \setminus U_E| \geq b(V) - \nu(G,b)$, or, equivalently, that $b(V) \leq |U \setminus U_E| + \nu(G,b)$. Consider the function $x : E \to \mathbb{Z}_+$ defined by setting

$$x(e) = |\varphi'(u) \cap \varphi'(v)| \qquad \text{for each edge } e = \{u,v\} \in E\,.$$

We claim that x is a b-matching of G. Let v be a vertex of G. Let E_v be the set of edges in G that are incident with v and let $N(v)$ be the set of neighbors of v in G. Recall that by the assumption that the graph is triangle-free, each color $c \in U \setminus U_E$ appears in at most two of the color sets $\varphi'(u)$, $u \in V$. Therefore, for each vertex $v \in V$, the sets $\varphi'(u) \cap \varphi'(v)$, $u \in N(v)$, are pairwise disjoint. This implies that

$$\sum_{e \in E_v} x(e) = \sum_{u \in N(v)} |\varphi'(u) \cap \varphi'(v)| = \left| \varphi'(v) \cap \left(\bigcup_{u \in N(v)} \varphi'(u) \right) \right| \leq |\varphi'(v)| = b(v)\,,$$

and hence x is indeed a b-matching of D. For each $v \in V$, let us denote by C_v the set of *private* colors of v, that is, those colors $c \in \varphi'(v)$ such that $c \notin \varphi'(u)$ for all $u \in V \setminus \{v\}$. We have

$$\begin{aligned} b(V) &= \sum_{v \in V} b(v) = \sum_{v \in V} |\varphi'(v)| = \sum_{v \in V} \left(|C_v| + |\varphi'(v) \setminus C_v|\right) \\ &= \sum_{v \in V} |C_v| + \sum_{v \in V} \sum_{u \in N(v)} |\varphi'(u) \cap \varphi'(v)| = \sum_{v \in V} |C_v| + 2 \sum_{e \in E} x(e) \\ &= \left(\sum_{v \in V} |C_v| + \sum_{e \in E} x(e) \right) + \sum_{e \in E} x(e) \leq |U \setminus U_E| + \nu(G,b)\,, \end{aligned}$$

where the last inequality follows from the fact that each color in $U \setminus U_E$ is counted exactly once in the sum $\sum_{v \in V} |C_v| + \sum_{e \in E} x(e)$ and that $\sum_{e \in E} x(e) \leq \nu(G,b)$, since x is a b-matching in D.

Finally, we observe that an ℓ-constrained intersection representation (U, φ) of G with minimum cardinality can be computed in polynomial time. First, we compute the capacity function b according to the definition. Including also the time for the computation of the vertex degrees, this can be done in time $\mathcal{O}(|V| + |E|) = \mathcal{O}(|V|^2)$. Then we compute a maximum weight b-matching x in G for which we can either use the algorithm by Pulleyblank [14] (see also [18]), that requires time $\mathcal{O}(B|V|^2)$, or (when B is superpolynomial in $|V|$ and/or for the case of G being sparse) the algorithm of Gabow [8] which runs in $\mathcal{O}(|E|^2 \log |V| \log B)$. Finally, we use Equations (6) to compute an intersection representation (U, φ) of $(G, \preceq)$ with cardinality $|E| + b(V) - \nu(G,b)$. This can be done in time proportional to the total size of this representation, which is $\sum_{v \in V} |\varphi(v)| = \sum_{v \in V} w(v) = \mathcal{O}(|V|^2)$. Since we can choose the best of the two above options for the computation of maximum weight b-matching x, we conclude that the overall running time satisfies $\mathcal{O}(\min\{B|V|^2, |E|^2 \log |V| \log B\})$. ◀

3.6 Intersection number of triangle-free partially ordered graphs

▶ **Corollary 6.** *Let $(G, \preceq)$ be a partially ordered graph such that $G = (V,E)$ is triangle-free. Let M be the set of minimal elements in the poset $(V, \preceq)$. Let $w : V \to \mathbb{Z}_+$ be the vertex weight function on G defined by*

$$w(v) = \begin{cases} \deg(v) & \text{if } v \in M\,, \\ \max\left\{1 + \max\limits_{u : u \prec v} w(u), \deg(v)\right\} & \text{if } v \notin M\,. \end{cases} \tag{3}$$

Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of G defined by setting $b(v) = w(v) - \deg(v)$ for all $v \in V$. Then $IN(G, \preceq) = |E| + b(V) - \nu(G, b)$ and an intersection representation (U, φ) of $(G, \preceq)$ with minimum cardinality can be computed in time $\mathcal{O}(|V|^3)$.

Proof. The claim $IN(G, \preceq) = |E| + b(V) - \nu(G, b)$ follows directly from observing that by Theorem 4, upon defining a demand function ℓ on the vertices of G such that $\ell(v) = w(v)$ for each $v \in V$, we have $IN(G, \preceq) = IN(G, \ell)$. Moreover, by Theorem 5, we have $IN(G, \ell) = |E| + b(V) - \nu(G, b)$.

Finally, we show that a directed intersection representation (U, φ) of $(G, \preceq)$ with minimum cardinality can be computed in time $\mathcal{O}(|V|^3)$. For this, we first observe that the weight function w according to the definition given by Equation (3) and the capacity function b can be computed in time $\mathcal{O}(|V| + |A|) = \mathcal{O}(|V|^2)$ where $D = (V, A)$ is the DAG representing the poset $(V, \preceq)$. Then, we note that from the definition, it follows that for each vertex $v \in V$, we have $b(v) \leq w(v) \leq \Delta + L$ where Δ denotes the maximum vertex degree in G and L denotes the maximum length (that is, the number of arcs) in a directed path in D. Therefore $B \leq 2(|V| - 1)$, hence $B|V|^2 = O(|V|^3)$, which implies that, using the algorithm by Pulleyblank [14], we can compute a maximum weight b-matching in G in time $\mathcal{O}(|V|^3)$. And in particular, the bound on the construction of (U, φ) follows from Theorem 4, using $B|V|^2 = O(|V|^3)$. ◀

4 Some final observations and open questions

The bipartite case. In a bipartite graph G, the maximum size of a b-matching is equal to the minimum b-weight of a vertex cover (see, e.g., [18]). Using this, we can give an alternative expression of the result in Theorem 1 in the case of bipartite Hamiltonian DAGs.

▶ **Theorem 10.** *Let $D = (V, A)$ be a bipartite DAG with a Hamiltonian path $P = (v_1, \ldots, v_n)$. Let $w : V \to \mathbb{Z}_+$ be a vertex weight function on D defined recursively along P as follows: $w(v_1) = \deg(v_1)$, and for all $i \in \{2, \ldots, n\}$, we set $w(v_i) = \max\{w(v_{i-1}) + 1, \deg(v_i)\}$. Let $b : V \to \mathbb{Z}_+$ be a capacity function on the vertices of D defined by setting $b(v) = w(v) - \deg(v)$ for all $v \in V$. Then $DIN(D) = |A| + \alpha(D, b)$.*

$\mathcal{O}(1)$-approximations. Before this paper, the only class of graphs for which a constant approximation polynomial algorithm was known for computing the *DIN* was the class of arborescences. As a consequence of our results, we can significantly widen the class of DAGs where the problem admits a constant approximation as recorded in the following observation (the details are deferred to the full version of the paper).

▶ **Observation 11.** *The DIN can be approximated in polynomial time to a constant factor on DAGs obtained from graphs with bounded chromatic number and bounded path cover number by orienting the edges along a bounded path cover.*

Argument: such graphs need $\Omega(n^2)$ colors, which matches the algorithmic bound from Liu et al. [11] to within a constant. The lower bound can be argued by taking a path with linearly many vertices and finding a constant-fraction independent set along that path.

Other open questions are about the extent, in terms of graph classes, to which the relationships between the different intersection representations hold (see the diagram in Figure 1). For example, does the fact that the ℓ-constrained intersection number of a graph G generalizes the computation of intersection number over any partially ordered graph $(G, \preceq)$ hold beyond the class of triangle-free graphs? In particular, we can show that this is true in the larger

class of diamond-free graphs, where the diamond is the graph obtained from the 4-vertex complete graph by removing an edge (details are deferred to the full version of the paper). However, for assessing the complexity of such a reduction one would need to determine the complexity of the following problem: Given a diamond-free graph G and a function $b : V(G) \to \mathbb{Z}_+$, find a function $x : \mathcal{C}(G) \to \mathbb{Z}_+$, where $\mathcal{C}(G)$ is the set of maximal cliques of G, such that for all $v \in V(G)$, the sum of the values $x(C)$ over all maximal cliques C containing v does not exceed $b(v)$, and the sum $\sum_{C \in \mathcal{C}(G)} x(C)$ is maximized.

References

1 Andrea Caucchiolo and Ferdinando Cicalese. On the complexity of directed intersection representation of dags. In Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors, *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*, volume 12273 of *Lecture Notes in Computer Science*, pages 554–565. Springer, 2020.

2 Andrea Caucchiolo and Ferdinando Cicalese. On the intractability landscape of digraph intersection representations. In Cristina Bazgan and Henning Fernau, editors, *Combinatorial Algorithms - 33rd International Workshop, IWOCA 2022, Trier, Germany, June 7-9, 2022, Proceedings*, volume 13270 of *Lecture Notes in Computer Science*, pages 270–284. Springer, 2022.

3 Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.

4 Hiroshi Era and Morimasa Tsuchiya. On intersection graphs with respect to uniform families. *Utilitas Math.*, 37:3–11, 1990.

5 Hiroshi Era and Morimasa Tsuchiya. On intersection numbers of graphs. In *Graph theory, combinatorics, algorithms, and applications (San Francisco, CA, 1989)*, pages 545–556. SIAM, Philadelphia, PA, 1991.

6 Paul Erdős, A. W. Goodman, and Louis Pósa. The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112, 1966.

7 Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, 1998.

8 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 448–456, New York, NY, USA, 1983. Association for Computing Machinery.

9 Alexandr V. Kostochka, Xujun Liu, Roberto Machado, and Olgica Milenkovic. Directed intersection representations and the information content of digraphs. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1477–1481. IEEE, 2019.

10 Lawrence T. Kou, Larry J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM*, 21(2):135–139, 1978.

11 Xujun Liu, Roberto Assis Machado, and Olgica Milenkovic. Directed intersection representations and the information content of digraphs. *IEEE Trans. Inform. Theory*, 67(1):347–357, 2021.

12 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

13 James Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977.

14 William R. Pulleyblank. *FACES OF MATCHING-POLYHEDRA*. ProQuest LLC, Ann Arbor, MI, 1973. Thesis (Ph.D.)–University of Waterloo (Canada).

15 Norman J. Pullman. Clique coverings of graphs—a survey. In *Combinatorial mathematics, X (Adelaide, 1982)*, volume 1036 of *Lecture Notes in Math.*, pages 72–85. Springer, Berlin, 1983.

16 Fred S. Roberts. On the mobile radio frequency assignment problem and the traffic light phasing problem. In *Second International Conference on Combinatorial Mathematics (New York, 1978)*, volume 319 of *Ann. New York Acad. Sci.*, pages 466–483. New York Acad. Sci., New York, 1979.

17 Fred S Roberts. Applications of edge coverings by cliques. *Discrete applied mathematics*, 10(1):93–109, 1985.

18 Alexander Schrijver. *Combinatorial optimization. Polyhedra and efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

19 Morimasa Tsuchiya. On uniform intersection numbers. *Ars Combin.*, 48:225–232, 1998.

20 Matteo Zeggiotti. On the complexity of the RDIN Problem. Master's thesis, Master's degree in Computer Science and Engineering, University of Verona, 2021.

On Finding Short Reconfiguration Sequences Between Independent Sets

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India

Soumita Hait
Indian Institute of Technology, Kharagpur, India

Amer E. Mouawad
American University of Beirut, Lebanon
Universität Bremen, Germany

Abstract

Assume we are given a graph G, two independent sets S and T in G of size $k \geq 1$, and a positive integer $\ell \geq 1$. The goal is to decide whether there exists a sequence $\langle I_0, I_1, ..., I_\ell \rangle$ of independent sets such that for all $j \in \{0, \ldots, \ell - 1\}$ the set I_j is an independent set of size k, $I_0 = S$, $I_\ell = T$, and I_{j+1} is obtained from I_j by a predetermined reconfiguration rule. We consider two reconfiguration rules, namely token sliding and token jumping. Intuitively, we view each independent set as a collection of tokens placed on the vertices of the graph. Then, the Token Sliding Optimization (TSO) problem asks whether there exists a sequence of at most ℓ steps that transforms S into T, where at each step we are allowed to slide one token from a vertex to an unoccupied neighboring vertex (while maintaining independence). In the Token Jumping Optimization (TJO) problem, at each step, we are allowed to jump one token from a vertex to any other unoccupied vertex of the graph (as long as we maintain independence). Both TSO and TJO are known to be fixed-parameter tractable when parameterized by ℓ on nowhere dense classes of graphs. In this work, we investigate the boundary of tractability for sparse classes of graphs. We show that both problems are fixed-parameter tractable for parameter $k + \ell + d$ on d-degenerate graphs as well as for parameter $|M| + \ell + \Delta$ on graphs having a modulator M whose deletion leaves a graph of maximum degree Δ. We complement these result by showing that for parameter ℓ alone both problems become W[1]-hard already on 2-degenerate graphs. Our positive result makes use of the notion of independence covering families introduced by Lokshtanov et al. [25]. Finally, we show as a side result that using such families we can obtain a simpler and unified algorithm for the standard Token Jumping Reachability problem (a.k.a. Token Jumping) parameterized by k on both degenerate and nowhere dense classes of graphs.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Token sliding, token jumping, fixed-parameter tractability, combinatorial reconfiguration, shortest reconfiguration sequence

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.39

Related Version *Full Version*: https://arxiv.org/abs/2209.05145

Funding *Amer E. Mouawad*: Research supported by the Alexander von Humboldt Foundation, by PHC Cedre project 2022 "PLR", and partially supported by URB project "A theory of change through the lens of reconfiguration".

1 Introduction

Given a simple undirected graph G, a set of vertices $I \subseteq V(G)$ is an *independent set* if the vertices of I are pairwise non-adjacent. Finding an independent set of size k, i.e., the Independent Set (IS) problem, is known to be NP-complete [22] and W[1]-complete parameterized by solution size k [12]. We view an independent set as a collection of k

tokens placed on the vertices of a graph such that no two tokens are placed on adjacent vertices. This gives rise to two natural adjacency relations between independent sets (or token configurations), also called *reconfiguration steps*. These reconfiguration steps, in turn, give rise to several *combinatorial reconfiguration* problems [28, 27, 8].

In the Token Sliding Reachability (TSR) problem, introduced by Hearn and Demaine [15], two independent sets are adjacent if one can be obtained from the other by removing a token from a vertex u and immediately placing it on another unoccupied vertex v with the requirement that $\{u, v\}$ must be an edge of the graph. The token is said to *slide* from vertex u to vertex v along the edge $\{u, v\}$. Generally speaking, in the Token Sliding Reachability problem, we are given a graph G and two independent sets S and T of size k in G. The goal is to decide whether there exists a sequence of slides (a *reconfiguration sequence*) that transforms S to T. The TSR problem has been extensively studied [5, 6, 11, 14, 18, 21, 24]. It is known that the problem is PSPACE-complete, even on restricted graph classes such as planar graphs of bounded bandwidth (and hence pathwidth) [15, 30, 29], split graphs [3], and bipartite graphs [23]. However, Token Sliding Reachability can be decided in polynomial time on trees [11], interval graphs [5], bipartite permutation and bipartite distance-hereditary graphs [14], line graphs [17], and claw-free graphs [6]. In the Token Sliding Optimization (TSO) problem, we are additionally given a parameter ℓ and the goal is to decide if S can be transformed to T in at most ℓ token slides. Very little is known about the optimization variant of the problem other than the hardness results that follow immediately from the reachability variant. In fact, to the best of our knowledge, the only known polynomial-time solvable instances of TSO are those restricted to interval graphs [31, 20], cographs [21], and spider trees (trees obtained by attaching paths to a central vertex) [16].

In the Token Jumping Reachability (TJR) problem, introduced by Kamiński et al. [21], we drop the restriction that the token should move along an edge of G and instead we allow it to move to any unoccupied vertex of G provided it does not break the independence of the set of tokens. That is, a single reconfiguration step consists of first removing a token on some vertex u and then immediately adding it back on any other unoccupied vertex v, as long as no two tokens become adjacent. The token is said to *jump* from vertex u to vertex v. Token Jumping Reachability is also PSPACE-complete on planar graphs of bounded bandwidth [15, 30, 29]. Lokshtanov and Mouawad [23] showed that, unlike Token Sliding Reachability, which is PSPACE-complete on bipartite graphs, the Token Jumping Reachability problem becomes NP-complete on bipartite graphs. On the positive side, it is "easy" to show that Token Jumping Reachability can be decided in polynomial-time on trees (and even on split/chordal graphs) since we can simply jump tokens to leaves (resp. vertices that only appear in the bag of a leaf in the clique tree) to transform one independent set into another. In the Token Jumping Optimization (TJO) problem, we are additionally given a parameter ℓ and the goal is to decide if S can be transformed to T in at most ℓ token jumps. To the best of our knowledge, the only known polynomial-time solvable instances of TJO are those restricted to even-hole-free graphs [21, 26].

In this paper we focus on the parameterized complexity of the aforementioned problems with respect to parameters k and ℓ and when restricted to sparse classes of graphs. Given an NP-hard or PSPACE-hard problem, *parameterized complexity* [13] allows us to refine the notion of hardness; does the hardness come from the whole instance or from a small parameter? A problem Π is *FPT (fixed-parameter tractable) parameterized by k* if one can solve it in time $f(k) \cdot poly(n)$, for some computable function f (sometimes called FPT-time). In other words, the combinatorial explosion can be restricted to the parameter k. In the rest

of the paper, we mainly consider parameters k (the number of tokens) and ℓ (the number of reconfiguration steps). TSO and TJO are known to be W[1]-hard (and XNL-complete [4]) parameterized by $k+\ell$ on general graphs [8]. TSR and TJR are known to be W[1]-hard (and XL-complete [4]) parameterized by k on general graphs [24] . When we restrict our attention to sparse classes of graphs, TSO and TJO are known to be fixed-parameter tractable when parameterized by ℓ on nowhere dense classes of graphs [26]. TJR and TJO are known to be fixed-parameter tractable parameterized by k on graphs of bounded degree [19]. For TJR, the problem becomes fixed-parameter tractable parameterized by k on biclique-free classes of graphs [7]. Finally, for TSR, the problem becomes fixed-parameter tractable parameterized by k on planar graphs, chordal graphs of bounded clique number, and graphs of bounded degree [2]. We refer the reader to the recent survey by Bousquet et al. [8] for more background on the parameterized complexity of these problems.

Given that TSO and TJO are fixed-parameter tractable when parameterized by ℓ on nowhere dense classes of graphs, it is natural to ask whether this result can be extended beyond nowhere dense graphs to biclique-free graphs. Even simpler, can we show that TSO and TJO remain fixed-parameter tractable when parameterized by ℓ on graph of bounded degeneracy? Recall that any degenerate or nowhere dense class of graphs is a biclique-free class, but not vice versa. Motivated by these questions, we show the following:

- Both problems are fixed-parameter tractable for parameter $k+\ell+d$ on d-degenerate graphs;
- Both problems are fixed-parameter tractable for parameter $|N|+k+\ell+d$ on graphs having a modulator N whose deletion leaves a d-degenerate graph (assuming N is given as part of the input); and
- Both problems are fixed-parameter tractable for parameter $|M|+\ell+\Delta$ on graphs having a modulator M whose deletion leaves a graph of maximum degree Δ.
- We complement these result by showing that for parameter ℓ alone both problems become W[1]-hard already on 2-degenerate graphs.

In fact, our hardness reductions construct 2-degenerate graphs which can be partitioned into two sets V_1 and V_2, where V_1 is an independent set and every vertex in V_2 has constant degree in the graph. Hence, our positive result for parameter $|M|+\ell+\Delta$ shows that when $|M|$ is part of our parameter we can drop k and still obtain fixed-parameter tractable algorithms; and when $|M|$ (and k) is not part of the parameter the problem is W[1]-hard.

Most of our positive results make use of the notion of independence covering families introduced by Lokshtanov et al. [25], which we believe could be of independent interest for the reconfiguration of independent sets. Let us start by formally defining such families and the various algorithms for extracting them on different graph classes.

▶ **Definition 1.1** ([25]). *For a graph G and $k \geq 1$, a family of independent sets of G is called an* independence covering family for (G,k)*, denoted by $\mathcal{F}(G,k)$, if for any independent set I in G of size at most k, there exists $J \in \mathcal{F}(G,k)$ such that $I \subseteq J$.*

▶ **Theorem 1.2** ([25]). *There is a deterministic algorithm that given a d-degenerate graph G and $k \geq 1$, runs in time $O((kd)^{O(k)} \cdot (n+m)\log n)$, and outputs an independence covering family for (G,k) of size at most $O((kd)^{O(k)} \cdot \log n)$.*

▶ **Theorem 1.3** ([25]). *Let $k,d \in \mathbb{N}$ and G be a graph. Let $S \subseteq V(G)$ such that $G-S$ is d-degenerate. There is a deterministic algorithm that given a G, S, and $k,d \in \mathbb{N}$, runs in time $O(2^{|S|} \cdot (kd)^{O(k)} \cdot 2^{O(kd)} \cdot (n+m)\log n)$, and outputs an independence covering family for (G,k) of size at most $O(2^{|S|} \cdot (kd)^{O(k)} \cdot 2^{O(kd)} \cdot \log n)$.*

▶ **Theorem 1.4** ([25]). *Let G be a graph such that $G \in \mathcal{G}$, where $\mathcal{G}$ is a class of nowhere dense graphs. There is a deterministic algorithm that given $k \geq 1$, runs in time $O(f_{\mathcal{G}}(k) \cdot (n+m)\log n)$, and outputs an independence covering family for (G,k) of size at most $O(g_{\mathcal{G}}(k) \cdot n \log n)$, where $f_{\mathcal{G}}(k)$ and $g_{\mathcal{G}}(k)$ depend on k and the class $\mathcal{G}$ but are independent of the size of the graph.*

We use Theorems 1.2 and 1.3 to design fixed-parameter tractable algorithms for parameters $k+\ell+d$ and $|N|+k+\ell+d$, respectively. Our algorithm for parameter $|M|+\ell+\Delta$ is based on the random separation technique [9]. Finally, we show that using independence covering families we can obtain a simpler and unified algorithm for the standard Token Jumping Reachability problem (a.k.a. Token Jumping) parameterized by k on both degenerate and nowhere dense classes of graphs; this is in contrast to the algorithms presented in [24]. To do so, we make use of Theorems 1.2 and 1.4. Note that the major difference between Theorems 1.2 and 1.4 is that in the former we are guaranteed a family of size at most $O((kd)^{O(k)} \cdot \log n)$ while in the latter the family is of size at least $O(g_{\mathcal{G}}(k) \cdot n \log n)$, i.e., we have an extra linear dependence on n. This difference is the reason why our algorithm for parameter $k+\ell+d$ cannot be adapted to work for nowhere dense graphs. The current complexity status of all problems considered in this work is summarized in Table 1.

Table 1 Parameterized complexity status of the reachability and optimization variants of Token Sliding and Token Jumping. Results proved in this paper are shown in bold.

	Degenerate	Nowhere dense	Biclique free
TSR parameterized by k	Open	Open	Open
TSO parameterized by k	Open	Open	Open
TSO parameterized by ℓ	**W[1]-hard**	FPT	**W[1]-hard**
TSO parameterized by $k+\ell$	**FPT**	FPT	Open
TJR parameterized by k	FPT	FPT	FPT
TJO parameterized by k	Open	Open	Open
TJO parameterized by ℓ	**W[1]-hard**	FPT	**W[1]-hard**
TJO parameterized by $k+\ell$	**FPT**	FPT	Open

2 Preliminaries

Sets and functions. We denote the set of natural numbers (including 0) by $\mathbb{N}$. For $n \in \mathbb{N}$, we use $[n]$ and $[n]_0$ to denote the sets $\{1,2,\cdots,n\}$ and $\{0,1,2,\cdots,n\}$, respectively. For a set X, we denote its power set by $2^X = \{X' \mid X' \subseteq X\}$. For a function $f : X \to Y$ and an element $y \in Y$, $f^{-1}(y)$ denotes the set $\{x \in X \mid f(x) = y\}$. For a non-empty set X, a family $\mathcal{F} \subseteq 2^X$ is a *partition* of X, if i) for each $Y \in \mathcal{F}$, $Y \neq \emptyset$, ii) for distinct $Y, Z \in \mathcal{F}$, we have $Y \cap Z = \emptyset$, and iii) $\cup_{Y \in \mathcal{F}} Y = X$. An observation that we will make use of is the following:

▶ **Proposition 2.1.** *For $k, n \in \mathbb{N}$, where $n \geq c$ for some constant c, we have $(\log n)^k \leq n + k^{2k}$.*

Graphs. Unless otherwise stated, we assume that each graph G is a simple, undirected graph with vertex set $V(G)$ and edge set $E(G)$, where $|V(G)| = n$ and $|E(G)| = m$. The *open neighborhood*, or simply *neighborhood*, of a vertex v is denoted by $N_G(v) = \{u \mid \{u,v\} \in E(G)\}$, the *closed neighborhood* by $N_G[v] = N_G(v) \cup \{v\}$. Similarly, for a set of vertices $S \subseteq V(G)$, we define $N_G(S) = \{v \mid \{u,v\} \in E(G), u \in S, v \notin S\}$ and $N_G[S] = N_G(S) \cup S$. The *degree* of a vertex is $|N_G(v)|$. We drop the subscript G when clear from context. A *subgraph*

of G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The *induced subgraph* of G with respect to $S \subseteq V(G)$ is denoted by $G[S]$; $G[S]$ has vertex set S and edge set $E(G[S]) = \{\{u, v\} \in E(G) \mid u, v \in S\}$.

3 FPT algorithm for parameter $k + \ell + d$

In this section we start by designing a fixed-parameter tractable algorithm for the TSO problem parameterized by $k+\ell+d$ on d-degenerate graphs. We then show how the algorithm can be adapted for TJO as well as for parameter $|N|+k+\ell+d$ on graphs having a modulator N whose deletion leaves a d-degenerate graph (assuming N is given as part of the input).

We let (G, S, T, k, ℓ) denote an instance of TSO, where G is d-degenerate. Moreover, we assume that we have computed in time $O((kd)^{O(k)} \cdot (n+m)\log n)$ an independence covering family $\mathcal{F}(G,k)$ for (G,k) of size at most $O((kd)^{O(k)} \cdot \log n)$ (Theorem 1.2). Without loss of generality, we assume that both S and T belong to $\mathcal{F}(G,k)$; as otherwise we can simply add them. Note that if (G, S, T, k, ℓ) is a yes-instance then there exists a sequence $\langle I_0, I_1, ..., I_\ell \rangle$ of independent sets such that for all $j \in \{0, \ldots, \ell-1\}$ the set I_j is an independent set of size k in G, $I_0 = S$, $I_\ell = T$, and I_{j+1} is obtained from I_j by a token slide. This implies that there exists a sequence $\langle J_0, J_1, ..., J_\ell \rangle$ of elements of $\mathcal{F}(G,k)$ such that $J_0 = S$, $J_\ell = T$, and for $j \in \{1, \ldots, \ell-1\}$ we have $I_j \subseteq J_j$. In what follows, we assume that we guessed a sequence $\langle J_0, J_1, ..., J_\ell \rangle$ of elements of $\mathcal{F}(G,k)$ such that $J_0 = S$ and $J_\ell = T$. Our goal now is to design an algorithm that either finds a reconfiguration sequence $\langle I_0 = S, I_1 \subseteq J_1, ..., I_{\ell-1} \subseteq J_{\ell-1}, I_\ell = T \rangle$ or determine that no such sequence exists.

We define a constraint as a pair (X, b) where $X \subseteq V(G)$ and b is a positive integer, called the budget of X. We denote a set of constraints by $C = \{(X, b), \ldots\}$. We say that the constraint (X, b) is satisfied (by Z) if $|Z \cap X| = b$, where $Z \subseteq V(G)$. We say that the set of constraints C is satisfied if all pairs $(X, b) \in C$ are satisfied. We denote a set of sets of constraints by $\mathcal{C}$. We now proceed by building sets of sets of constraints $\mathcal{C}_0$, $\mathcal{C}_1$, ..., $\mathcal{C}_\ell$ and show that for each $i \in [\ell]_0$, the following invariants are satisfied:

- **Correctness Invariant I:** If a k-sized set $Z \subseteq J_i$ satisfies at least one set of constraints in $\mathcal{C}_i$, then Z is reachable from $S = J_0$.
- **Correctness Invariant II:** For any k-sized set $Z \subseteq J_i$, if there is a reconfiguration sequence $S = I_0, I_1, I_2, \ldots, I_i = Z$, where for each $p \in [i]_0$, $I_p \subseteq J_p$, then Z satisfies at least one set of constraints in $\mathcal{C}_i$.
- **Size Invariant:** The total number of constraints at the i^{th} step is $\sum_{C \in \mathcal{C}_i} |C| \leq (i+1)!$.

At the base case, we let $\mathcal{C}_0 = \{\{(S, k)\}\}$. The correctness of the base case immediately follows from its construction. We now proceed recursively as follows. Consider $i \in [\ell]$. We assume that for each $p \in [i-1]$, we have computed $\mathcal{C}_p$ that satisfy the correctness and size invariants. Initialize $\mathcal{C}_i = \emptyset$.

For each $C \in \mathcal{C}_{i-1}$,
  For each constraint $(X, b) \in C$,

1. Initialize a constraint set $C' = \emptyset$.
2. If $b = 1$,
   a. Add $(N(X) \cap J_i, 1)$ to C'.
   b. Add $(X' \cap J_i, b')$ for all other constraints $(X', b') \in C$ to C'.
3. Else
   a. Add $(X \cap J_i, b-1)$ to C'.
   b. Add $(N(X) \cap J_i, 1)$ to C'.
   c. Add $(X' \cap J_i, b')$ for all other constraints $(X', b') \in C$ to C'.
4. Add C' to $\mathcal{C}_i$.

▶ **Lemma 3.1.** *For every $C \in \mathcal{C}_i$, we have $\bigcup_{(X,b)\in C} X \subseteq J_i$, $\sum_{(X,b)\in C} b = k$, and all the vertex subsets which are part of the constraints in C are pairwise disjoint.*

▶ **Lemma 3.2** (Size Invariant)**.** *The total number of constraints at the i^{th} step is $c_i = \sum_{C\in\mathcal{C}_i} |C| \leq (i+1)!$. Therefore, $c_\ell \leq (\ell+1)!$.*

Proof. Let $c_i = \sum_{C\in\mathcal{C}_i} |C|$. We have $c_0 = 1$ from the base case of the algorithm. At each step the number of constraints in a set of constraints added increases by at most 1. For $i = 0$, we have only one constraint in $\{(S,k)\} \in \mathcal{C}_0$. Therefore, at the i^{th} step, the maximum number of constraints in any set contained in $\mathcal{C}_i$ is at most $i+1$. In the i^{th} recursive step of the algorithm, we add a new set of constraints C' corresponding to each constraint (X,b) contained in some member of $\mathcal{C}_{i-1}$. So, we get the following recursive relation: $|\mathcal{C}_i| = c_{i-1}$. Using the fact that all members of $\mathcal{C}_i$ contain at most $i+1$ constraints, we get that $c_i = \sum_{C\in\mathcal{C}_i} |C| \leq (i+1)|\mathcal{C}_i| = (i+1)c_{i-1}$. Solving the recurrence, we get $c_i \leq (i+1)!$. Therefore, $c_\ell \leq (\ell+1)!$. ◀

▶ **Lemma 3.3** (Correctness Invariant I)**.** *If a k-sized independent set $Z \subseteq J_i$ satisfies at least one set of constraints in $\mathcal{C}_i$, then Z is reachable from S.*

Proof. We use induction to prove the lemma. For $i = 0$, we have $\mathcal{C}_0 = \{\{(S,k)\}\}$. For $C = \{(S,k)\}$, we can see that the lemma holds trivially. For the inductive step, we assume that the lemma holds true for $i-1$, and prove it for i. Let $Z \subseteq J_i$ and $|Z| = k$ such that it satisfies some set of constraints $C \in \mathcal{C}_i$. Let (X,b) be the constraint in $C' \in \mathcal{C}_{i-1}$ which produces this set of constraints C in the i^{th} recursive step of the algorithm.

Let v^* be the vertex in $Z \cap (N(X) \cap J_i)$. Let u^* be a vertex in X sharing an edge with v^*. Take $Z' = (Z \cup \{u^*\}) \setminus \{v^*\}$. It can be seen that $|Z'| = k$ and Z can be obtained from Z' by sliding one token. Since Z satisfies the set of constraints C, we have:

1. $|Z \cap (N(X) \cap J_i)| = 1$
2. $|Z \cap (X \cap J_i)| = |Z \cap X| = b - 1$ (0 if $b = 1$)
3. $|Z \cap (X' \cap J_i)| = |Z \cap X'| = b'$ for all other constraints $(X', b') \in C'$

The way we construct Z', it must satisfy the following conditions:

1. $|Z' \cap X| = b \geq 1$ (since u^* is included in Z'); and
2. $|Z' \cap X'| = b'$ for all other constraints $(X', b') \in C'$.

It can be clearly seen that Z' satisfies the set of constraints $C' \in \mathcal{C}_{i-1}$. So, $|Z' \cap (\cup_{(X',b')\in C'} X')| = \sum_{(X',b')\in C'} |Z' \cap X'| = \sum_{(X',b')\in C'} b' = k$, where the first equality follows from the fact that all X' such that $(X', b') \in C'$ are pairwise disjoint by Lemma 3.1 and the last equality follows from Lemma 3.1. Therefore, $Z' \subseteq \cup_{(X',b')\in C'} X' \subseteq J_{i-1}$ again by Lemma 3.1.

Thus, Z' is a k-sized subset of J_{i-1} and satisfies at least one set of constraints in $\mathcal{C}_{i-1}$. By the induction hypothesis, Z' is reachable from S. Now, since Z is reachable from Z', Z is also reachable from S. ◀

▶ **Lemma 3.4** (Correctness Invariant II)**.** *For any k-sized independent set $Z \subseteq J_i$, if there is a reconfiguration sequence $S = I'_0, I'_1, I'_2, \ldots, I'_i = Z$, where for each $p \in [i]_0$, $I'_p \subseteq J_p$, then Z satisfies at least one set of constraints in $\mathcal{C}_i$.*

Proof. We use induction to prove the lemma. For $i = 0$, we have $\mathcal{C}_0 = \{\{(S,k)\}\}$. The set S satisfies the set of constraints $C = \{(S,k)\}$ and the lemma holds.

We now assume that the lemma holds true for $i-1$, and prove it for i. Let $C \in \mathcal{C}_{i-1}$ be the set of constraints that I'_{i-1} satisfies. So, $|I'_{i-1} \cap (\cup_{(X,b)\in C} X)| = \sum_{(X,b)\in C} |I'_{i-1} \cap X| = \sum_{(X,b)\in C} b = k$, where the first equality follows from the fact that all X such that $(X,b) \in C$

are pairwise disjoint by Lemma 3.1 and the last equality follows again from Lemma 3.1. Since $|I'_{i-1}| = k$, we have $I'_{i-1} \subseteq \cup_{(X,b)\in C} X$. In the i^{th} step of the reconfiguration sequence, we slide a token from I'_{i-1} to I'_i, i.e. from some set X such that $(X, b) \in C$ to its open neighbourhood. Consider the set of constraints $C' \in \mathcal{C}_i$ obtained by splitting the constraint (X, b) in the i^{th} recursive step of the algorithm. We will show that I'_i satisfies C'. Since I'_{i-1} satisfies the set of constraints C, we have $|I'_{i-1} \cap X| = b$ for all other constraints $(X, b) \in C$. So, in the i^{th} step we have $|I'_i \cap (N(X) \cap J_i)| = |I'_i \cap N(X)| = 1$, where the first equality is because $I'_i \subseteq J_i$ and the second equality is because I'_i is obtained from I'_{i-1} by sliding a token from X to its neighbourhood. We have $|I'_i \cap (X \cap J_i)| = |I'_i \cap X| = |I'_{i-1} \cap X| - 1 = b - 1$ as one token is moved from X. When $b = 1$, we get $I'_i \cap X = \emptyset$ and this budget constraint is not included in the i^{th} recursive step of the algorithm. For all other $(X', b') \in C$, we have $|I'_i \cap (X' \cap J_i)| = |I'_i \cap X'| = |I'_{i-1} \cap X'| = b'$ as none of the tokens in any X' are moved in the i^{th} step of the reconfiguration sequence. Therefore, $I'_i = Z$ satisfies all the constraints in C', as needed. ◀

We are now ready to prove our first main theorem.

▶ **Theorem 3.5.** *Token Sliding Optimization is fixed-parameter tractable parameterized by $k + \ell + d$ where d denotes the degeneracy of the graph.*

Proof. Let (G, S, T, k, ℓ) denote an instance of TSO, where G is d-degenerate. We first computed in time $O((kd)^{O(k)} \cdot (n + m) \log n)$ an independence covering family $\mathcal{F}(G, k)$ for (G, k) of size at most $O((kd)^{O(k)} \cdot \log n)$ (by Theorem 1.2). We then add S and T to $\mathcal{F}(G, k)$ (in case they do not already belong to $\mathcal{F}(G, k)$). Next, we "guess" a (iterate over every) sequence $\langle J_0, J_1, ..., J_\ell \rangle$ of elements of $\mathcal{F}(G, k)$ such that $J_0 = S$, $J_\ell = T$. Note that this guessing can be accomplished in time $O(((kd)^{O(k)} \cdot \log n)^{\ell+1})$, which by Proposition 2.1 is still FPT-time. Finally, we compute $\mathcal{C}_0$, $\mathcal{C}_1$, ..., $\mathcal{C}_\ell$, which by Lemma 3.2 can also be done in FPT-time. To conclude, we simply need to check whether T satisfies at least one set of constraints in $\mathcal{C}_\ell$. The correctness of the algorithm follows from Lemma 3.3 and 3.4. ◀

▶ **Theorem 3.6.** *Token Sliding Optimization is fixed-parameter tractable parameterized by $|N| + k + \ell + d$ on graphs having a modulator N whose deletion leaves a d-degenerate graph (assuming N is given as part of the input).*

Proof. We proceed exactly as in the proof of Theorem 3.5 but we invoke Theorem 1.3 instead of Theorem 1.2. ◀

We conlude this section by showing how we can adapt the previous two results for the Token Jumping Optimization problem. To allow tokens to jump to arbitrary vertices of the graph we only need to slightly modify our construction of the sets $\mathcal{C}_1$, ..., $\mathcal{C}_\ell$ to obtain the following:

▶ **Theorem 3.7.** *Token Jumping Optimization is fixed-parameter tractable parameterized by $k + \ell + d$ where d denotes the degeneracy of the graph and fixed-parameter tractable parameterized by $|N| + k + \ell + d$ on graphs having a modulator N whose deletion leaves a d-degenerate graph (assuming N is given as part of the input).*

4 FPT algorithm for parameter $|M| + \ell + \Delta$

In this section, we prove that TSO and TJO are fixed-parameter tractable parameterized by $|M| + \ell + \Delta$. Recall that an instance of either problem is denoted by (G, S, T, k, ℓ) where $V(G)$ can be partitioned into H and M and every vertex in H has degree at most Δ in G.

Our algorithm is randomized and based on a variant of the color-coding technique [1] that is particularly useful in designing parameterized algorithms on graphs of bounded degree. The technique, known in the literature as random separation [9], boils down to a simple, but fruitful observation that in some cases, if we randomly color the vertex set of a graph using two colors, the solution or vertices we are looking for are appropriately colored with high probability. In our case, we want to make sure that the set of vertices involved in token slides or jumps gets highlighted. We note that our algorithm is an adaptation of the algorithm of Mouawad et al. [26] and it can easily be derandomized using standard techniques [10].

We start with an instance $(G = (H, M, E), S, T, k, \ell)$ of TSO; the algorithm is identical for TJO. We color independently every vertex of H using one of two colors, say red and blue (denoted by $\mathcal{R}$ and $\mathcal{B}$), with probability $\frac{1}{2}$. We let $\chi : H \to \{\mathcal{R}, \mathcal{B}\}$ denote the resulting random coloring. Suppose that (G, S, T, k, ℓ) is a yes-instance, and let σ denote a reconfiguration sequence from S to T of length at most ℓ. We say a vertex $v \in H$ is *touched* in σ whenever a token slides from a neighbor of v to v or from v to some neighbor of v. We let $V(\sigma)$ denote the set of vertices touched by σ. We say that the coloring χ is successful if both of the following conditions hold:

- Every vertex in $V(\sigma) \cap H$ is colored red; and
- Every vertex in $N_H(V(\sigma) \cap H)$ is colored blue.

Observe that $V(\sigma) \cap H$ and $N_H(V(\sigma) \cap H)$ are disjoint. Therefore, the two aforementioned conditions are independent. Moreover, since the maximum degree of $G[H]$ is Δ, we have $|V(\sigma) \cap H| + |N_H(V(\sigma) \cap H)| \leq 2\ell\Delta$. Consequently, the probability that χ is successful is at least:

$$\frac{1}{2^{|V(\sigma)\cap H|+|N_H(V(\sigma)\cap H)|}} \geq \frac{1}{2^{2\ell\Delta}} = \frac{1}{4^{\ell\Delta}}.$$

Let $H_{\mathcal{R}}$ denote the set of vertices of H colored red and $H_{\mathcal{B}}$ denote the set of vertices of H colored blue. Moreover, we let $C_1, \ldots, C_q$ denote the set of connected components of $G[H_{\mathcal{R}}]$. The main observation now is the following:

▶ **Lemma 4.1.** *If χ is successful then $V(\sigma)$ has a non-empty intersection with at most 2ℓ connected components of $G[H_{\mathcal{R}}]$, and each one of those components consists of at most 2ℓ vertices.*

Given an instance $(G = (H, M, E), S, T, k, \ell)$ of TSO and a coloring χ of H, we know from Lemma 4.1 that when χ is successful every connected component of $G[H_{\mathcal{R}}]$ consists of at most 2ℓ vertices. We now construct a new (reduced) instance (G', S', T', k', ℓ) of TSO. We first guess the vertices of M that will be touched in a solution and we let M' denote this set. Note that this guessing can be accomplished in time 2^M-time. Starting from a copy of G we proceed as follows:

- If there exists $v \in (S \cap T) \cap H$ and v is colored blue then we delete v and its neighbors from the graph;
- If there exists $v \in (S \cap T) \cap (M \setminus M')$ then we delete v and its neighbors from the graph;
- If there exists $v \in (S \cap T) \cap H$, v is colored red, and v belongs to a red component C of $G[H_{\mathcal{R}}]$ such that $|V(C)| > 2\ell$ then we delete v and its neighbors from the graph;
- If there exists a blue vertex v which is not in $S \cap T$ then we delete v from the graph;
- If there exists a red vertex v which is not in $S \cap T$ and v belongs to a red component C of $G[H_{\mathcal{R}}]$ such that $|V(C)| > 2\ell$ then we delete v from the graph.

We adjust S, T, and k appropriately to obtain the new equivalent instance (G', S', T', k', ℓ). Note that in this new instance (assuming a successful coloring) no vertices are colored blue and (assuming a correct guess) all vertices of M' will be touched in a solution. In other

words, G' can be partitioned into M' and H' where H' consists of (an unbounded number of) connected components each consisting of at most 2ℓ vertices. Note that when the number of connected components is constant then we are done since we can solve the problem via brute-force. In other words, we can simply enumerate all possible sequences of length at most ℓ and make sure that at least one of them is the required reconfiguration sequence from S' to T'. This brute-force testing can be accomplished in time $2^{\mathcal{O}(\ell \log \ell)} \cdot n^{\mathcal{O}(1)}$.

Let us now consider the general case when the number of components is not necessarily bounded. We say a component C of $G' - M'$ is *important* if $V(C) \cap ((S' \setminus T') \cup (T' \setminus S')) \neq \emptyset$. There are at most 2ℓ important components. Hence, we only need to bound the number of *unimportant* components. To that end, we partition the unimportant components of $G' - M'$ into equivalence classes with respect to the relation $\simeq$. For two graphs G_1, G_2 and two sets $X_1 \subseteq V(G)$, $X_2 \subseteq V(G_2)$, we say that (G_1, X_1) and (G_2, X_2) are *isomorphic* if the graphs G_1 and G_2 are isomorphic where vertices of X_1 and X_2 are now assigned the same color. Formally, a *c-colored graph* G is a tuple $(V, E, \mathcal{K})$ such that $\mathcal{K} = \{K_1, \ldots, K_c\}$ is a collection of subsets of $V(G)$ where each K_i is called a *color set*. Two colored graphs $G_1 = (V_1, E_1, \mathcal{K}_1)$ and $G_2 = (V_2, E_2, \mathcal{K}_2)$ are isomorphic if there is a *color-preserving isomorphism* $f : V_1(G_1) \to V_2(G_2)$ such that:

- for all $u, v \in V_1(G_1)$, $\{u, v\} \in E_1(G_1)$ if and only if $\{f(u), f(v)\} \in E_2(G_2)$; and
- for all $v \in V_1(G_1)$ and $K_i^1 \in \mathcal{K}_1$, $v \in K_i^1$ if and only if $f(v) \in K_i^2$.

Hence, (G_1, X_1) and (G_2, X_2) are isomorphic if the colored graphs $G_1 = (V_1, E_1, \{X_1\})$ and $G_2 = (V_2, E_2, \{X_2\})$ are isomorphic. Let C_1 and C_2 be two components in $G' - M'$ and let N_1 and N_2 be their respective neighborhoods in M'. We say C_1 and C_2 are *equivalent*, i.e., $C_1 \simeq C_2$, whenever $N_1 = N_2 = N$ and $(G[V(C_1) \cup N], V(C_1) \cap S' \cap T')$ is isomorphic to $(G[V(C_2) \cup N], V(C_2) \cap S' \cap T')$ by an isomorphism that fixes N point-wise.

▶ **Lemma 4.2.** *The total number of 2-colored graphs with at most 2ℓ vertices is at most $2^{\mathcal{O}(\ell^2)}$, and therefore, the equivalence relation $\simeq$ has at most $2^{\mathcal{O}(\ell^2)}$ equivalence classes.*

Assume that some equivalence class contains more than 2ℓ unimportant components. We claim that retaining only 2ℓ of them is enough. To see why, it is enough to note that $V(\sigma)$ intersects with at most 2ℓ of those components; they are all equivalent. Putting it all together, we know that we have at most $2^{\mathcal{O}(\ell^2)}$ equivalence classes, each with at most 2ℓ components, and each component is of size at most 2ℓ. Hence, we can guess the sequence from S' to T' in time $2^{\mathcal{O}(\ell^3 \log \ell)} \cdot n^{\mathcal{O}(1)}$ (testing whether two graphs with 2ℓ vertices are isomorphic can be accomplished naively in time $2^{\ell \log \ell}$).

We have proven that the probability that χ is successful is at least $4^{-\ell\Delta}$. Hence, to obtain a Monte Carlo algorithm with false negatives, we repeat the above procedure $4^{\ell\Delta}$ times and obtain the following result:

▶ **Theorem 4.3.** *There exists a one-sided error Monte Carlo algorithm with false negatives that solves TSO and TJO parameterized by $|M| + \ell + \Delta$ in time $\mathcal{O}(2^M \cdot 4^{\ell\Delta} \cdot 2^{\mathcal{O}(\ell^3 \log \ell)} \cdot n^{\mathcal{O}(1)})$.*

5 Hardness of TSO parameterized by ℓ on 2-degenerate graphs

In the Multicolored Clique problem, we are given an input graph G whose vertices are colored with k colors and the goal is to find a clique containing one vertex from each color. We show that TSO parameterized by ℓ is W[1]-hard on 2-degenerate graphs via a reduction from Multicolored Clique, known to be W[1]-hard.

We construct an instance $(G', S, T, \kappa, \ell = 8\binom{k}{2} + 2k)$ of TSO from an instance of Multicolored Clique denoted by $(G, k, (V_1, V_2, \ldots, V_k))$, where, w.l.o.g., we assume that there are no edges between two vertices of G of the same color.

Construction of G'. We subdivide all the edges in G. Let the vertex set of G be V. All the vertices corresponding to the edges in G are partitioned into $\binom{k}{2}$ sets of the form $\mathcal{E}_{ij}$, where $i = \{1, 2, \ldots, k\}$ and $j = \{1, 2, \ldots, k\}$ and $i \neq j$, such that $\mathcal{E}_{ij}$ contains all the vertices corresponding to the edges in G having one incident vertex of color i and the other incident vertex of color j. Let the union of all the sets $\mathcal{E}_{ij}$ be denoted by E.

We introduce two independent sets X and Y, each of size $\binom{k}{2}$. Let us label the vertices in X from 1 to $\binom{k}{2}$ and the sets $\mathcal{E}_{ij}$ from 1 to $\binom{k}{2}$. We add edges between vertex with label b in X and all the vertices in the $\mathcal{E}_{ij}$ with label b. Similarly, we label the vertices of Y and add edges from each vertex in Y to all the vertices in the $\mathcal{E}_{ij}$ having the same label. Each of these edges is further subdivided three times. Let the vertices on the subdivided edges from X to E, which are neither adjacent to some vertex in X nor E be denoted by U_1, and the vertices on the subdivided edges from Y to E, which are neither adjacent to some vertex in Y nor E be denoted by U_2. We take $U = U_1 \cup U_2$.

We also add a vertex corresponding to each vertex in V and add an edge between the two. Let this set of vertices be Z. The induced subgraph of G' having $V \cup Z$ as its vertex set forms a perfect matching. Our initial independent set $S = V \cup X \cup U$ and our target independent set $T = V \cup Y \cup U$. Note that $|S| = |T| = n + \binom{k}{2} + |U| = \kappa$. We set $\ell = 8\binom{k}{2} + 2k$.

▶ **Lemma 5.1.** *The graph G' is 2-degenerate.*

Proof. Recall that a graph G' is 2-degenerate if every induced subgraph H of G' has a vertex of degree at most 2. Consider any induced subgraph H of G'. If H contains a vertex of Z or a vertex from the subdivided edges from $X \cup Y$ to E then we are done; as those vertices have degree at most two in G'. Otherwise, we know that H either contains an isolated vertex from $X \cup Y$ or a degree-two vertex from E, as needed. ◀

▶ **Lemma 5.2.** *If $(G, k, (V_1, V_2, \ldots, V_k))$ is a yes-instance of Multicolored Clique then there is a reconfiguration sequence of length at most ℓ from S to T in G'.*

Proof. Let the solution to the Multicolored Clique instance be $\{v_1, v_2, \ldots, v_k\} \subseteq V$. Consider the following reconfiguration sequence from S to T:

1. Slide each token on v_i to its matched neighbour in Z; for a total of k slides.
2. Since the vertices $\{v_1, v_2, \ldots, v_k\}$ form a clique in G, there are $\binom{k}{2}$ edges, each having distinct pair of colors on their incident vertices. So in G', all the vertices corresponding to the edges of the clique lie in distinct partitions $\mathcal{E}_{ij}$. We slide all the tokens from X to Y using these $\binom{k}{2}$ vertices. Consider the path from a vertex $v_x \in X$ to a vertex $v_y \in Y$, passing through one of these $\binom{k}{2}$ vertices, say v_i where $i \in [k]$. This path contains a vertex $u_1 \in U_1$ and a vertex $u_2 \in U_2$. Slide the token on u_2 to v_y (2 slides), the token on u_1 to u_2 through v_i (4 slides), and the token on v_x to u_1 along this path (2 slides); for a total of $8\binom{k}{2}$ slides.
3. Finally we slide the tokens in Z back to V; for a total of k slides.

The length of the reconfiguration sequence is $8\binom{k}{2} + 2k$. This completes the proof. ◀

▶ **Lemma 5.3.** *If there is a reconfiguration sequence of length at most ℓ from S to T in G' then $(G, k, (V_1, V_2, \ldots, V_k))$ is a yes-instance of Multicolored Clique.*

The combination of Lemmas 5.2 and 5.3 give us the following:

▶ **Theorem 5.4.** *Token Sliding Optimization parameterized by ℓ is W[1]-hard on 2-degenerate graphs.*

6 Hardness of TJO parameterized by ℓ on 2-degenerate graphs

We now show that TJO parameterized by ℓ is W[1]-hard on 2-degenerate graphs via a reduction from the Clique problem, known to be W[1]-hard. We construct an instance $(G', S, T, \kappa, 2\binom{k}{2}+\binom{k}{2}^2+2k)$ of TJO starting from a Clique instance (G, k). The construction is quite similar to that of the sliding variant but with some adaptation to account for the possibility of tokens jumping anywhere in the graph.

Construction of G'. We subdivide all the edges in G. Let the vertex set of G be V. We let the set of vertices in G' corresponding to the edges be denoted by E. We introduce a biclique with parts L and R, each of size $\binom{k}{2}$. Next, we subdivide all the edges of the biclique twice. Let this entire set of vertices, i.e. $L \cup N(L) \cup N(R) \cup R$ be denoted by X. The vertices in X do not have edges with those in E or V. We also add a vertex corresponding to each vertex in V and add an edge between the two. Let this set of vertices be Z. The induced subgraph of G' having $V \cup Z$ as its vertex set forms a perfect matching. Our initial independent set $S = V \cup L \cup N(R)$ and our target independent set $T = V \cup R \cup N(L)$. Note that $|S| = |T| = \kappa$. We let $\ell = 2\binom{k}{2} + \binom{k}{2}^2 + 2k$.

▶ **Lemma 6.1.** *The graph G' is 2-degenerate.*

Proof. Consider any induced subgraph H of G'. If H contains a vertex of Z or a vertex from $N(L) \cup N(R)$ then we are done; as those vertices have degree at most two in G'. Otherwise, we know that H either contains an isolated vertex from $L \cup R$ or a degree-two vertex from E, as needed. ◀

▶ **Lemma 6.2.** *If (G, k) is a yes-instance of Clique then there is a reconfiguration sequence of length at most ℓ from S to T in G'.*

Proof. Let the solution to the Clique instance be $\{v_1, v_2, \ldots, v_k\} \subseteq V$. Consider the following reconfiguration sequence from S to T:

1. Jump each token on v_i to its matched neighbour in Z; for a total of k jumps.
2. Since the vertices $\{v_1, v_2, \ldots, v_k\}$ form a clique in G, there are $\binom{k}{2}$ edges in the subgraph induced on those vertices. Let the set of corresponding vertices in E be E_C. We jump all the $\binom{k}{2}$ tokens from L to the vertices in E_C; for a total of $\binom{k}{2}$ jumps.
3. Jump all the tokens in $N(R)$ to their adjacent vertex in $N(L)$; for a total of $\binom{k}{2}^2$ jumps.
4. Now jump all the tokens in E_C to R; for a total of $\binom{k}{2}$ jumps.
5. Finally we jump the tokens in Z back to V; for a total of k jumps.

The length of the reconfiguration sequence is $2\binom{k}{2} + \binom{k}{2}^2 + 2k$. This completes the proof. ◀

▶ **Lemma 6.3.** *If there is a reconfiguration sequence of length at most ℓ from S to T in G' then (G, k) is a yes-instance of Clique.*

The combination of Lemmas 6.2 and 6.3 give us the follwing:

▶ **Theorem 6.4.** *Token Jumping Optimization parameterized by ℓ is W[1]-hard on 2-degenerate graphs.*

7 FPT algorithm for Token Jumping Reachability parameterized by k

We propose a generalized scheme for solving Token Jumping Reachability parameterized by k on graphs having a small k-independence covering family, i.e., a family of size $\mathcal{O}(f(k) \cdot \text{poly}(n))$. Degenerate and nowhere dense graphs admit such independence covering families as shown in [25].

We remove all the sets in the covering family of size less than k. We find out if the independent sets S and T are a part of the independence covering family. If not, we add them to the family. Let the size of the resulting k-independence covering family $\mathcal{F}(G,k)$ be q. For denoting an independent set in the family, we will use capital letters like, $X, Y (\subseteq V(G))$. We construct a graph $\mathcal{G}$ with q vertices corresponding to the q sets in the family. Consider two independent sets I and I' in $\mathcal{F}(G,k)$. We add an edge between the vertices i and i' in $\mathcal{G}$ if and only if $|I \cap I'| \geq k-1$. Note that for any two k-sized independent sets J and J' we can find a trivial reconfiguration sequence from J to J' if both of them are contained in some I in $\mathcal{F}(G,k)$.

In the algorithm, we find out if the vertices i_s and i_t in $\mathcal{G}$ are in the same connected component. If yes, then we know that S is reachable from T from the construction of $\mathcal{G}$. Otherwise no reconfiguration sequence from S to T exists.

▶ **Lemma 7.1.** *If there exists a path from i_s to i_t in $\mathcal{G}$ then there is a reconfiguration sequence from S to T in G.*

Proof. Let $i_s = i_0, i_1, i_2, \ldots, i_\ell = i_t$ be the path from i_s to i_t. We start the reconfiguration sequence with S. For each pair of vertices i_j and i_{j+1} in the path, we have $|I_j \cap I_{j+1}| \geq k-1$ according to the construction. Now, let $X_j \subseteq I_j$ be a k-sized independent set in the reconfiguration sequence and $Y_j \subseteq I_j \cap I_{j+1}$ be a $(k-1)$-sized set. Let u_j be a vertex in X_j. We can obtain a k-sized independent set $Z_j = Y_j \cup \{u_j\}$ from X_j by at most $k-1$ token jumps. Next we jump the token on u_j to a vertex in $I_{j+1} \setminus Y_j$ to obtain a k-sized independent set $X_{j+1} \subseteq I_{j+1}$. This gives us a reconfiguration sequence from S to T, as needed. ◀

▶ **Lemma 7.2.** *If there is a reconfiguration sequence $S = I_0, I_1, I_2, \ldots, I_\ell = T$ then there exists a path from i_s to i_t in $\mathcal{G}$.*

Proof. Let $I'_1, I'_2, \ldots, I'_{\ell-1}$ be the sets in the covering family such that $I_i \subseteq I'_i$ for $i \in [\ell-1]$. Since $|I_i \cap I_{i+1}| = k-1$, we have $|I'_i \cap I'_{i+1}| \geq k-1$. If I'_i and I'_{i+1} are the same set, then they correspond to the same vertex in $\mathcal{G}$. Otherwise, they are connected by an edge according to the construction of $\mathcal{G}$. We start from the vertex i_s and following the reconfiguration sequence, we reach i_t. This gives us a walk from i_s to i_t and a walk contains a path, as needed. ◀

The combination of Lemmas 7.1 and 7.2 give us the following:

▶ **Theorem 7.3.** *Token Jumping Reachability parameterized by k is fixed-parameter tractable on any graph class $\mathcal{C}$ for which we can, given any n-vertex graph $G \in \mathcal{C}$, compute a k-independence covering family $\mathcal{F}(G,k)$ of size $\mathcal{O}(f(k) \cdot n^{\mathcal{O}(1)})$ in time $\mathcal{O}(g(k) \cdot n^{\mathcal{O}(1)})$, where f and g are computable functions.*

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In *Encyclopedia of Algorithms*, pages 335–338. Springer, 2016. doi:10.1007/978-1-4939-2864-4_76.

2 Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic Token Sliding. *CoRR*, abs/2204.05549, 2022. doi:10.48550/arXiv.2204.05549.

3 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021. doi:10.1007/s00224-020-09967-8.

4 Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized Complexities of Dominating and Independent Set Reconfiguration. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2021.9.

5 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science – 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, volume 10520 of *Lecture Notes in Computer Science*, pages 127–139. Springer, 2017. doi:10.1007/978-3-319-68705-6_10.

6 Paul S. Bonsma, Marcin Kaminski, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory – SWAT 2014 – 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2014. doi:10.1007/978-3-319-08404-6_8.

7 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token jumping in minor-closed classes. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory – 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, volume 10472 of *Lecture Notes in Computer Science*, pages 136–149. Springer, 2017. doi:10.1007/978-3-662-55751-8_12.

8 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *CoRR*, abs/2204.10526, 2022. doi:10.48550/arXiv.2204.10526.

9 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006. doi:10.1007/11847250_22.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Polynomial-time algorithm for sliding tokens on trees. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation – 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 389–400. Springer, 2014. doi:10.1007/978-3-319-13075-0_31.

12 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

13 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

14 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In Khaled M. Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation – 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, volume 9472 of *Lecture Notes in Computer Science*, pages 237–247. Springer, 2015. doi:10.1007/978-3-662-48971-0_21.

15 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

16 Duc A. Hoang, Amanj Khorramian, and Ryuhei Uehara. Shortest reconfiguration sequence for sliding tokens on spiders. In Pinar Heggernes, editor, *Algorithms and Complexity – 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 2019. doi:10.1007/978-3-030-17402-6_22.

17 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

18 Takehiro Ito, Marcin Kaminski, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In T. V. Gopal, Manindra Agrawal, Angsheng Li, and S. Barry Cooper, editors, *Theory and Applications of Models of Computation – 11th Annual Conference, TAMC 2014, Chennai, India, April 11-13, 2014. Proceedings*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer, 2014. doi:10.1007/978-3-319-06089-7_24.

19 Takehiro Ito, Marcin Jakub Kaminski, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. Parameterized complexity of independent set reconfiguration problems. *Discret. Appl. Math.*, 283:336–345, 2020. doi:10.1016/j.dam.2020.01.022.

20 Takehiro Ito and Yota Otachi. Reconfiguration of colorable sets in classes of perfect graphs. *Theor. Comput. Sci.*, 772:111–122, 2019. doi:10.1016/j.tcs.2018.11.024.

21 Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

22 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

23 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.

24 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *J. Comput. Syst. Sci.*, 95:122–131, 2018. doi:10.1016/j.jcss.2018.02.004.

25 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Covering small independent sets and separators with applications to parameterized algorithms. *ACM Trans. Algorithms*, 16(3):32:1–32:31, 2020. doi:10.1145/3379698.

26 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Sebastian Siebertz. Vertex cover reconfiguration and beyond. *Algorithms*, 11(2):20, 2018. doi:10.3390/a11020020.

27 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

28 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

29 Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 282–293. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.282.

30 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.

31 Takeshi Yamada and Ryuhei Uehara. Shortest reconfiguration of sliding tokens on subclasses of interval graphs. *Theor. Comput. Sci.*, 863:53–68, 2021. doi:10.1016/j.tcs.2021.02.019.

On Graphs Coverable by k Shortest Paths

Maël Dumas
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Florent Foucaud
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Anthony Perez
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Ioan Todinca
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Abstract

We show that if the edges or vertices of an undirected graph G can be covered by k shortest paths, then the pathwidth of G is upper-bounded by a function of k. As a corollary, we prove that the problem Isometric Path Cover with Terminals (which, given a graph G and a set of k pairs of vertices called *terminals*, asks whether G can be covered by k shortest paths, each joining a pair of terminals) is FPT with respect to the number of terminals. The same holds for the similar problem Strong Geodetic Set with Terminals (which, given a graph G and a set of k terminals, asks whether there exist $\binom{k}{2}$ shortest paths, each joining a distinct pair of terminals such that these paths cover G). Moreover, this implies that the related problems Isometric Path Cover and Strong Geodetic Set (defined similarly but where the set of terminals is not part of the input) are in XP with respect to parameter k.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Shortest paths, covering problems, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.40

Funding *Florent Foucaud*: This author was financed by the IFCAM project "Applications of graph homomorphisms" (MA/IFCAM/18/39), the ANR project GRALMECO (ANR-21-CE48-0004-01) and the French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).

1 Introduction

Path problems such as Hamiltonian Path are among the most fundamental problems in the field of algorithms. Hamiltonian Path can be generalized as the covering problem Path Cover [2], where one asks to cover the vertices of an input graph using a prescribed number of paths. The corresponding packing problem is Disjoint Paths where, given a set of k pairs of terminal vertices of a graph G, one asks whether there are k vertex-disjoint paths in G, each joining two paired terminals. Disjoint Paths is a fundamental problem and a precursor to the field of parameterized complexity due to the celebrated fixed-parameter tractable algorithm devised by Robertson and Seymour [21] for the parameter "number of paths". We recall that in the field of parameterized algorithms and complexity, one studies *parameterized problems*, whose input I comes together with a parameter k. A parameterized problem is said to be FPT (fixed-parameter tractable) if it can be solved in time $f(k) \cdot |I|^{O(1)}$, for some computable function f. If the problem can be solved in time $O(|I|^{f(k)})$, it belongs to class XP; see, e.g., [7] for more details.

In this paper, we will not consider arbitrary paths, but *shortest paths*, which are fundamental for many applications. In the problem Disjoint Shortest Paths, given a graph G and k pairs of terminals, one asks whether G contains k vertex-disjoint shortest paths pairwise connecting the k pairs of terminals. This problem was introduced in [11] and recently shown to be polynomial-time solvable for every fixed k by an XP algorithm [16]. The problem Isometric Path Cover with Terminals, which we define as follows, is the covering counterpart of Disjoint Shortest Paths.

Isometric Path Cover with Terminals
Input: A graph G, and k pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$ called *terminals*.
Question: Does there exist a set of k shortest paths, the ith path being an s_i-t_i shortest path, such that each vertex of G belongs to at least one of the paths?

The name Isometric Path Cover with Terminals comes from the related Isometric Path Cover problem where the terminals are not part of the input, which was introduced in [12] in the context of the Cops and Robbers game on graphs (see also [1]).

Isometric Path Cover
Input: A graph G, and an integer k.
Question: Does there exist a set of k shortest paths, such that each vertex of G belongs to at least one of the paths?

Closely related variants of Isometric Path Cover with Terminals and Isometric Path Cover have been studied, in which there are only k terminals, and one asks to find $\binom{k}{2}$ shortest paths joining each pair of terminals. The version without terminals has been called Strong Geodetic Set in the literature; we call the version with terminals Strong Geodetic Set with Terminals. It was first studied (independently) in [8, 9], see also [15].

Strong Geodetic Set with Terminals
Input: A graph G, and a set of k vertices of G called *terminals*.
Question: Does there exist a set of $\binom{k}{2}$ shortest paths, each path joining a distinct pair of terminals, such that each vertex of G belongs to at least one of the paths?

The variant where the terminals are not given in the input was defined in [4] as follows.

Strong Geodetic Set
Input: A graph G, and an integer k.
Question: Does there exist a set of k terminals and a set of $\binom{k}{2}$ shortest paths, each path joining a distinct pair of terminals, such that each vertex of G belongs to at least one of the paths?

The complexity of these problems has been studied in the literature. It was shown in [5] that Isometric Path Cover is NP-complete, even for chordal graphs, a class for which the authors of [5] also showed that the problem can be approximated within a constant factor. For general graphs, it was shown in [22] that the problem can be approximated in polynomial time within a factor of $O(\log d)$, where d is the diameter of the input graph. It was proven to be polynomial-time solvable on block graphs [20]. It is shown in [8] that Strong Geodetic Set with Terminals is NP-hard. In [9], it is shown that this holds even for bipartite graphs of maximum degree 4 or diameter 6, however Strong Geodetic Set with Terminals is polynomial-time solvable on split graphs, graphs of diameter 2, block graphs, and cactus graphs. We prove here (cf. Proposition 15) that Isometric Path Cover with Terminals is also NP-complete.

Finally, Strong Geodetic Set is known to be NP-hard [4], even for bipartite graphs, chordal graphs, graphs of diameter 2 and cobipartite graphs [9] as well as for subcubic graphs of arbitrary girth [8]. However it is polynomial-time solvable on outerplanar graphs [19], cactus graphs, block graph and threshold graphs [9].

All these problems can also be studied in their edge-covering version, where one requires to cover all edges of the input graph by the corresponding shortest paths. For instance, the Strong Edge Geodetic Set problem is studied in [18].

Our results

Our main combinatorial theorem is as follows (see Section 2 for the definition of pathwidth).

▶ **Theorem 1.** *Let G be a graph whose **edge set** can be covered by at most k shortest paths. Then the pathwidth of G is at most $f(k)$, for function $f(k) = \sum_{i=1}^{k} 2^{i+1} \cdot \frac{k!}{(k-i)!}$.*

*If G is such that its **vertex set** can be covered by at most k shortest paths, then the pathwidth of G is at most $(2k-1) \cdot f(k)$.*

We actually show that in such a graph G, given an arbitrary vertex a and an integer D, the number of vertices at distance exactly D from a is upper-bounded by a function of k and it does not depend on the size of the input graph. It follows that a very simple linear-time algorithm based on a breadth-first search provides a path decomposition whose width is upper-bounded by the aforementioned function of k. The complexity of the algorithm itself does not depend on k.

Besides the combinatorial bounds, we employ the celebrated theorem of Courcelle [6], stating that problems expressible in Monadic Second-Order Logic (MSOL_2) can be solved in linear time for graphs of bounded treewidth (and thus, of bounded pathwidth). More precisely, we reduce the problem Isometric Path Cover with Terminals to an optimization problem expressible in MSOL_2. The result can also be obtained by dynamic programming but the algorithm would be tedious and not particularly efficient, therefore we prefer the general logic-based framework for further extensions. The running time is linear in n, the number of vertices of the graph, but super-exponential in the parameter k. Together with Theorem 1, this implies the following.

▶ **Theorem 2.** *Problems Isometric Path Cover with Terminals and Strong Geodetic Set with Terminals are FPT when parameterized by the number of terminals.*

▶ **Corollary 3.** *Problems Isometric Path Cover and Strong Geodetic Set are in XP when parameterized by the number of paths/terminals.*

Thanks to the flexibility of Monadic Second-Order Logic, our algorithmic results easily extend to the edge-covering versions of our problems, and to variants where we require the paths to be edge-disjoint, or vertex-disjoint (as studied in [17]). The second part of Theorem 2 answers positively a question asked in [15].

After some preliminaries in Section 2, we prove Theorem 1 in Sections 3 and 4. More specifically, Section 3 provides the upper bound on the pathwidth of graphs whose edges are coverable by k shortest paths, then the tools are extended to vertex-coverings in the next section. Algorithmic consequences (Theorem 2) are derived in Section 5, and we conclude with some open questions.

2 Preliminaries and notations

Paths and concatenation operators $\oplus$ and $\odot$

We refer to [10] for usual notations on graphs. In this paper we only consider undirected, unweighted graphs. For simplicity, we assume that our input graph $G = (V, E)$ is connected, though all our combinatorial and algorithmic results extend to non-connected graphs. As usually $N(x)$ denotes the neighbourhood of vertex x.

A path P of graph $G = (V, E)$ is a sequence of distinct vertices $(x_1, \ldots, x_l)$ such that for each $i, 1 \leq i \leq l-1$, $\{x_i, x_{i+1}\}$ is an edge of the graph. We also say that P is an x_1-x_l path. Note that our paths are simple and they do not use twice the same vertex. We denote by $V(P)$ the vertices of path P, and by $E(P)$ its edges. Given two vertices $x, y \in V(P)$, we denote by $P[x, y]$ the subpath of P between x and y. Moreover we let $|P|$ denote the *length* of path P, that is, the number of its edges. The *distance* between two vertices a and b in G is denoted $\mathrm{dist}(a, b)$ and corresponds to the length of a shortest a-b path.

Throughout the paper, we will construct paths by concatenation operations. It is convenient to think of our paths as directed: when we speak of an a-b path, we think of it as being directed from a to b.

Given two vertex disjoint paths $\nu = (x_1, \ldots, x_l)$ and $\eta = (y_1, \ldots, y_t)$ of G such that $\{x_l, y_1\}$ is an edge of G, we define the *concatenation operator* $\oplus$ whose result is $\nu \oplus \eta = (x_1, \ldots, x_l, y_1, \ldots, y_t)$. In particular, $|\nu \oplus \eta| = |\nu| + |\eta| + 1$.

We define similarly the *glueing operator* $\odot$ between two paths $\nu = (x_1, \ldots, x_l)$ and $\eta = (x_l, y_1, \ldots, y_t)$ with $V(\nu) \cap V(\mu) = \{x_l\}$ by $\nu \odot \eta = (x_1, \ldots, x_l, y_1, \ldots, y_t)$. Note that in this case $|\nu \odot \eta| = |\nu| + |\eta|$.

Path decompositions through breadth-first search

A path decomposition of $G = (V, E)$ is a sequence $\mathcal{P} = (X_1, X_2, \ldots, X_q)$ of vertex subsets of G, called *bags*, such that for every edge $\{x, y\} \in E$ there is at least one bag containing both endpoints, and for every vertex $x \in V$, the bags containing x form a continuous sub-sequence of $\mathcal{P}$. The width of $\mathcal{P}$ is $\max\{|X_i| - 1 \mid 1 \leq i \leq q\}$, and the *pathwidth* $\mathrm{pw}(G)$ of G is the minimum width over all path decompositions of G.

The *treewidth* $\mathrm{tw}(G)$ of graph G is defined similarly (see e.g. [10]), using a so-called tree decomposition; for our purpose, we only need to know that for any graph G, $\mathrm{tw}(G) \leq \mathrm{pw}(G)$, in particular any path decomposition is also a tree decomposition of the same width. We also need the following folklore lemma on path decompositions.

▶ **Lemma 4.** *Let $G = (V, E)$ be a graph, a be a vertex of G and K be an upper bound on the number of vertices of G at distance exactly D from a, for any integer D.*

Then, $\mathrm{pw}(G) \leq 2K - 1$. Moreover, a path decomposition of width $2K - 1$ can be computed in linear time, by breadth-first search.

Proof. Let $\mathrm{ecc}(a)$ be the eccentricity of vertex a (i.e., $\max_{x \in V} \mathrm{dist}(a, x)$). For any D with $0 \leq D \leq \mathrm{ecc}(a)$ we denote by $\mathrm{Layer}(D)$ the set of vertices at distance exactly D from a, i.e. the layers of a breadth-first search on G starting at a. Observe that, by taking as bags the unions $\mathrm{Layer}(D) \cup \mathrm{Layer}(D+1)$ of pairs of consecutive layers, $0 \leq D < \mathrm{ecc}(a)$, and by ordering them according to D, we obtain a path decomposition of G. Indeed for each edge $\{x, y\}$ both endpoints are in the same layer or in two consecutive layers, thus will appear in the same bag. For each vertex x, it appears in at most two bags: if $d = \mathrm{dist}(a, x)$ then x is in bags $\mathrm{Layer}(d-1) \cup \mathrm{Layer}(d)$ and $\mathrm{Layer}(d) \cup \mathrm{Layer}(d+1)$ (or one bag if $d = 0$ or $d = \mathrm{ecc}(a)$), and these bags appear consecutively in the decomposition. Since each layer has at most K vertices, the width of this decomposition is at most $2K - 1$. ◀

3 Warm-up: edge-covering with k shortest paths

As a warm-up, we start by proving Theorem 1 in the case when graph $G = (V, E)$ is edge-coverable by k shortest paths. In this case, there is a simple and elegant encoding of shortest paths leading to the desired result; the case of vertex-covering, which is more technical, will be studied in the next section.

In this section, $G = (V, E)$ denotes a graph whose edge set is coverable by k shortest paths. Let us fix such a set of paths $\mu_1, \ldots, \mu_k$, and call them the *base paths* of G. All constructions in this section are built on this particular set of base paths (without explicitly recalling it for each lemma, in order to ease the notations). We endow each base path μ_c, $1 \leq c \leq k$ with an arbitrary *direction*. E.g., assuming that the vertices of G are numbered from 1 to n, the direction of path P is from its smallest towards its largest end-vertex. A (directed) subpath $\mu_c[x, y]$ of μ_c is given a positive *sign* $+$ if it follows the direction of μ_c, otherwise it is given a negative sign $-$. For each edge e of G, let colours(e) be the set of all values $c \in \{1 \ldots, k\}$ such that e is an edge of μ_c.

Good colourings

Let P be an a-b path of G, from vertex a to vertex b. A *colouring* of P is a function col : $E(P) \to \{1, \ldots, k\}$ assigning to each edge e of the path one of its colours col$(e) \in$ colours(e). The colouring col of P is said to be *good* if, for any colour c, the set of edges using this colour form a connected subpath $P[x, y]$ of P. (Since our paths are simple, this condition entails that $P[x, y] = \mu_c[x, y]$.) A pair (P, col) formed by a path together with a good colouring is called *well-coloured*.

Operator $\odot$ defined in Section 2 naturally extends to coloured paths. Given a coloured path (P, col), we simply denote by $(P[x, y], \text{col})$ its restriction to a subpath $P[x, y]$ of P. Finally, we define for any path P and any colour $1 \leq c \leq k$ the function monochr$_c : E(P) \to c$. Hence all edges of the coloured path $(P, \text{monochr}_c)$ have colour c.

With these notations, any well-coloured a-b path (P, col) with colours $(c_1, \ldots, c_l)$ appearing in this order can be written as $(\mu_{c_1}[x_1, x_2], \text{monochr}_{c_1}) \odot (\mu_{c_2}[x_2, x_3], \text{monochr}_{c_2}) \odot \ldots \odot (\mu_{c_l}[x_l, x_{l+1}], \text{monochr}_{c_l})$ for some vertices $a = x_1, x_2, x_3 \ldots, x_l, x_{l+1} = b$. In full words, $P[x_i, x_{i+1}]$ are the monochromatic subpaths of (P, col), coloured c_i.

▶ **Lemma 5** (Good colouring lemma). *For any pair of vertices a and b of G, there exists a well-coloured a-b path (P, col) such that P is a shortest a-b path.*

We will simply call (P, col) a well-coloured shortest a-b path.

Proof. Among all shortest a-b paths, choose one that admits a colouring with a minimum number of monochromatic subpaths. Let P be this path, and col the corresponding colouring. Assume by contradiction that the colouring is not good. Then there exist three edges $e_1 = \{x_1, y_1\}$, $e_2 = \{x_2, y_2\}$ and $e_3 = \{x_3, y_3\}$, appearing in this order, such that $\text{col}(e_1) = \text{col}(e_3) \neq \text{col}(e_2)$. Assume w.l.o.g. that the vertices appear in the order $x_1, y_1, x_2, y_2, x_3, y_3$ from a to b (note that we my have $y_1 = x_2$ or $y_2 = x_3$). Let $c = \text{col}(e_1) = \text{col}(e_3)$. Therefore y_1 and x_3 are on the same base path μ_c. Let P' be the path obtained from P by replacing $P[y_1, x_3]$ by $\mu_c[y_1, x_3]$. First, P' is no longer than P, since $\mu_c[y_1, x_3]$ is a shortest possible y_1-x_3 path of graph G (in particular, P' has no repeated vertices). Second, in P' we can colour all edges of $P'[y_1, x_3]$ with colour c, and keep all other colours unchanged. Hence P' has strictly fewer monochromatic subpaths than P – a contradiction. ◀

Let (P, col) be a well-coloured a-b path, with colours $(c_1, \ldots, c_l)$ in this order. Recall that each monochromatic subpath $P[x_i, x_{i+1}]$ of P, of colour c_i, induces a sign on the corresponding base path μ_c (positive if $P[x_i, x_{i+1}]$ has the same direction as μ_c, negative otherwise). Therefore, we can define the *colours-signs word* $\mathrm{ColoursSignsW}(P, \mathrm{col}) = ((c_1, s_1), (c_2, s_2), \ldots, (c_l, s_l))$ on the alphabet $\{1, \ldots, k\} \times \{+, -\}$, corresponding to the colours and signs of the monochromatic subpaths of P, according to the ordering in which these subpaths appear from a to b. Observe that such words have at most k letters on an alphabet of size $2k$. Therefore the number of different words is upper bounded by a function of k:

▶ **Lemma 6.** *The number of possible colours-signs words, over all well-coloured paths of G, is upper bounded by $h(k) = \sum_{l=1}^{k} 2^l \frac{k!}{(k-l)!}$.*

Proof. We claim that the number of colours-signs words of l letters is upper bounded by $2^l \frac{k!}{(k-l)!}$. Observe that the colours form a word of length l, on an alphabet of size k, without repetition. The number of such words is $\frac{k!}{(k-l)!}$ (e.g., by choosing l letters among the k possible ones, and applying all possible permutations). Since each letter also has a sign in $\{+, -\}$, we multiply this quantity by 2^l, and the conclusion follows by summing over all possible values of l. ◀

The following crucial lemma implies that, given a start vertex a, a distance D and a colours-signs word ω, there is at most one vertex b at distance D from a, such that the well-coloured shortest a-b path respects word ω. This will allow to upper bound the number of vertices at distance D from a.

▶ **Lemma 7** (Colours-signs encoding)**.** *Consider two vertices b and c at the same distance from some vertex a of G. Let (P, col) be a well-coloured shortest a-b path and (P', col') be a well-coloured shortest a-c path. If $\mathrm{ColoursSignsW}(P, \mathrm{col}) = \mathrm{ColoursSignsW}(P', \mathrm{col}')$, then $b = c$.*

Proof. We proceed by induction on the number of letters of the word $\mathrm{ColoursSignsW}(P, \mathrm{col})$. Let us denote it by $\omega = ((c_1, s_1), (c_2, s_2), \ldots, (c_l, s_l))$.

Let $P[a, x_2]$ (resp. $P'[a, x'_2]$) be the maximal subpath of P (resp. P') of colour c_1 starting from a. Assume w.l.o.g. that $P[a, x_2]$ is at least as long as $P'[a, x'_2]$. Since both are subpaths of μ_{c_1}, starting from a and having the same sign s_1 w.r.t. μ_{c_1}, we actually have that $P'[a, x'_2]$ is contained in $P[a, x_2]$, in particular x'_2 is between a and x_2 in P and in μ_{c_1}.

Observe that, if word ω has only one letter, $P[a, x_2] = P$ and $P'[a, x'_2] = P'$, thus they are all of the same length. Since they are of the same sign w.r.t. μ_{c_1}, this implies that $x_2 = x'_2 = b = c$, which proves the base case of our induction.

Assume now that ω has $l \geq 2$ letters and that the lemma is true for words of length $l - 1$.

Consider first the case when $P[a, x_2]$ and $P'[a, x'_2]$ have the same length. Then $x_2 = x'_2$ is also the first vertex of the subpaths of colour c_2 of both P and P'. Then $(P[x'_2, b], \mathrm{col})$ and $(P'[x'_2, c], \mathrm{col}')$ are well-coloured shortest paths of the same length, and have the same colours-signs word $((c_2, s_2), \ldots, (c_l, s_l))$, with $l - 1$ letters. Hence the property follows by the induction hypothesis.

We now handle the second and last case, when $P[a, x_2]$ is strictly longer than $P'[a, x'_2]$.

Let x_3 be the last vertex of the subpath coloured c_2 in (P, col). In particular, x'_2, x_2 and x_3 are all vertices of μ_{c_2}.

Let us make an easy but crucial observation: on μ_{c_2}, vertex x_2 is between x'_2 and x_3. To prove this claim, note that in path P, vertices a, x'_2 and x_2 appear in this order (as observed in the beginning of the proof), and by construction x_2 appears between a and

x_3. Therefore a, x'_2, x_2, x_3 appear in this order on P, which is a shortest path. Hence $dist(x'_2, x_3) = dist(x'_2, x_2) + dist(x_2, x_3)$. Since the three vertices x'_2, x_2, x_3 are all on the shortest path μ_{c_2}, they must appear in this order on it. Consequently, $\mu_{c_2}[x'_2, x_3]$ induces the same sign s_2 on μ_{c_2} as $P[x_2, x_3] = \mu_{c_2}[x_2, x_3]$.

In particular, in path $(P[x'_2, b], \mathrm{col}) = (P[x'_2, x_2], \mathrm{monochr}_{c_1}) \odot (P[x_2, x_3], \mathrm{monochr}_{c_2}) \odot (P[x_3, b], \mathrm{col})$, we can replace the first subpath $P[x'_2, x_2]$ coloured c_1 by $\mu_{c_2}[x'_2, x_2]$, coloured c_2, without changing the total length. We obtain the well-coloured shortest x'_2-b path $(\tilde{P}[x'_2, b], \tilde{col}) = (\mu_{c_2}[x'_2, x_3], \mathrm{monochr}_{c_2}) \odot (P[x_3, b], \mathrm{col})$. Its colours-signs word is $((c_2, s_2), \ldots, (c_l, s_l))$, the same as for the shortest x'_2-c path $(P'[x'_2, c], \mathrm{col}')$. Moreover, the two paths have the same length, $|P| - |P[a, x'_2]|$, hence by the induction hypothesis we have $b = c$, which proves our lemma. ◀

▶ **Corollary 8.** *For any vertex a of G and any integer D, there are at most $h(k) = \sum_{l=1}^{k} 2^l \frac{k!}{(k-l)!}$ vertices at distance exactly D from a.*

Proof. For any fixed vertex a and fixed integer D, thanks to Lemma 7 the number of vertices x at distance exactly D from a is upper-bounded by the number of colours-signs words, which is in turn upper bounded by $h(k)$ by Lemma 6. ◀

In order to complete the proof of the first part of Theorem 1 and show that $\mathrm{pw}(G) \leq 2h(k)$, we simply apply Lemma 4 with $K = h(k)$.

4 Vertex-covering with k shortest paths

In this section, $G = (V, E)$ denotes a graph whose *vertices* can be covered by k shortest paths $\mu_1, \ldots, \mu_k$. As before we endow each base path μ_c with a direction, but now colours are assigned to vertices. We can easily adapt the notions of good colourings of the previous section to these vertex-colourings. Again, for any pair of vertices a and b, there is a well-coloured shortest path joining them (Lemma 9), which defines a colours-signs word. But we shall see that now (unlike in the simpler case of edge-coverings), we may have two distinct vertices b and c at the same distance D from a, and well-coloured shortest a-b and a-c paths with the same colours-signs word. More efforts will be needed to recover a (slightly larger) upper bound on the number of vertices at distance D from a (Corollary 14).

Good colourings

For each vertex v of G, let $\mathrm{colours}(v)$ denote the set of indices (colours) $c \in \{1 \ldots, k\}$ such that v is a vertex of μ_c. Let P be an a-b path of G, from vertex a to vertex b. A *colouring* of P is a function $\mathrm{col} : V(P) \to \{1, \ldots, k\}$ assigning to each vertex v of P one of its colours $\mathrm{col}(v) \in \mathrm{colours}(v)$. A coloured path is a pair (P, col). The colouring col of P is said to be *good* if, for any colour c, the subgraph induced by the set of *vertices* using this colour c forms a connected subpath $P[x, y]$ of P (which implies that $P[x, y] = \mu_c[x, y]$). A coloured path (P, col) where col is a good colouring is called *well-coloured*.

Operators $\oplus$ and $\odot$ naturally extend to (vertex) coloured paths, with the precaution that $(\nu, \mathrm{col}) \odot (\eta, \mathrm{col}')$ is defined only when their common vertex x, the last of ν and first of η, satisfies $\mathrm{col}(x) = \mathrm{col}'(x)$. Given a coloured path (P, col), we again denote by $(P[x, y], \mathrm{col})$ its restriction to a subpath $P[x, y]$ of P. For each colour $1 \leq c \leq k$, let now $(P, \mathrm{monochr}_c)$ denote the monochromatic colouring of $V(P)$ with colour c.

With these notations, any well-coloured a-b path (P, col) with colours $(c_1, \ldots, c_l)$ is of the form $(\mu_{c_1}[a_1, b_1], \mathrm{monochr}_{c_1}) \oplus (\mu_{c_2}[a_2, b_2], \mathrm{monochr}_{c_2}) \oplus \ldots \oplus (\mu_{c_l}[a_l, b_l], \mathrm{monochr}_{c_l})$ for some vertices $a = a_1, b_1, a_2, b_2, \ldots, a_l, b_l = b$, as in Figure 2.

Like in the previous section, we have:

▶ **Lemma 9.** *For any pair of vertices a and b of G, there exists a well-coloured shortest a-b path.*

Proof. Among all shortest a-b paths, choose one that admits a colouring with a minimum number of monochromatic subpaths. Let (P, col) be such a coloured path. Assume for a contradiction that the colouring col is not good. Then there exist three vertices x, y and z, appearing in this order in P such that $\mathrm{col}(x) = \mathrm{col}(z) \neq \mathrm{col}(y)$. Therefore x and z are on the same base path μ_c. Let P' be the path obtained from P by replacing $P[x, z]$ by $\mu_c[x, z]$. Notice that P' is no longer than P, since $\mu_c[x, z]$ is a shortest x-z path of graph G. Moreover, in P' we can colour all vertices of $P'[x, z]$ with colour c, and keep all other colours unchanged. Hence P' has strictly fewer monochromatic subpaths than P – a contradiction. ◀

Colours-signs word

Let (P, col) be a well-coloured a-b path; we recall that we see it as being directed from a to b. As in Section 3, each monochromatic subpath P' of P, say of colour c, induces a sign ($+$ or $-$) depending on its direction w.r.t. μ_c, if P' has at least two vertices. If P' has a unique vertex, we assign to it sign $+$. Therefore, we can again define the *colours-signs word* $\mathrm{ColoursSignsW}(P, \mathrm{col}) = ((c_1, s_1), (c_2, s_2), \ldots, (c_l, s_l))$ on the alphabet $\{1, \ldots, k\} \times \{+, -\}$, corresponding to the colours and signs of the monochromatic subpaths of P according to the ordering in which these subpaths appear from a to b.

In the case of edge-covering, we had the elegant statement of Lemma 7, by which, given a vertex a, a colours-signs word ω and a distance D, there is a unique vertex b (if any exists) at distance D such that the well-coloured shortest a-b path corresponds to this word. Unfortunately, this does not extend to vertex-covering: Figure 1 presents two distinct vertices b and c located at the same distance D from vertex a, together with a well-coloured shortest a-b path (P, col) and a well-coloured shortest a-c path (P', col'). These coloured paths starting from a have the same colours-signs word and the same length, but this does not imply that their endpoints are equal.

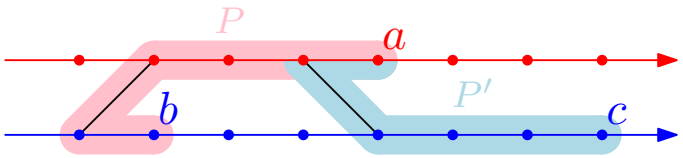


Figure 1 Two well-coloured paths (P, col) and (P', col') with same colours-signs word $\omega = ((red, -), (blue, +))$, same length (5) and same start vertex (a), but different end-vertices (b and c).

Canonical well-coloured paths

In order to obtain a situation somewhat similar to the case of edge-covering, we define a *canonical representation* of well-coloured paths (see Figure 2 for an example). Given a colours-signs word $\omega = ((c_1, s_1), (c_2, s_2), \ldots, (c_l, s_l))$ with no repetition of colours, a start vertex a and a length L, we define a *unique* well-coloured path $\mathrm{CanonPath}(\omega, a, L)$ starting from a, of the prescribed length L and having the colours-signs word ω, as follows:

1. If $l = 1$, let b be the vertex at distance L of a on the path μ_{c_1} w.r.t. s_1. Then $\text{CanonPath}(\omega, a, L) = (\mu_{c_1}[a, b], \text{monochr}_{c_1})$.
2. Otherwise, let b_1 be the first vertex of μ_{c_1} starting from a w.r.t. s_1 having a neighbour in μ_{c_2}. Among the vertices of μ_{c_2} adjacent to b_1, choose a_2 to be the one that appears first in μ_{c_2}, according to its direction. We let:

$$\text{CanonPath}(\omega, a, L) = (\mu_{c_1}[a, b_1], \text{monochr}\, c_1) \oplus \text{CanonPath}(((c_2, s_2), \ldots, (c_l, s_l)), a_2, L - |\mu_{c_1}[a, b_1]| - 1) \quad (1)$$

Note that the path $\text{CanonPath}(\omega, a, L)$ might not exist, e.g. if, in the base case of the construction, $a \notin \mu_{c_1}$ or the subpath of μ_{c_1} starting from a and following sign s_1 is shorter than L or, in the second step, vertex b_1 does not exist, or $|\mu_{c_1}[a, b_1]|$ exceeds L.

▶ **Observation 10.** *Given a colours-signs word ω, a vertex a and a length L, if the path $(P, \text{col}) = \text{CanonPath}(\omega, a, L)$ exists, then it is well-coloured. Moreover, this path is unique and satisfies $(P, \text{col}) = \text{CanonPath}(\text{ColoursSignsW}(P, \text{col}), a, |P|)$.*

Proof. Notice first that, by construction, we have $|P| = L$ and $\text{ColoursSignsW}(P, \text{col}) = \omega$ which implies the last part of the statement. Now since this path is recursively obtained by the concatenation of monochromatic (simple) subpaths (Equation 1), it is well-coloured. The uniqueness of path (P, col) comes from the deterministic choices made during the algorithm, which concludes the proof. ◀

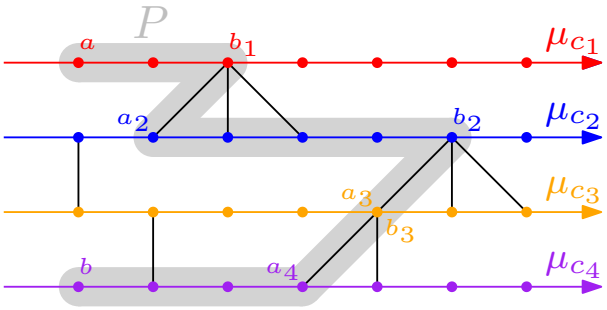


Figure 2 Example of a canonical well-coloured a-b path (P, col) with a colours-signs word $\text{ColoursSignsW}(P, \text{col}) = ((c_1, +), (c_2, +), (c_3, +), (c_4, -))$.

▶ **Definition 11** (canonical well-coloured path). *A well-coloured a-b path (P, col) is called* canonical *if $(P, \text{col}) = \text{CanonPath}(\text{ColoursSignsW}(P, \text{col}), a, |P|)$.*

To obtain a result similar to Lemma 7, we provide an algorithm that takes as input a well-coloured a-b path P, the corresponding good colouring col with l colours, and that computes a canonical well-coloured a-b path whose length is upper-bounded by $|P| + 2(l - 1)$ (see Algorithm 1 and Lemma 12). As before, let $\omega = ((c_1, s_1), \ldots, (c_l, s_l))$ be the colours-signs word of (P, col) and for each $1 \leq i \leq l$, let a_i (resp. b_i) denote the first (resp. last) vertex of (P, col) coloured c_i. Informally, the algorithm recursively computes a coloured path as follows: if $l = 1$, we let $\text{Canonize}(P, \text{col}) = (P, \text{col})$. Otherwise, we consider b'_1 as the first vertex of $P[a, b_1]$ (hence, of μ_{c_1} starting from a following sign s_1) having a neighbour in μ_{c_2} (line 4). As in the definition of the CanonPath function, we choose the neighbour a'_2 of b'_1 to be the vertex of μ_{c_2} that appears first on this path, according to its direction, among the neighbours of b'_1 (line 5). We next replace $P[b'_1, b_2]$ by $(b'_1, a'_2) \odot \mu_{c_2}[a'_2, b_2]$ (see line 6 and

Figure 3), and then re-apply the transformation on the new path starting from a'_2 (line 7). We note that the colours-signs word of the resulting well-coloured path may be different from ω, since a sign of ω might be flipped in the construction of the new path in line 6.

Algorithm 1 Function computing the canonical well-coloured path Canonize(P, col).

Input : A well-coloured a-b path (P, col)
Output : A canonical well-coloured a-b path
1 **Function** Canonize(P, col):
2 **if** $l == 1$ **then**
3 **return** (P, col);
  /* first vertex of μ_{c_1} starting from a w.r.t. s_1 having a neighbour in μ_{c_2} */
4 $b'_1 \leftarrow$ first vertex of $P[a, b_1]$ having a neighbour in μ_{c_2};
5 $a'_2 \leftarrow$ vertex of $V(\mu_{c_2}) \cap N(b'_1)$ that appears first on μ_{c_2} according to its direction;
  /* $P[b'_1, b_2]$ is replaced by $(b'_1, a'_2) \odot \mu_{c_2}[a'_2, b_2]$ */
6 $(P', \text{col}') = (\mu_{c_2}[a'_2, b_2], \text{monochr}_{c_2}) \odot (P[b_2, b], \text{col})$;
7 **return** $(P[a, b'_1], \text{monochr}_{c_1}) \oplus \text{Canonize}(P', \text{col}')$;

▶ **Lemma 12** (canonization of well-coloured paths). *Given a well-coloured a-b path (P, col), with a colours-signs word of l letters, the result of* Canonize(P, col) *is a canonical well-coloured a-b path of length at most $|P| + 2(l-1)$.*

Proof. Denote $(P_c, \text{col}_c) = \text{Canonize}(P, \text{col})$ and let $\omega_c = \text{ColoursSignsW}(P_c, \text{col}_c)$ be its colours-signs word. We first claim that (P_c, col_c) is a canonical well-coloured path, i.e. equal to $\text{CanonPath}(\omega_c, a, |P_c|)$ (Definition 11). In order to prove the claim we proceed by induction on the number l of colours of (P, col). The property is true if (P, col) has a unique colour, since in this (base) case, $(P_c, \text{col}_c) = (P, \text{col})$ (line 3 of Algorithm 1).

Otherwise, by induction, (P_c, col_c) is formed by $(P[a, b'_1], \text{monochr}_{c_1})$ concatenated with $(P'', \text{col}'') = \text{Canonize}(P', \text{col}')$, where (P', col') is a well-coloured a'_2-b path with $l-1$ different colours. Let $\omega'' = \text{ColoursSignsW}(P'', \text{col}'')$ be the colours-signs word of P''. By the induction hypothesis, $(P'', \text{col}'') = \text{CanonPath}(\omega'', a'_2, |P''|)$. Moreover, for constructing path P_c we started from a along μ_{c_1} following sign s_1 until we reached the first vertex b'_1 adjacent to vertex a'_2 of μ_{c_2}, and concatenated it with the canonical a'_2-b path (P'', col''), as in the definition of the CanonPath function. Therefore, path (P_c, col_c) is canonical for the triple $(\omega_c, a, |P_c|)$.

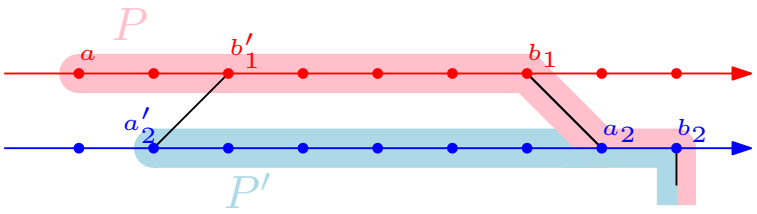


Figure 3 The construction of Canonize(P, col).

It remains to show that the length of P_c is at most $|P| + 2(l-1)$. We proceed again by induction on l. The property is true if $l = 1$ since $P_c = P$. Otherwise, for $l \geq 2$, recall that $(P_c, \text{col}_c) = (P[a, b'_1], \text{monochr}_{c_1}) \oplus (P'', \text{col}'')$ where $(P'', \text{col}'') = \text{Canonize}(P', \text{col}')$ and $(P', \text{col}') = (\mu_{c_2}[a'_2, b_2], \text{monochr}_{c_2}) \odot (P[b_2, b], \text{col})$ is a well-coloured a'_2-b path with $l-1$ different colours (by lines 6 and 7 of Algorithm 1). Now we show that:

$$|P[a, b'_1] \oplus P'| \leq |P| + 2 \tag{2}$$

We recall that in the well-coloured path (P, col), b_1 denotes the last vertex on P such that $\mathrm{col}(b_1) = c_1$ and a_2 the first vertex on P such that $\mathrm{col}(a_2) = c_2$. In particular (see Figure 3),

$$P[a, b'_1] \oplus P' = P[a, b'_1] \oplus \mu_{c_2}[a'_2, b_2] \odot P[b_2, b]$$

while P can be described by

$$P = P[a, b'_1] \odot \mu_{c_1}[b'_1, b_1] \oplus \mu_{c_2}[a_2, b_2] \odot P[b_2, b]$$

Therefore, in order to prove Inequality 2, we need to show that

$$|\mu_{c_2}[a'_2, b_2]| \leq |\mu_{c_1}[b'_1, b_1]| + |\mu_{c_2}[a_2, b_2]| + 2 \tag{3}$$

Observe that $|\mu_{c_2}[a'_2, a_2]| \leq |\mu_{c_1}[b'_1, b_1]| + 2$, otherwise $\mu_{c_2}[a'_2, a_2]$ would not be a shortest a'_2-a_2 path since the a'_2-a_2 path $(a'_2) \oplus \mu_{c_1}[b'_1, b_1] \oplus (a_2)$ would be of length $|\mu_{c_1}[b'_1, b_1]| + 2$, thus shorter. Next, observe that $|\mu_{c_2}[a'_2, b_2]| \leq |\mu_{c_2}[a'_2, a_2]| + |\mu_{c_2}[a_2, b_2]|$ no matter in which order the vertices a'_2, a_2 and b_2 appear on μ_{c_2}. These observations prove Inequality 3, which proves Inequality 2.

Inequality 2 entails that $|P'| \leq |P| + 1 - |P[a, b'_1]|$. By the induction hypothesis on (P'', col''), since (P', col') uses $l - 1$ colours, we have that $|P''| \leq |P'| + 2(l - 2)$. Thus

$$|P''| \leq |P| + 2l - 3 - |P[a, b'_1]|.$$

Now since $(P_c, \mathrm{col}_c) = (P[a, b'_1], \mathrm{monochr}_{c_1}) \oplus (P'', \mathrm{col}'')$ we have that $|P_c| = |P[a, b'_1]| + 1 + |P''|$. Plugging this into the previous inequality we have $|P_c| \leq |P[a, b'_1]| + 1 + |P| + 2l - 3 - |P[a, b'_1]|$, which completes the proof of our lemma. ◀

▶ **Corollary 13** (canonical representation). *Given two vertices a and b, there exists a canonical well-coloured a-b path (P, col) such that $|P| \leq \mathrm{dist}(a, b) + 2(k - 1)$.*

Proof. Let (P_s, col_s) be a shortest well-coloured a-b path, which exists by Lemma 9. By Lemma 12, $(P, \mathrm{col}) = \mathrm{Canonize}(P_s, \mathrm{col}_s)$ is the required canonical well-coloured a-b path, of length $|P| \leq |P_s| + 2(k - 1) = \mathrm{dist}(a, b) + 2(k - 1)$. ◀

In particular, Lemma 12 implies the following.

▶ **Corollary 14.** *Let G be a graph whose vertices are covered by k shortest paths. For any vertex a of G and any fixed distance D, there are at most $g(k)$ vertices at distance D from a, where:*

$$g(k) = (2k - 1) \cdot \left[\sum_{i=1}^{k} 2^i \cdot \frac{k!}{(k - i)!} \right]$$

Proof. For any fixed vertex a and fixed length L, the number of vertices that can be reached from a through a canonical well-coloured path of length L is upper bounded by $h(k) = \sum_{i=1}^{k} 2^i \cdot \frac{k!}{(k-i)!}$, the number of colours-signs words, by Lemma 6. Moreover for each vertex b, by Corollary 13, there exists a canonical well-coloured a-b path P such that $|P| \leq \mathrm{dist}(a, b) + 2(k - 1)$. Therefore the number of vertices at a fixed distance D from a in G is at most $(2k - 1) \cdot h(k)$. Indeed, vertex b is uniquely identified by a, the colours-signs word of the well-coloured path P, and the quantity $|P| - \mathrm{dist}(a, b)$. The latter has $2k - 1$ possible values, from 0 to $2(k - 1)$, which completes the proof. ◀

In order to complete the proof of Theorem 1 (in the case where the k paths cover all vertices of the graph) and show that the pathwidth of G is at most $2 \cdot g(k)$, we apply Lemma 4 for $K = g(k)$, and the conclusion follows.

5 Algorithmic consequences

Problem Strong Geodetic Set with Terminals is known to be NP-complete by [8]. By a simple reduction, so is Isometric Path Cover with Terminals.

▶ **Proposition 15.** *Isometric Path Cover with Terminals is NP-Complete.*

Proof. We provide a straightforward reduction from Strong Geodetic Set with Terminals. Let $(G = (V, E), k)$ be an instance of Strong Geodetic Set with Terminals with terminals $v_1, \dots v_k$. We build an instance of Isometric Path Cover with Terminals by considering all $\binom{k}{2}$ possible pairs of terminals i.e. $(G = (V, E), \bigcup_{1 \leq i < j \leq k} \{(v_i, v_j)\})$. By definition, (G, k) is a Yes-instance of Strong Geodetic Set with Terminals, i.e. there exists a set of $\binom{k}{2}$ shortest paths covering $V(G)$ if and only if (G, k') is a Yes-instance of Isometric Path Cover with Terminals, i.e. a set of k' shortest v_i-v_j paths, $1 \leq i < j \leq k$ covering $V(G)$. ◀

We now prove Theorem 2 and Corollary 3. Recall that, for simplicity, we assume that our input graph is connected, but all results easily extend to disconnected graphs. We first show that problem Isometric Path Cover with Terminals is FPT when parameterized by k, the number of pairs of terminals. As a first consequence, so is problem Strong Geodetic Set with Terminals, a special case of Isometric Path Cover with Terminals with $\binom{k}{2}$ pairs of terminals. (Both problems are NP-complete, by [8] and Appendix 15.)

Corollary 3 follows immediately, since for both Isometric Path Cover and Strong Geodetic Set it suffices to try all possible sets of terminals and use the FPT algorithms for the versions with terminals.

Let us focus on Isometric Path Cover with Terminals, with parameter k, input G and the k pairs of terminals $(s_1, t_1), \dots, (s_k, t_k)$. We can assume that the treewidth of the input graph is upper bounded by a function of k, as stated in Theorem 1, and that we have in the input a tree decomposition of such width. Indeed recall that Theorem 1 does not only provide a combinatorial bound on the pathwidth of YES-instances, but also a simple, BFS-algorithm for computing the suitable path decompositions (which is also, as stated in Section 2, a tree decomposition of the same width). If the algorithm fails to find a path decomposition of small width, we can directly conclude that our input graph is a NO-instance.

Therefore, we can use the classical Monadic Second-Order Logic of graphs (MSOL_2) tools on bounded treewidth graphs. MSOL_2 includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices, and sets of edges, the quantifiers $\forall$ and $\exists$ that can be applied to these variables, and five binary relations: $\text{adj}(u, v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent; $\text{inc}(v, e)$, where v is a vertex variable and e is an edge variable and the interpretation is that v is incident to e; $v \in V'$, where v is a vertex variable and V' is a vertex set variable; the similar $e \in E'$ on edge variable e and edge set variable E', and eventually equality of two variables of the same nature.

By a celebrated theorem of Courcelle [6], any problem expressible in MSOL_2 can be solved in time $f(\text{tw}) \cdot n$ time on bounded treewidth graphs, if a tree decomposition of the input graph is also given. Function f depends on the formula (hence, on the problem).

Courcelle's theorem extends in several ways to optimization problems, and slightly larger classes of formulae, e.g., allowing to identify a fixed number of terminal vertices, as we shall detail later. Here we will refer to [3], one of the (alternative) proofs of Courcelle's theorem, with some extensions.

As noted for example in [3], MSOL_2 allows to express properties as $\text{Connected}(V', E')$ where V' is a vertex set variable and E' is an edge set variable and the property is true if and only if (V', E') is a connected subgraph of G. Also let $\text{Cover}(V_1, \ldots, V_k)$ express the fact that vertex subsets $V_1, \ldots, V_k$ cover all vertices of the graph, by simply stating that $\forall x(x \in V_1 \vee x \in V_2 \vee \ldots x \in V_k)$.

This allows us to express Isometric Path Cover with Terminals as an optimization MSOL_2 problem, called an EMS-problem in [3].

Let $\varphi(E_1, E_2, \ldots, E_k)$ be the formula on edge sets $E_1, \ldots, E_k$ expressing the property that there exist k connected subgraphs $(V_1, E_1), \ldots, (V_k, E_k)$ of G such that the sets $V_1, \ldots, V_k$ cover all vertices of G, and graph (V_i, E_i) contains terminals s_i, t_i, for all $1 \leq i \leq k$. More formally:

$$\begin{aligned} \varphi(E_1, E_2, \ldots, E_k) &= \exists\, V_1, V_2, \ldots V_k\ [(s_1 \in V_1) \wedge (t_1 \in V_1) \wedge \cdots \wedge (s_k \in V_k) \wedge (t_k \in V_k) \\ &\wedge \text{Cover}(V_1, \ldots, V_k) \wedge \text{Connected}(V_1, E_1) \wedge \cdots \wedge \text{Connected}(V_k, E_k)] \end{aligned}$$

Consider now the optimization version of this problem, where the goal is to find edge sets $E_1, E_2, \ldots, E_k$ satisfying $\varphi(E_1, E_2, \ldots, E_k)$ and minimizing $|E_1| + |E_2| + \cdots + |E_k|$. Let OptCover denote this optimum. By Theorem 5.6 in [3], this problem can be solved in linear time on bounded treewidth graphs. More precisely, Arnborg et al. [3] call such problems "EMS linear extremum problems", in the sense that they correspond to linear optimization functions over the sizes of set variables, when these variables satisfy an MSOL_2 formula over labelled graphs with a fixed number of labels (here we consider each terminal vertex labeled with a different label). In contemporary terms, the problem is FPT parameterized by treewidth plus the number of terminals, and the running time is linear in n.

We now conclude by observing that our input is a YES-instance of Isometric Path Cover with Terminals if and only if $\text{OptCover} = \text{dist}(s_1, t_1) + \text{dist}(s_2, t_2) + \ldots \text{dist}(s_k, t_k)$. Indeed if there exist the required shortest s_i-t_i paths $P_1, \ldots, P_k$ covering the vertex set of the whole graph, then their edge sets $E_1, \ldots, E_k$ provide a solution for our optimization problem whose objective is the sum of the lengths of the paths. Conversely, for any of the k connected subgraphs (V_i, E_i) of G such that $s_i, t_i \in V_i$, we have $|E_i| \geq \text{dist}(s_i, t_i)$. Therefore by simply checking if OptCover corresponds to the sum of the distances between pairs of terminals, we decide whether the input satisfies Isometric Path Cover with Terminals. Altogether, this problem is FPT parameterized by k, which concludes the proof of Theorem 2.

As mentioned in the introduction, the same techniques extend to variants where the covering paths are required to be edge-disjoint or vertex-disjoint, by simply adding disjointness conditions in the MSOL_2 formula φ.

6 Conclusion

We have shown that graphs that can be covered by k shortest paths have their pathwidth upper-bounded by a function of k. Our bound is super-exponential, $2^{\Theta(k \log k)}$. The first natural open question is whether this upper bound can be improved to something smaller, perhaps even polynomial in k? Such an improvement cannot rely on path decompositions based on the layers of an arbitrary BFS, since we have examples (see Figure 4) where the

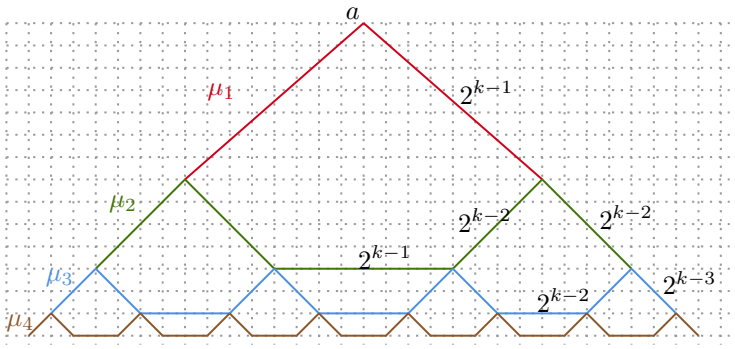


Figure 4 A graph that can be edge-covered with k shortest paths and with 2^k vertices at distance $2^k - 1$ from a. Therefore, a path decomposition obtained by a BFS from vertex a has width exponential in k. Nevertheless, one can easily prove that this graph has pathwidth at most k.

same layer contains 2^k vertices. Nevertheless, we leave as an open question whether graphs whose vertices (or edges) can be covered by k shortest paths have treewidth at most a polynomial in k.

Observe that the approach does not generalize to coverings with few *induced paths*, since grids have arbitrarily large treewidth but are edge-coverable by four induced paths.

On the algorithmic side, we have proved that problems Isometric Path Cover with Terminals and Strong Geodetic Set with Terminals are FPT parameterized by the number of terminals. This directly entails that problems Isometric Path Cover and Strong Geodetic Set are in XP with respect to the same parameter, by simply enumerating all possible pairs (respectively, sets) of terminals. An exciting open question is whether these two problems are FPT. By Theorem 1, this is equivalent to asking if the problems are FPT when parameterized by the solution size (i.e., number of paths/terminals) + pathwidth. (Indeed, if k is the number of terminals, Theorem 1 ensures that either the pathwidth pw of the input graph is upper bounded by a function $f(k)$, or we can directly reject the input for being a NO-instance. Therefore, if one of the problems is FPT parameterized by $k + \text{pw}$, we obtain an FPT algorithm parameterized by k as follows. The algorithm checks that $\text{pw} \leq f(k)$ as in Theorem 1, by a simple breadth-first search from an arbitrary vertex. If the assertion is false, the algorithm rejects. Otherwise it simply remains to apply the algorithm parameterized by $k + \text{pw}$ on parameter $k + f(k)$.) Nevertheless, the answer to the question whether these problems are FPT for parameter $k + \text{pw}$ seems non-trivial. At least, while many optimization problems are FPT when parameterized by treewidth/pathwidth, several problems including constraints on distances remain $W[1]$-hard even when parameterized by such structural parameters, plus solution size. We can cite recent hardness results for d-Scattered Set [13], whose goal is to find a large set of vertices at pairwise distance at least d or, even closer to our problems, Geodetic set [14], where one aims to find a small set of terminals of the input graph such that the set of all shortest paths between every pair of terminals covers the graph.

References

1 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984. doi:10.1016/0166-218X(84)90073-8.

2 Giovanni Andreatta and Francesco Mason. Path covering problems and testing of printed circuits. *Discrete Applied Mathematics*, 62(1):5–13, 1995. doi:10.1016/0166-218X(94)00142-Z.

3 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

4 Andrew Arokiaraj, Sandi Klavžar, Paul D. Manuel, Elizabeth Thomas, and Antony Xavier. Strong geodetic problems in networks. *Discussiones Mathematicae Graph Theory*, 40(1):307–321, 2020. doi:10.7151/dmgt.2139.

5 Dibyayan Chakraborty, Antoine Dailly, Sandip Das, Florent Foucaud, Harmender Gahlawat, and Subir K. Ghosh. Complexity and algorithms for isometric path cover on chordal graphs and beyond. In *Proceedings of the 33rd International Symposium on Algorithms and Computation, ISAAC 2022*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

6 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

8 Tom Davot, Lucas Isenmann, and Jocelyn Thiebaut. On the approximation hardness of geodetic set and its variants. In *Proceedings of the 27th International Computing and Combinatorics Conference, COCOON 2021*, volume 13025 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 2021. doi:10.1007/978-3-030-89543-3_7.

9 João Henrique Gonçalves de Sousa. Exact algorithms and computational complexity for the strong geodetic set problem. Master's thesis, Universidade Federal de Minas Gerais, Brazil, 2018. URL: http://hdl.handle.net/1843/ESBF-BA8NJD.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998. doi:10.1016/S0166-218X(97)00121-2.

12 David C. Fisher and Shannon L. Fitzpatrick. The isometric number of a graph. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 38(1):97–110, 2001.

13 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.

14 Leon Kellerhals and Tomohiro Koana. Parameterized complexity of geodetic set. In *Proceedings of the 15th International Symposium on Parameterized and Exact Computation, IPEC 2020*, volume 180 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.20.

15 Carlos V. G. C. Lima, Vinicius F. dos Santos, João H. G. Sousa, and Sebastián A. Urrutia. On the computational complexity of the strong geodetic recognition problem, 2022. doi:10.48550/ARXIV.2208.01796.

16 Willian Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 169–178. SIAM, 2021. doi:10.1137/1.9781611976465.12.

17 Paul Manuel. On the isometric path partition problem. *Discussiones Mathematicae Graph Theory*, 41(4):1077–1089, 2021. doi:10.7151/dmgt.2236.

18 Paul Manuel, Sandi Klavžar, Antony Xavier, Andrew Arokiaraj, and Elizabeth Thomas. Strong edge geodetic problem in networks. *Open Mathematics*, 15(1):1225–1235, 2017. doi:10.1515/math-2017-0101.

19 Mauro Mezzini. An $O(mn^2)$ algorithm for computing the strong geodetic number in outerplanar graphs. *Discussiones Mathematicae Graph Theory*, 42(2):591–599, 2022. doi:10.7151/dmgt.2311.

20 Jun-Jie Pan and Gerard J. Chang. Isometric-path numbers of block graphs. *Information Processing Letters*, 93(2):99–102, 2005. doi:10.1016/j.ipl.2004.09.021.

21 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

22 Maximilian Thiessen and Thomas Gaertner. Active learning of convex halfspaces on graphs. In *Proceedings of the 35th Conference on Neural Information Processing Systems, NeurIPS 2021*, volume 34, pages 23413–23425. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper/2021/file/c4bf1e24f3e6f92ca9dfd9a7a1a1049c-Paper.pdf.

On Maximizing Sums of Non-Monotone Submodular and Linear Functions

Benjamin Qi ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We study the problem of `Regularized Unconstrained Submodular Maximization` (`RegularizedUSM`) as defined by [Bodek and Feldman '22]. In this problem, we are given query access to a non-negative submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ and a linear function $\ell\colon 2^{\mathcal{N}} \to \mathbb{R}$ over the same ground set $\mathcal{N}$, and the objective is to output a set $T \subseteq \mathcal{N}$ approximately maximizing the sum $f(T) + \ell(T)$. Specifically, an algorithm is said to provide an (α, β)-approximation for `RegularizedUSM` if it outputs a set T such that $\mathbb{E}[f(T) + \ell(T)] \geq \max_{S \subseteq \mathcal{N}}[\alpha \cdot f(S) + \beta \cdot \ell(S)]$. We also study the setting where S and T are constrained to be independent in a given matroid, which we refer to as `Regularized` *`Constrained`* `Submodular Maximization` (`RegularizedCSM`).

The special case of `RegularizedCSM` with monotone f has been extensively studied [Sviridenko et al. '17, Feldman '18, Harshaw et al. '19]. On the other hand, we are aware of only one prior work that studies `RegularizedCSM` with non-monotone f [Lu et al. '21], and that work constrains ℓ to be non-positive. In this work, we provide improved (α, β)-approximation algorithms for both `RegularizedUSM` and `RegularizedCSM` with non-monotone f. In particular, we are the first to provide nontrivial (α, β)-approximations for `RegularizedCSM` where the sign of ℓ is unconstrained, and the α we obtain for `RegularizedUSM` improves over [Bodek and Feldman '22] for all $\beta \in (0, 1)$.

In addition to approximation algorithms, we provide improved inapproximability results for all of the aforementioned cases. In particular, we show that the α our algorithm obtains for `RegularizedCSM` with unconstrained ℓ is essentially tight for $\beta \geq \frac{e}{e+1}$. Using similar ideas, we are also able to show 0.478-inapproximability for maximizing a submodular function where S and T are subject to a cardinality constraint, improving a 0.491-inapproximability result due to [Oveis Gharan and Vondrak '10].

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Approximation algorithms analysis

Keywords and phrases submodular maximization, regularization, continuous greedy, inapproximability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.41

Related Version *Full Version*: `https://arxiv.org/abs/2205.15874`

Supplementary Material *Software (Source Code)*: `https://github.com/bqi343/maximizing-sums` archived at `swh:1:dir:0dbefd7cb01ec27b7ec94585161ada5356de2906`

Acknowledgements I thank Tasuku Soma for mentoring me through the MIT Undergraduate Research Opportunities Program, as well as Moran Feldman for providing many helpful suggestions.

1 Introduction

Submodularity is a property satisfied by many fundamental set functions, including coverage functions, matroid rank functions, and directed cut functions. Optimization of submodular set functions has found a wealth of applications in machine learning, including the spread of influence in social networks [13], sensor placement [16], information gathering [15], document summarization [17, 26, 9], image segmentation [11], and multi-object tracking [23], among others (see Krause and Golovin [14] for a survey).

Problems involving maximization of non-negative submodular functions can be classified as either *unconstrained* or *constrained*. In the unconstrained case, the objective is to return a set in the domain of the function approximately maximizing the function; we refer to this problem as USM. In the most commonly studied form of constrained submodular maximization, the returned set is subject to a "matroid constraint," which means that the returned set is constrained to be independent in a given matroid. We refer to this form of constrained submodular maximization as CSM. The simplest nontrivial example of a matroid constraint is a *cardinality* constraint, which means that an upper bound is given on the allowed size of the returned set. Additionally, we refer to the special case where the function f we are maximizing is monotone as "monotone CSM."

Approximation algorithms for both USM and CSM have been studied extensively. Say that an algorithm running in polynomial time with respect to the size of the ground set provides an *α-approximation* if it returns a set with expected value at least α times that of the optimum. For USM, a 0.5-approximation algorithm was provided by Buchbinder et al. [3]. For monotone CSM, a $(1 - e^{-1})$-approximation was achieved by Nemhauser et al. [20] using a *greedy* algorithm for the special case of cardinality constraints and later generalized by Calinescu et al. [4] to matroid constraints using the *continuous greedy* algorithm. For general CSM, the *measured continuous greedy* algorithm of Feldman et al. [7] achieves an $e^{-1} > 0.367$-approximation, and a subsequent algorithm due to Buchbinder and Feldman [2] achieves a 0.385-approximation.

To bound how far the algorithms from the previous paragraph are from optimal, corresponding inapproximability results have been shown. Say that a problem is *α-inapproximable* if no algorithm running in sub-exponential time with respect to the size of the ground set can provide an α-approximation. The first two approximation factors from the previous paragraph are in fact the best achievable; $(0.5+\epsilon)$-inapproximability and $(1-e^{-1}+\epsilon)$-inapproximability for any $\epsilon > 0$ were shown by Feige et al. [5] and Nemhauser and Wolsey [19], respectively, using ad hoc methods. On the other hand, the best achievable approximability for general CSM remains open; the best known inapproximability factor is 0.478 due to Oveis Gharan and Vondrak [8] using the *symmetry gap* technique of Vondrak [25]. This technique has the advantage of being able to succinctly reprove the inapproximability results of [19, 5] and many others.

In this work we study approximation algorithms for maximizing the sum of a non-negative submodular function f and a linear function ℓ. Sviridenko et al. [24] were the first to study algorithms for the sum $f + \ell$ in the case of f monotone, in order to provide improved approximation algorithms for monotone CSM with bounded *curvature*. Here, the curvature $c \in [0, 1]$ of a non-negative monotone submodular function g is roughly a measure of how far g is from being linear. They provided a $(1 - c/e - \epsilon)$-approximation algorithm and a complementary $(1 - c/e + \epsilon)$-inapproximability result. The idea of the algorithm is to decompose g into $f+\ell$ and show that an approximation factor of $1-e^{-1}$ can be achieved with respect to f and an approximation factor of 1 can be achieved with respect to ℓ simultaneously. Formally, if $\mathcal{I}$ is the independent set family of a matroid, the algorithm computes a set $T \in \mathcal{I}$ that satisfies

$$\mathbb{E}[g(T)] = \mathbb{E}[f(T) + \ell(T)] \geq \max_{S \in \mathcal{I}}[(1 - e^{-1} - \epsilon)f(S) + (1 - \epsilon)\ell(S)]$$

by first "guessing" the value of $\ell(S)$, and then running the continuous greedy algorithm. Subsequently, Feldman eliminated the need for the guessing step and the dependence on $\epsilon\ell(S)$ by introducing a *distorted objective* [6]. Many faster algorithms and practical applications for the case of f monotone have since been introduced [10, 12, 21]. Note that ℓ has several potential interpretations; while setting ℓ to be non-negative provides improved approximations

for monotone submodular functions with low curvature, setting ℓ to be non-positive allows it to serve as a *regularizer* or *soft constraint* that favors returning smaller sets as suggested by Harshaw et al. [10].

On the other hand, we know of only two prior works that study `RegularizedUSM` where f is not constrained to be monotone. Bodek and Feldman [1] were the first to consider the case where f is non-monotone and the sign of ℓ is unconstrained. They defined and studied the problem of `Regularized Unconstrained Submodular Maximization` (`RegularizedUSM`):

▶ **Definition 1** (`RegularizedUSM`). *Given query access to a (not necessarily monotone) non-negative submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ and a linear function $\ell\colon 2^{\mathcal{N}} \to \mathbb{R}$ over the same ground set $\mathcal{N}$, an algorithm is said to provide an (α, β)-approximation for `RegularizedUSM` if it outputs a set $T \subseteq \mathcal{N}$ such that $\mathbb{E}[f(T) + \ell(T)] \geq \max_{S \subseteq \mathcal{N}}[\alpha \cdot f(S) + \beta \cdot \ell(S)]$.*

The main approximation result of [1] is the first non-trivial approximation algorithm for `RegularizedUSM` with f non-monotone and the sign of ℓ unconstrained. Specifically, they used *non-oblivious local search* to provide $(\alpha(\beta) - \epsilon, \beta - \epsilon)$-approximations for `RegularizedUSM` for all $\beta \in (0, 1]$, where $\alpha(\beta) \triangleq \beta(1-\beta)/(1+\beta)$ [1, Theorem 1.2]. They also proved inapproximability results for the cases of ℓ non-negative and ℓ non-positive using the symmetry gap technique of Vondrak [25]. In particular, they showed $(1 - e^{-\beta} + \epsilon, \beta)$-inapproximability for monotone f and non-positive ℓ for all $\beta \geq 0$ [1, Theorem 1.1], essentially matching the $(1 - e^{-\beta} - \epsilon, \beta)$-approximability provided by Lu et al.'s *distorted measured continuous greedy* algorithm [18] (note that (α, β)-inapproximability is defined in the same way as α-inapproximability).

In this work, we present improved approximability and inapproximability results for `RegularizedUSM` as well as the setting where S and T are subject to a matroid constraint, which we define analogously as `Regularized` ***`Constrained`*** `Submodular Maximization` (`RegularizedCSM`):

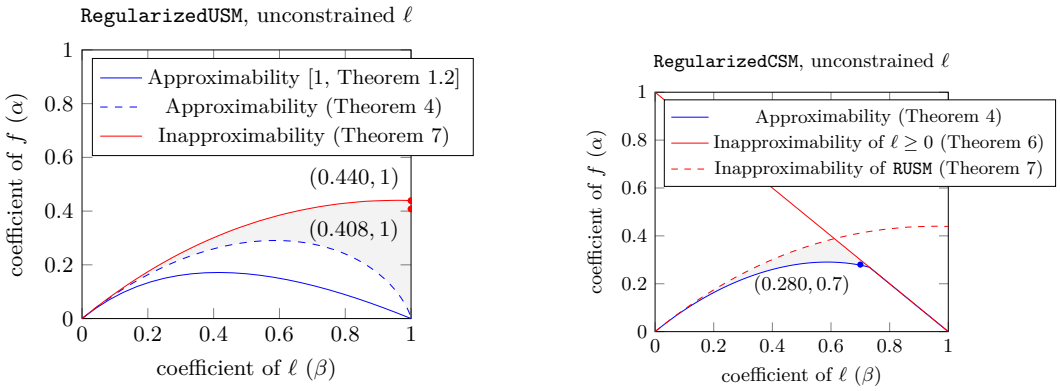
▶ **Definition 2** (`RegularizedCSM`). *Given query access to a (not necessarily monotone) non-negative submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ and a linear function $\ell\colon 2^{\mathcal{N}} \to \mathbb{R}$ over the same ground set $\mathcal{N}$, as well as a matroid with family of independent sets denoted by $\mathcal{I}$ also over the same ground set, an algorithm is said to provide an (α, β)-approximation for `RegularizedCSM` if it outputs a set $T \in \mathcal{I}$ such that $\mathbb{E}[f(T) + \ell(T)] \geq \max_{S \in \mathcal{I}}[\alpha \cdot f(S) + \beta \cdot \ell(S)]$.*

The only prior work considering `RegularizedCSM` for non-monotone f that we are aware of is that of Lu et al. [18], which as noted by [1] achieves $(\beta e^{-\beta} - \epsilon, \beta)$-approximations for `RegularizedCSM` for all $\beta \in [0, 1]$, but only when ℓ is constrained to be non-positive.

Organization of the Paper. We present the definitions and notation used throughout this work in Section 2. Sections 3–5 form the bulk of our paper and are described below.

1.1 Our Contributions

In Section 3 we consider the inapproximability of `CSM`. Oveis Gharan and Vondrak [8] used a symmetry gap construction [25] to prove 0.491-inapproximability of `CSM` in the special case where the matroid constraint is a cardinality constraint. Our first result improves the inapproximability factor for a cardinality constraint to 0.478 using a modified construction, matching the factor of the current best inapproximability result for `CSM` in the general case (also due to [8]).



(a) The $\left(\frac{\beta(1-\beta)}{1+\beta} - \epsilon, \beta - \epsilon\right)$-approximation for `RegularizedUSM` is due to [1, Theorem 1.2]. The red dot corresponds to $(0.408, 1)$-inapproximability, which is due to Theorem 8.

(b) The blue dot corresponds to $(0.280, 0.7)$-approximability, which is due to Theorem 5.

Figure 1 Graphical presentation of results for Sections 4 and 5. Following the convention of Bodek and Feldman [1], the x and y axes represent the coefficients of ℓ and f, respectively. We use blue for approximation algorithms and red for inapproximability results, and the shaded area represents the gap between the best known approximation algorithms and inapproximability results.

▶ **Theorem 3.** *There exist instances of the problem* $\max\{f(S) \colon S \subseteq \mathcal{N} \text{ and } |S| \leq w\}$ *where* f *is non-negative submodular such that a 0.478-approximation would require exponentially many value queries.*

In Section 4, we present the first nontrivial (α, β)-approximation algorithm for `RegularizedCSM` where the sign of ℓ is unconstrained. Furthermore, the α we obtain for `RegularizedUSM` improves over that of [1] for all $\beta \in (0, 1)$. The approximability and inapproximability results that follow are displayed in Figure 1.

▶ **Theorem 4.** *For all* $t \geq 0$*, there is a* $\left(\frac{te^{-t}}{t+e^{-t}} - \epsilon, \frac{t}{t+e^{-t}}\right)$*-approximation algorithm for* ***RegularizedUSM****. This algorithm achieves the same approximation guarantee for* ***RegularizedCSM*** *when* $t \leq 1$*.*

For certain values of β, we can achieve greater α for `RegularizedCSM` than that guaranteed by Theorem 4. Because the improvement is marginal and we do not have a closed form, our following result addresses only the specific case of $\beta = 0.7$. Note that Theorem 4 guarantees a $(0.277, 0.7)$-approximation (by setting $t \approx 0.925$).

▶ **Theorem 5** ([22, Theorem 8.3])**.** *There is a* $(0.280, 0.7)$*-approximation algorithm for* ***RegularizedCSM****.*

The proof of this result is much more involved than Theorem 4. Thus, due to space constraints, we defer this proof to the full version of this paper [22].

Finally, in Section 5 we present three inapproximability results for $f + \ell$ sums. The first shows that Theorem 4 is essentially tight for `RegularizedCSM` when $\beta \geq \frac{e}{e+1}$.

▶ **Theorem 6** (Inapproximability of `RegularizedCSM` Near $\beta = 1$)**.** *For any* $0 \leq \beta \leq 1$*, there exist instances of* ***RegularizedCSM*** *with non-negative* ℓ *such that a* $(1-\beta+\epsilon, \beta)$*-approximation would require exponentially many value queries.*

Our last two results show improved inapproximability for RegularizedUSM. The former generalizes Theorem 1.3 of [1] in order to show improved inapproximability for a range of β when ℓ is not necessarily constrained to be non-positive.

▶ **Theorem 7** (Inapproximability of RegularizedUSM). *There are instances of RegularizedUSM where $(\alpha(\beta), \beta)$ is inapproximable for any $(\alpha(\beta), \beta)$ in Table 1 (deferred to the appendix). In particular, $(0.440, 1)$ is inapproximable.*

The latter shows stronger inapproximability specifically for $\beta = 1$.

▶ **Theorem 8** (Inapproximability of RegularizedUSM, $\beta = 1$). *There are instances of RegularizedUSM where $(0.408, 1)$ is inapproximable.*

The best prior $(\alpha, 1)$-inapproximability result for RegularizedUSM is $(0.478, 1)$ due to Theorem 1.3 of [1], matching the 0.478-inapproximability result for CSM due to Oveis Gharan and Vondrak [8]. We note that as Bodek and Feldman [1] show inapproximability specifically for the case of non-positive ℓ, it is not too surprising that we can show improved inapproximability for general ℓ (though in the full version of this paper, we also show slightly improved inapproximability for non-positive ℓ [22, Theorem 5.6]). The gap between the best approximability and inapproximability results for RegularizedUSM remains quite large; in fact, it remains unclear whether an $(\epsilon, 1)$-approximation algorithm exists for any $\epsilon > 0$.

1.2 Our Techniques

To show approximability, the main techniques we use are the *measured continuous greedy* introduced by Feldman et al. [7] and used by [2, 18], as well as the *distorted objective* introduced by Feldman [6] and used by [18]. For Theorem 5 specifically, we additionally require the analysis of the 0.385-approximation algorithm for CSM due to Buchbinder et al. [2] and the "guessing step" used by Sviridenko et al. [24]. A more comprehensive overview of these techniques can be found in the full version of this paper [22, Section A.1].

To show inapproximability, the main technique we use is the symmetry gap of Vondrak [25], and most of our symmetry gap constructions are based on those of Oveis Gharan and Vondrak [8].

2 Preliminaries

Set Functions. Let $\mathcal{N} \triangleq \{u_1, u_2, \ldots, u_n\}$ denote the *ground set*. A set function $f \colon 2^{\mathcal{N}} \to \mathbb{R}$ is said to be *submodular* if for every two sets $S, T \subseteq \mathcal{N}$, $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. Equivalently, f is said to be submodular if it satisfies the property of "diminishing returns." That is, for every two sets $S \subseteq T \subseteq \mathcal{N}$ and an element $u \in \mathcal{N} \backslash T$, $f(u|S) \geq f(u|T)$, where $f(u|S) \triangleq f(S \cup \{u\}) - f(S)$ is the *marginal value* of u with respect to S. We use $f(u)$ as shorthand for $f(\{u\})$. All submodular functions are implicitly assumed to be non-negative unless otherwise stated.

A set function f is said to be *monotone* if for every two sets $S \subseteq T \subseteq \mathcal{N}$, $f(S) \leq f(T)$. A set function ℓ is said to be *linear* if there exist values $\{\ell_u \in \mathbb{R} \mid u \in \mathcal{N}\}$ such that for every set $S \subseteq \mathcal{N}$, $\ell(S) = \sum_{u \in S} \ell_u$. When considering the sum of a non-negative submodular function f and a linear function ℓ whose sign is unconstrained, define $\ell_+(S) \triangleq \sum_{u \in S} \max(\ell_u, 0)$ and $\ell_-(S) \triangleq \sum_{u \in S} \min(\ell_u, 0)$ to be the components of ℓ with positive and negative sign, respectively.

Value Oracles. We make the standard assumption that an algorithm for the sum $f+\ell$ does not have direct access to the representation of f; instead, it may obtain information about f only through a *value oracle*. Given any query set $S \subseteq \mathcal{N}$, a value oracle for f returns $f(S)$ in polynomial time. On the other hand, the coefficients of ℓ are directly provided to the algorithm.

Multilinear Extensions. All vectors of reals are in bold (e.g., $\mathbf{x}$). Given two vectors $\mathbf{x}, \mathbf{y} \in [0,1]^{\mathcal{N}}$, we define $\mathbf{x} \circ \mathbf{y}$ to be the coordinate-wise multiplication of $\mathbf{x}$ and $\mathbf{y}$. Given a set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}$, its *multilinear extension* is the function $F\colon [0,1]^{\mathcal{N}} \to \mathbb{R}$ defined by $F(\mathbf{x}) = \mathbb{E}[f(\mathtt{R}(\mathbf{x}))]$, where $\mathtt{R}(\mathbf{x})$ is a random subset of $\mathcal{N}$ including every element $u \in \mathcal{N}$ with probability $\mathbf{x}_u$, independently. One can verify that F is a multilinear function of its arguments as well an extension of f in the sense that $F(\mathbf{1}_S) = f(S)$ for every set $S \subseteq \mathcal{N}$. Here, $\mathbf{1}_S$ is the vector with value 1 at each $u \in S$ and 0 at each $u \in \mathcal{N} \backslash S$, and is known as the *characteristic vector* of the set S.

Matroid Polytopes. A *matroid* $\mathcal{M}$ may be specified by a pair of a ground set $\mathcal{N}$ and a family $\mathcal{I}$ of independent sets. The *matroid polytope* $\mathcal{P}$ corresponding to $\mathcal{M}$ is defined to be $conv(\{\mathbf{1}_S \mid S \in \mathcal{I}\})$, where *conv* denotes the convex hull. Due to the matroid axioms, $\mathcal{P}$ is guaranteed to be *down-closed*; that is, $0 \leq \mathbf{x} \leq \mathbf{y}$ and $\mathbf{y} \in \mathcal{P}$ imply $\mathbf{x} \in \mathcal{P}$. It is also well-known that $\mathcal{P}$ is *solvable*; that is, linear functions can be maximized over $\mathcal{P}$ in polynomial time [4, Section 2.3]. For `CSM` and `RegularizedCSM`, we let OPT denote any set such that $OPT \in \mathcal{I}$ (equivalently, $\mathbf{1}_{OPT} \in \mathcal{P}$), while for `USM` and `RegularizedUSM`, we let OPT denote any subset of $\mathcal{N}$. For example, in the context of `CSM`, $\mathbb{E}[f(T)] \geq \alpha f(OPT)$ is equivalent to $\forall S \in \mathcal{I}, \mathbb{E}[f(T)] \geq \alpha f(S)$.

Miscellaneous. We let ϵ denote any positive real. Many of our algorithms are "almost" (α, β) approximations in the sense that they provide an $(\alpha - \epsilon, \beta)$-approximation in poly $\left(n, \frac{1}{\epsilon}\right)$ time for any $\epsilon > 0$. Similarly, some of our results show $(\alpha + \epsilon, \beta)$-inapproximability for any $\epsilon > 0$.

3 Inapproximability of Maximization with Cardinality Constraint

In this section, we prove Theorem 3. First, we provide the relevant definitions about proving inapproximability using the symmetry gap technique from Vondrak [25].

▶ **Definition 9** (Symmetrization). *Let $\mathcal{G}$ be a group of permutations over $\mathcal{N}$. For $\mathbf{x} \in [0,1]^{\mathcal{N}}$, define the "symmetrization of $\mathbf{x}$" as $\overline{\mathbf{x}} = \mathbb{E}_{\sigma \in \mathcal{G}}[\sigma(\mathbf{x})]$, where $\sigma \in \mathcal{G}$ is uniformly random and $\sigma(\mathbf{x})$ denotes $\mathbf{x}$ with coordinates permuted by σ.*

▶ **Definition 10** (Symmetry Gap). *Let $\max\{f(S)\colon S \in \mathcal{F} \subseteq 2^{\mathcal{N}}\}$ be strongly symmetric with respect to a group $\mathcal{G}$ of permutations over $\mathcal{N}$, meaning that for all $\sigma \in \mathcal{G}$ and $S \subseteq 2^{\mathcal{N}}$, $f(S) = f(\sigma(S))$ and $S \in \mathcal{F} \Leftrightarrow S' \in \mathcal{F}$ whenever $\overline{\mathbf{1}_S} = \overline{\mathbf{1}_{S'}}$. Define $P(\mathcal{F}) = conv(\{\mathbf{1}_I\colon I \in \mathcal{F}\})$ to be the polytope associated with $\mathcal{F}$. Let $\mathbf{OPT} \triangleq \max_{\mathbf{x} \in P(\mathcal{F})} F(\mathbf{x})$ and $\overline{\mathbf{OPT}} \triangleq \max_{\mathbf{x} \in P(\mathcal{F})} F(\overline{\mathbf{x}})$. Then the symmetry gap of $\max\{f(S)\colon S \in \mathcal{F}\}$ is defined as $\gamma \triangleq \frac{\overline{\mathbf{OPT}}}{\mathbf{OPT}}$.*

▶ **Lemma 11** (Inapproximability due to Symmetry Gap). *Let $\max\{f(S)\colon S \in \mathcal{F}\}$ be an instance of non-negative submodular maximization, strongly symmetric with respect to $\mathcal{G}$, with symmetry gap γ. Let $\mathcal{C}$ be the class of instances $\max\{\tilde{f}(S)\colon S \in \tilde{\mathcal{F}}\}$ where $\tilde{f}$ is non-negative*

submodular and $\tilde{\mathcal{F}}$ is a refinement of $\mathcal{F}$. Then for every $\epsilon > 0$, any (even randomized) $(1+\epsilon)\gamma$-approximation algorithm for the class $\mathcal{C}$ would require exponentially many queries to the value oracle for $\tilde{f}(S)$.

The formal definition of refinement can be found in [25]. The important thing to note is that $\tilde{\mathcal{F}}$ satisfies the same properties as $\mathcal{F}$. In particular, $\tilde{\mathcal{F}}$ preserves cardinality and matroid independence constraints. Before proving Theorem 3, we start with a related lemma.

▶ **Lemma 12** (Inapproximability of Cardinality Constraint on Subset of Domain)**.** *Let T be some subset of the ground set $\mathcal{N}$. There exist instances of the problem $\max\{f(S)\colon S \subseteq \mathcal{N} \wedge |S \cap T| \leq w\}$ such that a 0.478-approximation would require exponentially many value queries.*

Proof. It suffices to provide $\mathcal{F}$, f, and $\mathcal{G}$ satisfying the definitions of Lemma 11 with symmetry gap $\gamma < 0.478$. The construction is identical to that of [8, Theorem E.2], except we omit $|S \cap \{a,b\}| \leq 1$ from the definition of $\mathcal{F}$. Specifically, we define $\mathcal{N} \triangleq \{a, b, a_{1\ldots k}, b_{1\ldots k}\}$ and

$$\mathcal{F} \triangleq \{S \mid S \subseteq \mathcal{N} \wedge |S \cap \{a_{1\ldots k}, b_{1\ldots k}\}| \leq 1\}$$

instead of:

$$\mathcal{F}_{orig} \triangleq \{S \mid S \subseteq \mathcal{N} \wedge |S \cap \{a,b\}| \leq 1 \wedge |S \cap \{a_{1\ldots k}, b_{1\ldots k}\}| \leq 1\}.$$

Recall that Theorem E.2 of [8] defines the submodular function f as the sum of the weighted cut functions of two directed hyperedges $(\{a_1, a_2, \ldots, a_k\}, a), (\{b_1, b_2, \ldots, b_k\}, b)$ and the undirected edge (a, b) (see Figure 4 of [8] for an illustration). Specifically, the weighted cut function on the directed hyperedge $(\{a_1, a_2, \ldots, a_k\}, a)$ contributes $\kappa \triangleq 0.3513$ to the value of $f(S)$ if $S \cap \{a_1, \ldots, a_k\} \neq \emptyset$ and $a \notin S$, and 0 otherwise. The weighted cut function on the directed hyperedge $(\{b_1, b_2, \ldots, b_k\}, b)$ is defined in the same way. Finally, the weighted cut function on the undirected edge (a, b) contributes $1 - \kappa$ if $|S \cap \{a, b\}| = 1$ and 0 otherwise. Thus, the multilinear extension of f is as follows:

$$\begin{aligned} F(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_{a_{1\ldots k}}, \mathbf{x}_{b_{1\ldots k}}) \triangleq & (1-\kappa)(\mathbf{x}_a(1-\mathbf{x}_b) + \mathbf{x}_b(1-\mathbf{x}_a)) \\ & + \kappa \left[\left(1 - \prod_{i=1}^{k}(1-\mathbf{x}_{a_i})\right)(1-\mathbf{x}_a) + \left(1 - \prod_{i=1}^{k}(1-\mathbf{x}_{b_i})\right)(1-\mathbf{x}_b) \right]. \end{aligned}$$

As in [8, Lemma 5.4], we let $\mathcal{G}$ be the group of permutations generated by $\{\sigma_1, \sigma_2\}$, where

$$\sigma_1(a) = b, \sigma_1(b) = a, \sigma_1(a_i) = b_i, \sigma_1(b_i) = a_i$$

swaps the two hyperedges, and

$$\sigma_2(a) = a, \sigma_2(b) = b, \sigma_2(a_i) = a_{i \pmod k + 1}, \sigma_2(b_i) = b_i$$

rotates the tail vertices of the first hyperedge. It is easy to check that $(f, \mathcal{F})$ are strongly symmetric with respect to both σ_1 and σ_2, and that the symmetrization of $\mathbf{x}$ is as follows:

$$\overline{\mathbf{x}} = \mathbb{E}_{\sigma \in \mathcal{G}}[\sigma(\mathbf{x})] = \begin{cases} \overline{\mathbf{x}}_a = \overline{\mathbf{x}}_b = \frac{\mathbf{x}_a + \mathbf{x}_b}{2} \\ \overline{\mathbf{x}}_{a_1} = \cdots = \overline{\mathbf{x}}_{a_k} = \overline{\mathbf{x}}_{b_1} = \cdots = \overline{\mathbf{x}}_{b_k} = \frac{\sum_{i=1}^{k}(\mathbf{x}_{a_i} + \mathbf{x}_{b_i})}{2k}. \end{cases}$$

Observe that

$$\mathbf{OPT} \geq \max_{S \in \mathcal{F}} f(S) \geq f(\{a, b_1\}) = (1-\kappa) + \kappa = 1.$$

Defining $q \triangleq \frac{\mathbf{x}_a+\mathbf{x}_b}{2}$ and $p \triangleq \frac{\sum_{i=1}^{k}(\mathbf{x}_{a_i}+\mathbf{x}_{b_i})}{2}$, the maximum of F over all symmetric $\mathbf{x}$ is thus:

$$\begin{aligned}\overline{\mathbf{OPT}} &= \max_{\mathbf{x}\in P(\mathcal{F})} F(\overline{\mathbf{x}}) = \max_{\mathbf{x}\in P(\mathcal{F})} F(q,q,\overbrace{p/k,p/k,\ldots,p/k}^{2k \text{ times}}) \triangleq \max_{\mathbf{x}\in P(\mathcal{F})} \hat{F}(q,p)\\ &= (1-\kappa)2q(1-q) + \kappa 2(1-q)(1-(1-p/k)^k)\\ &\approx (1-\kappa)2q(1-q) + \kappa 2(1-q)(1-e^{-p})\end{aligned}$$

where the approximate equality holds as $k \to \infty$. Now,

$$\overline{\mathbf{OPT}} = \max_{\mathbf{x}\in P(\mathcal{F})} F(\overline{\mathbf{x}}) = \max_{p\le 1/2} \hat{F}(q,p) = \max_{p,q\le 1/2} \hat{F}(q,p) = \max_{\mathbf{x}\in P(\mathcal{F}_{orig})} F(\overline{\mathbf{x}}) < 0.478.$$

The third equality holds (i.e., adding the constraint $q \le 1/2$ has no effect) since $\hat{F}(q,p) \le \hat{F}(1-q,p)$ for $q \in (1/2,1]$, while the inequality holds due to the proof of [8, Theorem E.2] and may be verified using a numerical optimizer. So the symmetry gap $\gamma = \frac{\overline{\mathbf{OPT}}}{\mathbf{OPT}}$ is less than 0.478, as desired. ◀

Now, to show Theorem 3, all we need to do is to convert the cardinality constraint on $S \cap T$ in Lemma 12 into a cardinality constraint on all of S.

Proof of Theorem 3. Again, it suffices to provide $\mathcal{F}$, f, and $\mathcal{G}$ satisfying the definitions of Lemma 11 with symmetry gap $\gamma < 0.478$. We start with the construction from Lemma 12, replace each element a_i and b_i with t copies $a_{i,1},\ldots,a_{i,t}$, and $b_{i,1},\ldots,b_{i,t}$ and set $w \triangleq t+1$. Then we redefine f such that F is as follows:

$$\begin{aligned}&F(\mathbf{x}_a,\mathbf{x}_b,\mathbf{x}_{a_{1\ldots k,1\ldots t}},\mathbf{x}_{b_{1\ldots k,1\ldots t}}) \triangleq (1-\kappa)(\mathbf{x}_a(1-\mathbf{x}_b)+\mathbf{x}_b(1-\mathbf{x}_a))\\ &+\kappa\left[\left(1-\prod_{i=1}^{k}\left(1-\frac{\sum_{j=1}^{t}\mathbf{x}_{a_{i,j}}}{t}\right)\right)(1-\mathbf{x}_a)+\left(1-\prod_{i=1}^{k}\left(1-\frac{\sum_{j=1}^{t}\mathbf{x}_{b_{i,j}}}{t}\right)\right)(1-\mathbf{x}_b)\right].\end{aligned}$$

Importantly, f remains non-negative submodular and symmetric, with the new symmetrization being as follows for an appropriate choice of $\mathcal{G}$:

$$\overline{\mathbf{x}} = \mathbb{E}_{\sigma\in\mathcal{G}}\left[\sigma(\mathbf{x})\right] = \begin{cases}\overline{\mathbf{x}}_a = \overline{\mathbf{x}}_b = \frac{\mathbf{x}_a+\mathbf{x}_b}{2}\\ \overline{\mathbf{x}}_{a_{1,1}} = \cdots = \overline{\mathbf{x}}_{a_{k,t}} = \overline{\mathbf{x}}_{b_{1,1}} = \cdots = \overline{\mathbf{x}}_{b_{k,t}} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{t}(\mathbf{x}_{a_{i,j}}+\mathbf{x}_{b_{i,j}})}{2kt}.\end{cases}$$

For example, we may define $\mathcal{G}$ to be the group generated by $\{\sigma_1,\sigma_2,\sigma_3\}$ where

- σ_1 swaps a with b and $a_{i,j}$ with $b_{i,j}$;
- σ_2 takes $a_{i,j}$ to $a_{i \pmod k+1,j}$ and leaves all other vertices unchanged;
- σ_3 takes $a_{1,j}$ to $a_{1,j \pmod t+1}$ and leaves all other vertices unchanged.

It can be verified that $F(\overline{\mathbf{x}})$ may be written in terms of the same function of two variables $\hat{F}(q,p)$ from Lemma 12. Let q be as defined above and redefine $p \triangleq \frac{\sum_{i=1}^{k}\sum_{j=1}^{t}(\mathbf{x}_{a_{ij}}+\mathbf{x}_{b_{ij}})}{2t}$, so that:

$$\begin{aligned}F(\overline{\mathbf{x}}) &\triangleq F(q,q,\overbrace{p/k,p/k,\ldots,p/k}^{2kt \text{ times}}) \triangleq \hat{F}(q,p)\\ &= (1-\kappa)2q(1-q)+\kappa 2(1-q)(1-(1-p/k)^k)\\ &\approx (1-\kappa)2q(1-q)+\kappa 2(1-q)(1-e^{-p})\end{aligned}$$

where the approximate equality holds as $k \to \infty$, same as before.

To finish, we must show that symmetry gap of f with respect to $\mathcal{F}$ remains less than 0.478 as $t \to \infty$. As in the proof of Lemma 12,

$$\mathbf{OPT} \geq \max_{S\colon |S|\leq t+1} f(S) \geq f(\{a, b_{11}, \ldots, b_{1t}\}) = 1,$$

$$\overline{\mathbf{OPT}} = \max_{\sum \mathbf{x}_i \leq t+1} F(\overline{\mathbf{x}}) = \max_{\sum \mathbf{x}_i \leq t+1} \hat{F}(q,p) \leq \max_{p \leq \frac{t+1}{2t}} \hat{F}(q,p) \approx \max_{p \leq 1/2} \hat{F}(q,p) < 0.478,$$

where the gap between the two sides of the approximate equality goes to 0 as $t \to \infty$ because $\hat{F}$ is Lipschitz continuous and its domain is bounded. So the symmetry gap $\gamma = \frac{\overline{\mathbf{OPT}}}{\mathbf{OPT}}$ is again less than 0.478, as desired. ◀

4 Approximability for Unconstrained ℓ

In this section, we prove Theorem 4. We need two lemmas, the first of which is trivial.

▶ **Lemma 13.** *When ℓ is unconstrained, there exists a $(0,1)$-approximation algorithm for* `RegularizedCSM`.

Proof. As noted in the preliminaries, linear functions can be maximized over a matroid polytope $\mathcal{P}$ in polynomial time. ◀

For the next lemma, we generalize the guarantee of the *distorted measured continuous greedy* of Lu et al. [18].

▶ **Lemma 14.** *For unconstrained ℓ and any $t_f \in [0,1]$, there is a polynomial-time algorithm for* `RegularizedCSM` *that returns $T \in \mathcal{I}$ such that*

$$\mathbb{E}[f(T) + \ell(T)] \geq (t_f e^{-t_f} - o(1)) f(OPT) + (1 - e^{-t_f} - o(1))\ell_+(OPT) + t_f \ell_-(OPT). \quad (1)$$

When $t_f > 1$, the algorithm provides the same approximation guarantee but is allowed to return any $T \subseteq \mathcal{N}$.

Proof. It suffices to show that for any $\epsilon > 0$, with high probability, Algorithm 1 from [18] generates $\mathbf{y}(t_f) \in [0,1]^{\mathcal{N}}$ such that $\mathbf{y}(t_f) \in t_f \cdot \mathcal{P}$ and

$$F(\mathbf{y}(t_f)) + L(\mathbf{y}(t_f)) \geq t_f e^{-t_f} f(OPT) + (1 - e^{-t_f})\ell_+(OPT) + t_f \ell_-(OPT) - \mathcal{O}(\epsilon M) \quad (2)$$

in $\mathrm{poly}(n, 1/\epsilon)$ time, where $M \triangleq \max\{\max_{u \in \mathcal{N}} f(u|\emptyset), -\min_{u \in \mathcal{N}} f(u|\mathcal{N} - u)\} > 0$. How to use $\mathbf{y}(t_f)$ to generate a set T satisfying the conditions in the statement of this lemma is standard and is deferred to the appendix.

First we briefly review the *measured continuous greedy* algorithm introduced by Feldman et al. [7]. The idea is to continuously evolve a solution $\mathbf{y}(t)$ from time $t = 0$ to time $t = t_f$ such that $\mathbf{y}(t) \in (t \cdot \mathcal{P}) \cap \left((1 - e^{-t}) \cdot [0,1]^{\mathcal{N}}\right)$. At all times, $\mathbf{y}'(t) = \mathbf{z}(t) \circ (\mathbf{1}_{\mathcal{N}} - \mathbf{y}(t))$, where $\mathbf{z}(t) \in \mathcal{P}$. To transform this continuous process into an algorithm running in finite time, it is necessary to discretize time into timesteps of size δ, where δ evenly divides t_f. Then $\mathbf{y}(t + \delta) \triangleq \mathbf{y}(t) + \delta \mathbf{z}(t) \circ (\mathbf{1}_{\mathcal{N}} - \mathbf{y}(t))$. How small δ needs to be to achieve the desired approximation factor is given by a polynomial in terms of n and ϵ.

Algorithm 1 of [18] combines measured continuous greedy with Feldman's *distorted objective* [6]. Specifically, algorithm 1 of [18] defines the objective at time t to be

$$\Phi(t) = (1 - \delta)^{(t_f - t)/\delta} F(\mathbf{y}(t)) + L(\mathbf{y}(t)) \approx e^{t - t_f} F(\mathbf{y}(t)) + L(\mathbf{y}(t))$$

and chooses $\mathbf{z}(t)$ so that with high probability,

$$\Phi(t+\delta) - \Phi(t) \geq \delta\left[e^{-t_f} f(OPT) + \ell(OPT)\right] - \frac{\delta}{t_f} \cdot \mathcal{O}\left(\epsilon M\right),$$

assuming that ℓ is non-positive [18, Lemma 3.7]. Summing this inequality over all $\frac{t_f}{\delta}$ timesteps yields the desired result for non-positive ℓ. [1]

We claim that when the sign of ℓ is unconstrained, the following generalization of [1, Lemma 3.7] holds:

$$\Phi(t+\delta) - \Phi(t) \geq \delta\left[e^{-t_f} f(OPT) + (1-\delta)^{t/\delta}\ell_+(OPT) + \ell_-(OPT)\right] - \frac{\delta}{t_f} \cdot \mathcal{O}\left(\epsilon M\right), \quad (3)$$

To show this, the only part of the proof of [18, Lemma 3.7] that needs to change is the part where [18, Lemma 3.6] is invoked. Lemma 3.6 of [18] states that for non-positive ℓ,

$$L(\mathbf{y}(t+\delta)) - L(\mathbf{y}(t)) = \delta L(\mathbf{z}(t) \circ (\mathbf{1}_{\mathcal{N}} - \mathbf{y}(t))) \geq \delta\langle \ell, \mathbf{z}(t)\rangle.$$

For unconstrained ℓ, we obtain the following inequality instead:

$$\begin{aligned} L(\mathbf{y}(t+\delta)) - L(\mathbf{y}(t)) &= \delta L(\mathbf{z}(t) \circ (\mathbf{1}_{\mathcal{N}} - \mathbf{y}(t))) \\ &= \delta\left(\langle \ell_+, \mathbf{z}(t) \circ (\mathbf{1}_{\mathcal{N}} - \mathbf{y}(t))\rangle + \langle \ell_-, \mathbf{z}(t) \circ (\mathbf{1}_{\mathcal{N}} - \mathbf{y}(t))\rangle\right) \\ &\geq \delta\left(\langle \ell_+, \mathbf{z}(t)\rangle \cdot (1-\delta)^{t/\delta} + \langle \ell_-, \mathbf{z}(t)\rangle\right), \end{aligned} \quad (4)$$

where the last inequality follows from [18, Lemma 3.1], which states that $\mathbf{y}_u(t) \leq 1 - (1-\delta)^{\frac{t}{\delta}}$ for all $u \in \mathcal{N}$. It is easy to verify that Equation (3) follows after substituting Equation (4) in place of [18, Lemma 3.6] in the proof of [18, Lemma 3.7].

To finish given Equation (3), we just need to check that $\sum_{i=0}^{t_f/\delta - 1} \delta(1-\delta)^i \geq 1 - (1-\delta)^{t_f/\delta} \geq 1 - e^{-t_f}$. Thus, Equation (2) has been proven. ◀

Proof of Theorem 4. The algorithm is described below.

Algorithm 1 Simple `RegularizedCSM` (t).

1. Let $T \leftarrow$ the set returned by running Lemma 14 for $t_f = t$ time.
2. Let $T' \leftarrow$ the set returned by Lemma 13 (trivial approximation).
3. **if** $f(T) + \ell(T) \geq \ell(T')$ **then return** T.
4. **else return** T'.

Now we show that the desired approximation factor is achieved. Disregard the factors of $o(1)$ in Lemma 14; they can always be taken into account later at the cost of introducing the factor of ϵ. Next, add $t + e^{-t} - 1$ times the inequality of Lemma 13 to the inequality of Lemma 14.

$$\begin{aligned} (t+e^{-t})\mathbb{E}[\max\left(f(T) + \ell(T), \ell(T')\right)] &\geq \mathbb{E}[f(T) + \ell(T)] + (t + e^{-t} - 1)\mathbb{E}[\ell(T')] \\ &\geq te^{-t} f(OPT) + t(\ell_+(OPT) + \ell_-(OPT)) \\ &= te^{-t} f(OPT) + t\ell(OPT). \end{aligned}$$

[1] Actually, the authors of [18] only described their algorithm for the case of $t_f = 1$. Though as noted in [1], the proof of their algorithm easily generalizes to arbitrary t_f. Of course, the step size δ must be adjusted accordingly.

To finish, divide both sides by $t + e^{-t}$ and return the set out of T and T' that gives the higher value of $f + \ell$. Thus, we have the desired result after accounting for the factors of $o(1)$:

$$\mathbb{E}[\max(f(T) + \ell(T), \ell(T'))] \geq \left(\frac{te^{-t}}{t + e^{-t}} - \epsilon\right) f(OPT) + \frac{t}{t + e^{-t}} \ell(OPT). \quad \blacktriangleleft$$

5 Inapproximability for Unconstrained ℓ

First, we state a generalization of the symmetry gap technique to the sum $f + \ell$ that we'll need for all of these results in this section.

▶ **Definition 15.** *We say that* $\max_{S \in \mathcal{F}} [f(S) + \ell(S)]$ *is strongly symmetric with respect to a group of permutations* $\mathcal{G}$ *if* $\ell(S) = \ell(\sigma(S))$ *for all* $\sigma \in \mathcal{G}$ *and* $(f, \mathcal{F})$ *are strongly symmetric with respect to* $\mathcal{G}$ *as defined in Definition 10.*

▶ **Lemma 16** (Inapproximability of (α, β) Approximations)**.** *Let* $\max_{S \in \mathcal{F}} [f(S) + \ell(S)]$ *be an instance of non-negative submodular maximization, strongly symmetric with respect to a group of permutations* $\mathcal{G}$*. For any two constants* $\alpha, \beta \geq 0$*, if*

$$\max_{\mathbf{x} \in P(\mathcal{F})} [F(\overline{\mathbf{x}}) + L(\overline{\mathbf{x}})] < \max_{S \in \mathcal{F}} [\alpha f(S) + \beta \ell(S)],$$

then no polynomial-time algorithm for `RegularizedCSM` *can guarantee a* (α, β)*-approximation. The same inapproximability holds for* `RegularizedUSM` *by setting* $\mathcal{F} = 2^{\mathcal{N}}$*.*

Proof. Theorem 3.1 of [1] proves this lemma only for the special case of $\mathcal{F} = 2^{\mathcal{N}}$ because the proof of [1, Lemma A.3] cites a special case of [25, Lemma 3.3] that only applies for $\mathcal{F} = 2^{\mathcal{N}}$. It suffices to modify the proof to cite the full version of [25, Lemma 3.3] instead. ◀

Now we are ready to prove Theorems 6–8. We note that the proofs of these results require the aid of a computer to verify. A link to the relevant code and a graph comparing these theorems to previous results are included in the appendix.

Proof of Theorem 6. Define $\alpha \triangleq 1 - \beta + \epsilon$. By Lemma 16, it suffices to construct a submodular function f satisfying

$$\max_{\mathbf{x} \in \mathcal{P}} [F(\overline{\mathbf{x}}) + L(\overline{\mathbf{x}})] < \max_{S \in \mathcal{I}} [\alpha \cdot f(S) + \ell(S)]. \tag{5}$$

We use the same f that Vondrak [25] uses for proving the inapproximability of maximization over matroid bases. Specifically, define $\mathcal{N} = \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$ and let f correspond to the sum of directed cut functions of k disjoint arcs; that is, $f(S) \triangleq \sum_{i=1}^{k} [a_i \in S \text{ and } b_i \notin S]$. Its multilinear extension is $F(\mathbf{x}_{a_1 \ldots a_k}, \mathbf{x}_{b_1 \ldots b_k}) = \sum_{i=1}^{k} \mathbf{x}_{a_i}(1 - \mathbf{x}_{b_i})$. We define $\mathcal{I}$ to consist of precisely the subsets of $\mathcal{N}$ that contain at most one element from $a_1, \ldots, a_k$ and at most $k - 1$ elements from $b_1, \ldots, b_k$, resulting in the following matroid independence polytope:

$$\mathcal{P} = \left\{ (\mathbf{x}_{a_i}, \mathbf{x}_{b_i}) \middle| \sum_{i=1}^{k} \mathbf{x}_{a_i} \leq 1 \text{ and } \sum_{i=1}^{k} \mathbf{x}_{b_i} \leq k - 1 \right\}.$$

Finally, we define ℓ as $\ell(a_i) = 0, \ell(b_i) = \frac{1}{k}$. Then the RHS of Equation (5) is at least:

$$\max_{S \in \mathcal{I}} [\alpha f(S) + \beta \ell(S)] \geq (\alpha f + \beta \ell)(\{a_1, b_2, b_3, \ldots, b_k\}) = \alpha + \beta \cdot \frac{k - 1}{k},$$

while the LHS of Equation (5) is:

$$\begin{aligned}\max_{\mathbf{x}\in\mathcal{P}}[F(\overline{\mathbf{x}})+L(\overline{\mathbf{x}})] &= \max_{0\le p\le 1/k, 0\le q\le (k-1)/k}[kp(1-q)+q]\\ &= \max_{0\le p\le 1/k, 0\le q\le (k-1)/k}[q(1-kp)+kp]\\ &= \max_{0\le p\le 1/k}[(1-kp)+kp] = 1,\end{aligned}$$

where the third equality follows because the expression is always maximized by setting $q=\frac{k-1}{k}$. For sufficiently large k we have

$$\alpha+\beta\cdot\frac{k-1}{k}\ge(\alpha+\beta)\frac{k-1}{k}=(1+\epsilon)\cdot\frac{k-1}{k}>1.$$

Equation (5) follows. ◀

In fact, Theorem 6 shows that Theorem 4 is essentially tight near $\beta=1$ for `RegularizedCSM`. On the other hand, Theorem 6 cannot possibly apply to `RegularizedUSM` because Theorem 4 achieves $(1-\beta+\epsilon,\beta)$-approximations for β close to one.

▶ **Corollary 17** (Tight `RegularizedCSM` Near $\beta=1$)**.** *There is a $(1-\beta-\epsilon,\beta)$-approximation algorithm for `RegularizedCSM` for any $\frac{e}{e+1}\le\beta<1$, almost matching the bound of Theorem 6.*

Proof. Setting $t=1$, the output of Theorem 4 is both a $\left(\frac{1}{e+1}-\epsilon,\frac{e}{e+1}\right)$-approximation and a $(0,1)$-approximation for `RegularizedCSM`. Therefore it is also an (α,β)-approximation for all (α,β) lying above the segment connecting $\left(\frac{1}{e+1}-\epsilon,\frac{e}{e+1}\right)$ and $(0,1)$. ◀

Proof of Theorem 7. Set f to be the same as defined in Lemma 12, and define $S\triangleq\{a,b_1\}$. For a fixed β, we can show (α,β)-inapproximability using Lemma 16 if it is possible to choose ℓ and κ such that:

$$\max_{\mathbf{x}\in[0,1]^{\mathcal{N}}}[F(\overline{\mathbf{x}})+L(\overline{\mathbf{x}})]<\alpha f(\{a,b_1\})+\beta\ell(\{a,b_1\})=\alpha+\beta\ell(\{a,b_1\}),$$

which is equivalent to

$$\max_{\mathbf{x}\in[0,1]^{\mathcal{N}}}[F(\overline{\mathbf{x}})+L(\overline{\mathbf{x}})]-\beta\ell(\{a,b_1\})<\alpha.$$

For a fixed β, our goal is to choose ℓ and κ to minimize the LHS of the above inequality. Theorem 1.3 of [1] sets $\ell_a=\ell_b=0$, and then chooses κ and $\ell_{a_{1\dots k}}=\ell_{b_{1\dots k}}\triangleq\ell_p$ in order to minimize the quantity

$$\begin{aligned}&\max_{\mathbf{x}\in[0,1]^{\mathcal{N}}}[F(\overline{\mathbf{x}})+L(\overline{\mathbf{x}})]-\beta\ell(\{a,b_1\})=\max_{\mathbf{x}\in[0,1]^{\mathcal{N}}}[F(\overline{\mathbf{x}})+L(\overline{\mathbf{x}})]-\beta\ell_p\\ &\approx\max_{0\le q\le 1,0\le p}\left[(1-\kappa)2q(1-q)+\kappa 2(1-q)(1-e^{-p})+2p\ell_p\right]-\beta\ell_p\end{aligned}$$

However, allowing $\ell_a=\ell_b\triangleq\ell_q$ to be nonzero gives better bounds for all β. That is, our goal is to compute

$$\min_{0\le\kappa\le 1,\ell_q,\ell_p}\left[\max_{0\le q\le 1,0\le p}\left[(1-\kappa)2q(1-q)+\kappa 2(1-q)(1-e^{-p})+2p\ell_p+2q\ell_q\right]-\beta(\ell_p+\ell_q)\right].$$

We can approximate the optimal value by brute forcing over a range of (κ,ℓ_q,ℓ_p). The best triples we found are displayed in Table 1. Allowing ℓ_q to be negative gives superior bounds for β near zero, while allowing ℓ_q to be positive gives superior bounds for β near one. In particular, for $\beta=1$, taking $\kappa=0.6022$, $\ell_p=-0.1819$, and $\ell_q=0.2152$ gives $\alpha(\beta)\approx 0.4390<0.440$. ◀

We can do slightly better than Theorem 7 for β very close to one with a construction inspired by [1, Theorem 1.6].

Proof of Theorem 8. Again, we use Lemma 16. Let $\mathcal{N} \triangleq \{a_{1\dots k}, b_{1\dots k}\}$, and define f as the directed cut function of a generalized hyperedge $(a_{1\dots k}; b_{1\dots k})$; that is, the generalized hyperedge is said to be cut by S if S contains at least one of the tails of the hyperedge $(a_{1\dots k})$ but not all of the heads of the hyperedge $(b_{1\dots k})$:

$$f(S) \triangleq [S \cap \{a_{1\dots k}\} \neq \emptyset] \cdot [\{b_{1\dots k}\} \not\subset S].$$

Also define $\ell(a_i) = -0.2037, \ell(b_i) = 0.2037$, $p \triangleq \sum_{i=1}^{k} \mathbf{x}_{a_i}$ and $q \triangleq k - \sum_{i=1}^{k} \mathbf{x}_{b_i}$, and $\mathcal{G}$ such that $a_1, \dots, a_k$ and $b_1, \dots, b_k$ are symmetric. Then as $k \to \infty$,

$$F(\overline{\mathbf{x}}) = \left(1 - (1 - p/k)^k\right)\left(1 - (1 - q/k)^k\right) \approx (1 - e^{-p})(1 - e^{-q}).$$

Now,

$$\begin{aligned}\max_{\mathbf{x} \in [0,1]^{\mathcal{N}}} [F(\overline{\mathbf{x}}) + L(\overline{\mathbf{x}})] &= \max_{p,q \geq 0} [(1 - e^{-p})(1 - e^{-q}) - 0.2037(p + q) + 0.2037k] \\ &= 0.2037k,\end{aligned}$$

where the last equality follows since the maximum is attained at $p = q = 0$, which may be verified using a numerical optimizer. On the other hand,

$$\max_{S} [\alpha f(S) + \ell(S)] \geq (f + \ell)(\{a_1, b_{1\dots k-1}\}) = \alpha + 0.2037(k - 2).$$

It follows from Lemma 16 that we have shown $(\alpha, 1)$-inapproximability for any α satisfying

$$0.2037k < \alpha + 0.2037(k - 2) \implies \alpha > 0.4074.$$ ◀

References

1. Kobi Bodek and Moran Feldman. Maximizing sums of non-monotone submodular and linear functions: Understanding the unconstrained case, 2022. To appear in ESA 2022. doi:10.48550/ARXIV.2204.03412.
2. Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.
3. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 649–658, 2012. doi:10.1109/FOCS.2012.73.
4. Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.
5. Uriel Feige, Vahab S Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.
6. Moran Feldman. Guess free maximization of submodular and linear sums. *Algorithmica*, 83(3):853–878, 2021.
7. Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 570–579. IEEE, 2011.
8. Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. SIAM, 2011.

9 Michael Gygli, Helmut Grabner, and Luc Van Gool. Video summarization by learning submodular mixtures of objectives. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3090–3098, 2015.

10 Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In *International Conference on Machine Learning*, pages 2634–2643. PMLR, 2019.

11 Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR 2011*, pages 1897–1904. IEEE, 2011.

12 Ehsan Kazemi, Shervin Minaee, Moran Feldman, and Amin Karbasi. Regularized submodular maximization at scale. In *International Conference on Machine Learning*, pages 5356–5366. PMLR, 2021.

13 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

14 Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3:71–104, 2014.

15 Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):1–20, 2011.

16 Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(2), 2008.

17 Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520, 2011.

18 Cheng Lu, Wenguo Yang, and Suixiang Gao. Regularized non-monotone submodular maximization, 2021. doi:10.48550/ARXIV.2103.10008.

19 George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

20 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

21 Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. An efficient framework for balancing submodularity and cost. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1256–1266, 2021.

22 Benjamin Qi. On maximizing sums of non-monotone submodular and linear functions, 2022. doi:10.48550/ARXIV.2205.15874.

23 Jianbing Shen, Zhiyuan Liang, Jianhong Liu, Hanqiu Sun, Ling Shao, and Dacheng Tao. Multiobject tracking by submodular optimization. *IEEE transactions on cybernetics*, 49(6):1990–2001, 2018.

24 Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Mathematics of Operations Research*, 42(4):1197–1218, 2017.

25 Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

26 Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 721–726, 2013.

A Appendix

Table 1 Inapproximability of $(\alpha(\beta), \beta)$-approximations for `RegularizedUSM` with unconstrained ℓ (Theorem 7). [2]

β	$\alpha(\beta)$	κ	ℓ_p	ℓ_q
0.1	0.0935	0.6708	-0.6064	-0.2644
0.2	0.1743	0.6598	-0.5370	-0.2102
0.3	0.2432	0.6577	-0.4809	-0.1504
0.4	0.3008	0.6519	-0.4261	-0.0940
0.5	0.3476	0.6465	-0.3778	-0.0407
0.6	0.3844	0.6390	-0.3309	0.0128
0.7	0.4114	0.6315	-0.2881	0.0655
0.8	0.4293	0.6226	-0.2498	0.1155
0.9	0.4383	0.6144	-0.2142	0.1668
1.0	0.4390	0.6022	-0.1819	0.2152

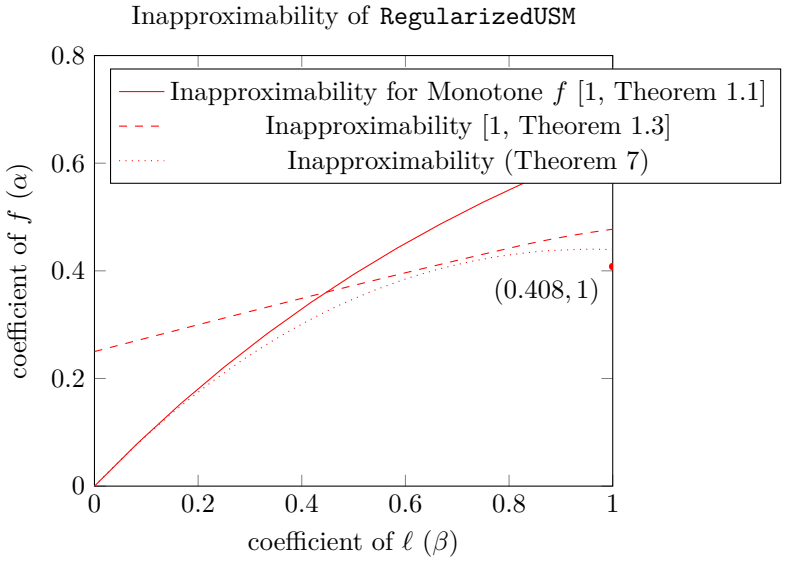


Figure 2 Previous and new inapproximability results for `RegularizedUSM` with an unconstrained linear function ℓ. $(0.408, 1)$-inapproximability is due to Theorem 8.

A.1 Omitted Proof

Proof of Lemma 14 (Omitted Details). We need to show that we can use an algorithm that outputs $\mathbf{y}(t_f)$ satisfying Equation (2) to generate $\mathbf{y} \in t_f \cdot \mathcal{P}$ such that

$$\mathbb{E}[F(\mathbf{y}) + L(\mathbf{y})] \geq (t_f e^{-t_f} - o(1)) f(OPT) + (1 - e^{-t_f} - o(1)) \ell_+(OPT) + t_f \ell_-(OPT), \quad (6)$$

[2] All values of $\alpha(\beta)$ are rounded to the nearest multiple of 10^{-4}. The numbers here are marginally better than those originally presented in the full paper [22, Table 4].

where the RHS of Equation (6) is identical to that of Equation (1). Once we have $\mathbf{y}$ satisfying Equation (6), we can use *pipage rounding* to round $\mathbf{y}$ to an integral solution $T \in \mathcal{I}$ if $t_f \leq 1$ or $T \subseteq \mathcal{N}$ otherwise [25]. Specifically, given $\mathbf{y} \in \mathcal{P}$, pipage rounding generates $T \in \mathcal{I}$ such that $\mathbb{E}[\mathbf{1}_T] = \mathbf{y}$ and $\mathbb{E}[F(\mathbf{1}_T) + L(\mathbf{1}_T)] \geq F(\mathbf{y}) + L(\mathbf{y})$.

We note that $\mathbf{y}(t_f)$ satisfying Equation (2) with high probability does not necessarily satisfy Equation (6) if either:

1. $M = \omega(f(OPT) + \ell_+(OPT))$.
2. Equation (2) does not hold.

However, we claim that output of the following algorithm *does* satisfy the conditions of Lemma 14:

Algorithm 2 `Modified Distorted Measured Continuous Greedy` (t).

```
Order the elements of 𝒩 such that f(u_1) ≤ f(u_2) ≤ ··· ≤ f(u_n).
for i = 0 to n do
    Let y_i ← the result of running [18, Algorithm 1] on 𝒩_i = {u_1, u_2, ..., u_i}.
    Let T_i ← the result of applying pipage rounding to y_i.
return the set T_i maximizing f(T_i) + ℓ(T_i).
```

It suffices to show that the two issues mentioned above are now resolved.

1. Assume

$$\emptyset \neq OPT \in \operatorname{argmax}_{OPT \subseteq \mathcal{N}} \left[t_f e^{-t_f} f(OPT) + (1 - e^{-t_f}) \ell_+(OPT) + t_f \ell_-(OPT) \right].$$

   Defining $j \triangleq \max\{i \mid u_i \in OPT\}$, it follows that

$$t_f e^{-t_f} f(OPT) + (1 - e^{-t_f}) \ell_+(OPT) \geq t_f e^{-t_f} \max(f(u_j), f(\emptyset)) \geq t_f e^{-t_f} \frac{M_j}{j}, \tag{7}$$

   where M_j is the value of M when restricted to $\mathcal{N}_j$. Thus, $M_j \leq n(f(OPT) + \ell_+(OPT))$. By choosing $\epsilon = o\left(\frac{1}{n}\right)$ in [18, Algorithm 1], Equation (2) implies that $\mathbf{y}_j$ satisfies Equation (6) with high probability.
2. Regardless of whether Equation (2) holds, the set T returned by Algorithm 2 always satisfies $f(T) + \ell(T) \geq f(T_0) + \ell(T_0) \geq 0$. Thus, if T satisfies Lemma 14 with high probability, it also satisfies Lemma 14 in expectation. ◀

A.2 Code

A Python notebook with code for all theorems in the full paper (including Theorems 5, 7, and 8 of this paper) can be found `here`.

On Reverse Shortest Paths in Geometric Proximity Graphs

Pankaj K. Agarwal
Department of Computer Science, Duke University, Durham NC, USA

Matthew J. Katz
Department of Computer Science, Ben-Gurion University of the Negev, Beer Sheva, Israel

Micha Sharir
School of Computer Science, Tel Aviv University, Tel Aviv, Israel

Abstract

Let S be a set of n geometric objects of constant complexity (e.g., points, line segments, disks, ellipses) in $\mathbb{R}^2$, and let $\varrho : S \times S \to \mathbb{R}_{\geq 0}$ be a *distance function* on S. For a parameter $r \geq 0$, we define the *proximity graph* $G(r) = (S, E)$ where $E = \{(e_1, e_2) \in S \times S \mid e_1 \neq e_2,\ \varrho(e_1, e_2) \leq r\}$. Given S, $s, t \in S$, and an integer $k \geq 1$, the *reverse-shortest-path* (RSP) problem asks for computing the smallest value $r^* \geq 0$ such that $G(r^*)$ contains a path from s to t of length at most k.

In this paper we present a general randomized technique that solves the RSP problem efficiently for a large family of geometric objects and distance functions. Using standard, and sometimes more involved, semi-algebraic range-searching techniques, we first give an efficient algorithm for the decision problem, namely, given a value $r \geq 0$, determine whether $G(r)$ contains a path from s to t of length at most k. Next, we adapt our decision algorithm and combine it with a random-sampling method to compute r^*, by efficiently performing a binary search over an implicit set of $O(n^2)$ candidate values that contains r^*.

We illustrate the versatility of our general technique by applying it to a variety of geometric proximity graphs. For example, we obtain (i) an $O^*(n^{4/3})$ expected-time randomized algorithm (where $O^*(\cdot)$ hides polylog(n) factors) for the case where S is a set of pairwise-disjoint line segments in $\mathbb{R}^2$ and $\varrho(e_1, e_2) = \min_{x \in e_1, y \in e_2} \|x - y\|$ (where $\|\cdot\|$ is the Euclidean distance), and (ii) an $O^*(n + m^{4/3})$ expected-time randomized algorithm for the case where S is a set of m points lying on an x-monotone polygonal chain T with n vertices, and $\varrho(p, q)$, for $p, q \in S$, is the smallest value h such that the points $p' := p + (0, h)$ and $q' := q + (0, h)$ are visible to each other, i.e., all points on the segment $p'q'$ lie above or on the polygonal chain T.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases Geometric optimization, proximity graphs, semi-algebraic range searching, reverse shortest path

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.42

Funding *Pankaj K. Agarwal*: Partially supported by NSF grants IIS-1814493, CCF-2007556, and CCF-2223870.
Matthew J. Katz: Partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
Micha Sharir: Partially supported by Grant 260/18 from the Israel Science Foundation.

1 Introduction

Let S be a set of n geometric objects of constant complexity (e.g., points, line segments, disks, ellipses) in $\mathbb{R}^2$, and let $\varrho : S \times S \to \mathbb{R}_{\geq 0}$ be a *distance function* on S. For a parameter $r \geq 0$, we define the *proximity graph* $G(r) = (S, E)$, where $E = \{(e_1, e_2) \in S \times S \mid e_1 \neq e_2,\ \varrho(e_1, e_2) \leq r\}$. If S is a set of n points in $\mathbb{R}^2$ and $\varrho(\cdot, \cdot)$ is the Euclidean metric, then

$G(r)$ is the well-studied *unit-disk* graph [15] (where the "unit" here is $r/2$). Here we study proximity graphs in considerably more general settings. For example, let $S = \{e_1, \ldots, e_n\}$ be a set of n pairwise-disjoint segments in $\mathbb{R}^2$. For a pair of segments, $e_1, e_2 \in S$, we define $\varrho(e_1, e_2) = \min_{x \in e_1, y \in e_2} \|x - y\|$, where $\|\cdot\|$ denotes the Euclidean norm. Equivalently, we can define $\varrho(e_i, e_j)$ as follows: For a value $r \geq 0$, let $B(r)$ be the disk of radius r centered at the origin, and let $K_i(r) \coloneqq e_i \oplus B(r) = \{x + y \mid x \in e_i, y \in B(r)\}$ be the Minkowski sum of e_i and $B(r)$. Then $\varrho(e_1, e_2) = \min\{r \mid K_i(r/2) \cap K_j(r/2) \neq \emptyset\}$. See Figure 1 for an illustration. The *segment-proximity graph* $G(r)$ (over S) has an edge between two segments if they are within distance r of each other. See below for more examples of geometric proximity graphs, and note that the above reformulation allows us to interpret proximity graphs as intersection graphs (the unit-disk example is another instance). We note that we do not require $\varrho(\cdot, \cdot)$ to be symmetric. If ϱ is not symmetric, then $G(r)$ is a directed graph in which a directed edge $e_1 \to e_2 \in E$ if $\varrho(e_1, e_2) \leq r$. See Section 1.2 below for an example.

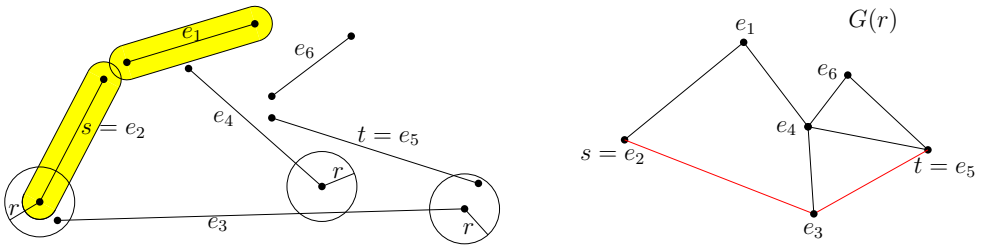


Figure 1 The proximity graph of segments. Left: A possible input consisting of 6 segments, including segments s and t, and a value $r > 0$. The Minkowski sums of e_1 and e_2 with a disk of radius $r/2$ are highlighted. Right: The corresponding proximity graph $G(r)$, where the red edges are those that appear in the shortest path from s to t.

Given S, $s, t \in S$, and an integer $k \geq 1$, the *reverse-shortest-path* (RSP) problem asks for computing the smallest value $r^* \geq 0$ such that $G(r^*)$ contains a path from s to t of length at most k. There are $O(n^2)$ *critical values* of r at which $G(r)$ changes, and our goal is to find the smallest of these critical values for which the proximity graph has the desired property. The RSP problem arises in many applications. For instance, we wish to determine the minimum transmission range for the sensors in a sensor network, such that there is a k-hop transmission path between two given sensors s and t. In this paper we present a general randomized technique for the RSP problem in geometric proximity (and intersection) graphs and apply it to a variety of such graphs.

1.1 Related work

There has been extensive work on understanding the combinatorial structure of geometric proximity graphs (e.g., realizibility of proximity graphs), as well as on developing improved algorithms for a wide range of problems (e.g., shortest paths, vertex covers, independent sets, matchings) on such graphs. Because of numerous applications, to communication networks and other topics, the most widely studied proximity graph is perhaps the unit-disk graph, defined above; see [5, 8, 15, 17, 19] and references therein for a sample of known results on unit-disk graphs. Although a unit-disk graph can have $\Theta(n^2)$ edges, near-linear or subquadratic algorithms are known for many reachability or proximity problems in unit disk graphs, by exploiting the underlying geometry. For example, an $O(n \log n)$ algorithm is known for performing BFS or DFS in unit-disk graphs [10, 12], an $O(n \log^2 n)$ algorithm

for computing shortest paths from a single source in a weighted unit-disk graph (where the weight of an edge (p, q) is $\|p - q\|$ if $\|p - q\| \leq 1$ and ∞ otherwise) [27] (see also [10, 26]), a slightly subquadratic algorithm for computing the diameter of such graphs exactly [12], $O^*(n)$ algorithms[1] for computing the diameter approximately [14] and for computing a spanner [14, 20]. Subquadratic algorithms are known for reachability problems in some other geometric proximity graphs as well, e.g., $O^*(n)$ and $O^*(n^{4/3})$ algorithms for determining whether there exists a path of length at most k between two nodes in disk-intersection or segment-intersection graphs, respectively [6, 13]. Fast algorithms have been developed for computing all pair shortest paths in geometric intersection graphs [13].

In principle, algorithms for geometric proximity graphs could be plugged into the so-called parametric search technique to obtain a subquadratic algorithm for the RSP problem in a geometric proximity graph. However, the difficulty with this approach is that an efficient implementation of the parametric-search technique requires a parallel algorithm for the so-called decision procedure [25], which is not always known. For example, many of the above algorithms use BFS on the resulting graph, but the known BFS algorithms are inherently sequential. Cabello and Jejčič [10] observed that an $O^*(n^{4/3})$ algorithm can be obtained for the RSP problem in a unit-disk graph by using a distance-selection algorithm (given a set S of points in $\mathbb{R}^2$ and an integer $k \geq 1$, return the kth smallest pairwise distance in S; see [3, 22]) to perform a binary search on the $O(n^2)$ pairwise-distances between the input points, and use an efficient BFS algorithm [10] at each step. Wang and Zhao [28] (see also [29]) improved the running time to $O^*(n^{5/4})$ by observing that some of the steps of the BFS algorithm [12] in a unit-disk graph can be parallelized, and they combine parametric search with the distance-selection algorithm. This bound was further improved by Katz and Sharir [21] to $O^*(n^{6/5})$ randomized expected time. We are not aware of any known results on the RSP problem for more general geometric proximity graphs.

We conclude this discussion by noting that the RSP problem has been studied in more general settings, where, given a weighted graph $G = (V, E)$, a pair of nodes $s, t \in V$, and a real parameter W, the goal is to minimize the edge weights as much as possible (subject to various constraints and penalties) so that there is a path in G from s to t of weight at most W. This problem is known to be NP-complete and approximation algorithms are known for a few special cases; see [9, 16, 30] and references therein.

1.2 Our results

There are two main contributions of this paper: First, we describe a general technique for solving the RSP problem in geometric proximity graphs in a fairly general setting. The running time of the algorithm depends on the complexity of the input objects and on the distance function, but it is always subquadratic, even though the proximity graph $G(r)$ may have quadratic size. In all cases described in the paper, this technique yields faster algorithms than those that are based on parametric search because, as mentioned above, the decision algorithms in these cases are hard to parallelize. Second, we consider a wide range of proximity graphs, and develop efficient decision procedures for each case, which are needed to apply our general technique to these instances, and which exploit the geometry of the underlying setup, often in a rather nontrivial manner.

[1] As in the abstract, the $O^*(\cdot)$ notation hides polylog(n) factors.

An overview of our technique. Let S and ϱ be as above. For a pair $e_i, e_j \in S$, we define a predicate $\Pi(e_i, e_j; r)$ that is true if and only if $\varrho(e_i, e_j) \leq r$. Assuming that the objects in S are semi-algebraic sets of constant complexity and that ϱ is also a semi-algebraic function of constant complexity, $\Pi(e_i, e_j; r)$ is a semi-algebraic predicate of constant complexity.[2] Therefore, (e_i, e_j) is an edge in $G(r)$ if and only if $\Pi(e_i, e_j; r)$ holds. Since $\Pi(\cdot)$ is a semi-algebraic predicate of constant complexity, we can efficiently compute a compact representation of $G(r)$, for any given r, as the union of edge-disjoint bipartite cliques (bicliques), with a small overall size of their vertex sets, using either standard halfspace range searching techniques (see [1]), in simpler situations, or (by now standard) semi-algebraic range-searching techniques [2, 4] for more complex setups. With such a compact representation of $G(r)$ at our disposal, the existence of a path from s to t in $G(r)$ of length at most k can be tested in time proportional to the total size of the vertex sets of these graphs by a careful and efficient implementation of BFS on the compact representation of $G(r)$, as in [6]. In other words, this technique yields an efficient decision procedure for the RSP problem, in which r is specified and we want to determine whether $G(r)$ has a path from s to t of length at most k. We describe, in Section 2, the range-searching machinery as well as the efficient BFS implementation on the compact representation of $G(r)$ as the union of bicliques, and show that the expected running time is $O^*(n^{4/3})$.

Next, we obtain a fast optimization algorithm by combining a variant of the decision algorithm with a randomization technique similar to that in [11, 18, 23]. In particular, the technique for computing a compact representation of $G(r)$ is also suitable for computing a compact representation of the graph $G(r, r')$, for $r \leq r'$, in which there is an edge between e_i and e_j if and only if $\varrho(e_i, e_j) \in (r, r']$. By defining the predicate $\Pi(e_1, e_2; r, r')$ to be true if and only if $\varrho(e_i, e_j) \in (r, r']$, we can again use the semi-algebraic range-searching techniques to count the number[3] of *critical values* that lie in the range $(r, r']$. Furthermore, as in [23], the counting procedure is adapted to obtain an efficient randomized procedure for choosing an approximate median of the critical values in a given range $(r, r']$. The expected cost of this procedure is the same (within a polylog(n) factor) as that of the decision procedure. Finally, with this tool available, we solve the optimization problem by a binary search through the critical values of r, using the selection procedure just mentioned to select values for the search, and the decision procedure to guide the search. The overall expected running time for the RSP problem in a segment-proximity graph is $O^*(n^{4/3})$. Unlike some previous application of this randomization techniques, e.g., [23], this approach significantly improves the asymptotic running time of the algorithm in our applications, over what one could have obtained using parametric search.

We remark here that the application of range searching in many cases here is non-trivial and requires new ideas, so that we can use multi-level data structures [1] in as low dimension as possible and obtain better bounds than what we would obtain by a naïve application. For example, a naïve application of semi-algebraic range searching in the running example of segment-proximity graphs would result in a bound of $O^*(n^{8/5})$, as the number of parameters needed to specify an input segment (i.e., its two endpoints) is four, and even simplex (batched) range searching in $\mathbb{R}^4$ results in the aforementioned bound. However, by constructing a

[2] Roughly speaking, a *semi-algebraic set* in $\mathbb{R}^d$ is the set of points in $\mathbb{R}^d$ satisfying a Boolean predicate over a set of polynomial inequalities; the complexity of the predicate and of the set is defined in terms of the number of polynomials involved and their maximum degree. A real-valued function is called semi-algebraic if its graph is a semi-algebraic set. See [7] for details.

[3] In many cases, we define the set of critical values to be a superset of $\{\varrho(e_i, e_j) \mid e_i, e_j \in S\}$, but this does not affect the general approach.

multi-level structure carefully, in which the input objects at each level (features of the actual input objects) can be specified by only two parameters, we succeed in obtaining the bound of $O^*(n^{4/3})$; see Section 2.

Efficient solutions to various specific proximity graphs. We illustrate the versatility of our technique by applying it to a wide range of geometric proximity graphs, as listed below.

Communication graphs of directional antennas. Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$, where each point p_i has a direction $u_i \in \mathbb{S}^1$ associated with it, and let $\delta > 0$ be a *range* parameter. For a parameter $\alpha \in [0, 2\pi]$, let $A_i(\alpha)$ denote the *directional antenna* of range δ located at p_i whose symmetry axis is u_i and which has an opening angle α, i.e., $A_i(\alpha) = \{x \in \mathbb{R}^2 \mid \|x - p_i\| \leq \delta \ \wedge \ \langle u_i, x - p_i\rangle \geq \cos(\alpha/2)\}$. (The case where each antenna has its own range will also be considered.) We consider both asymmetric and symmetric distance functions: For a pair of points $p_i, p_j \in S$ with $\|p_i - p_j\| \leq \delta$, we define $\varrho(p_i, p_j) = \min\{\alpha \in [0, 2\pi] \mid p_j \in A_i(\alpha)\}$ for the asymmetric version, and $\varrho(p_i, p_j) = \min\{\alpha \in [0, 2\pi] \mid p_j \in A_i(\alpha) \ \wedge \ p_i \in A_j(\alpha)\}$ for the symmetric version. We set $\varrho(p_i, p_j) = \infty$ if $\|p_i - p_j\| > \delta$ in both cases. See Figure 2 for an illustration. The *communication graph* $G(\alpha)$ is a directed graph for the asymmetric version and undirected for the symmetric version.

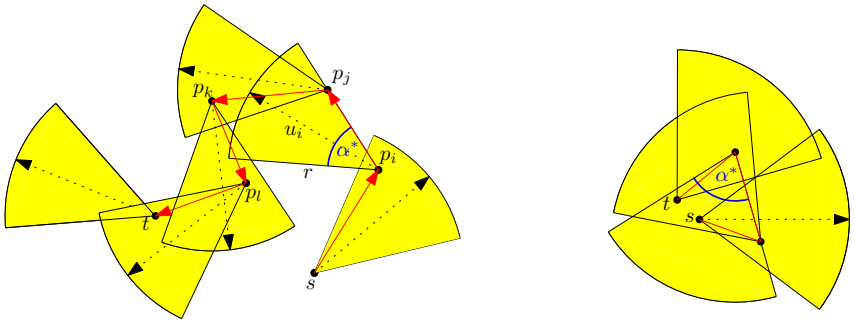


Figure 2 The directional antennas problem. Left: The asymmetric version with $k = 5$. Right: The symmetric version with $k = 3$.

We present (in Section 3.1) a randomized algorithm, with $O^*(n^{4/3})$ expected running time, for the RSP problem in the communication graph of directional antennas, for both symmetric and asymmetric versions. We then modify the algorithm to accommodate the case where each antenna has its own range. The expected running time of this latter algorithm is $O^*(n^{7/5})$.

Polyline-proximity graphs. The polyline-proximity graph is a generalization of the segment-proximity graph. The input consists of a set $\mathcal{T}$ of n pairwise-disjoint polygonal chains (representing roads, say), each of size (i.e., number of vertices) at most some constant l. For a pair $T_i, T_j \in \mathcal{T}$, we define $\varrho(T_i, T_j) = \min_{p \in T_i, q \in T_j} \|p - q\|$. As for segment-proximity graphs, an equivalent formulation is: For a value $r \geq 0$, let $K_i(r) := T_i \oplus B(r)$. Then $\varrho(T_i, T_j) = \min\{r \geq 0 \mid K_i(r/2) \cap K_j(r/2) \neq \emptyset\}$. See Figure 3 for an illustration. We present (in Section 3) a randomized algorithm, with $O^*(n^{4/3})$ expected time, for the RSP problem in polyline-proximity graphs.

Graphs of growing segments. Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$, so that each point p_i is associated with a direction $u_i \in \mathbb{S}^1$. For $r > 0$, let $e_i(r) := p_i + ru_i$ denote the segment of length r that emanates from p_i in direction u_i. (We can also consider cases

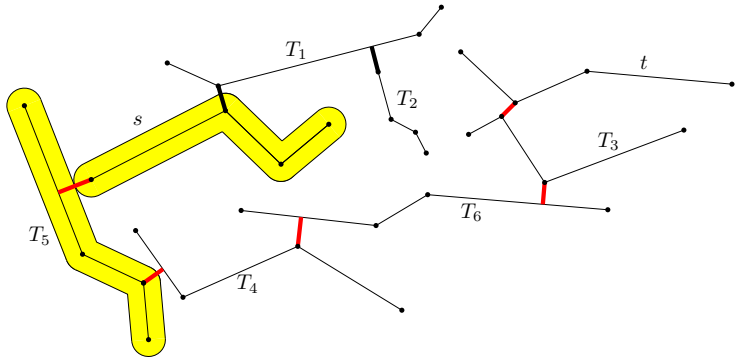


Figure 3 The proximity graph of polylines. In this example, $l = 4$, $k = 5$, and r^* is the distance between s and T_5 (the Minkowski sums of s and T_5, respectively, with a disk of radius $r/2$ are highlighted). The edges of the graph are drawn as either black or red thick segments, where the red ones are those that appear in the path (of length 5) between s and t.

where each $e_i(r)$ grows at a different scale, namely $e_i(r) := p_i + \lambda_i r u_i$, for suitable scalars λ_i.) We now define $\varrho(p_i, p_j) = \min_r\{r \geq 0 \mid e_i(r) \cap e_j(r) \neq \emptyset\}$. See Figure 4 for an illustration of the special case where all the points in P are on the x-axis. We present (in Section 3.2) a randomized algorithm, with $O^*(n^{4/3})$ expected time, for the RSP problem in graphs of growing segments.

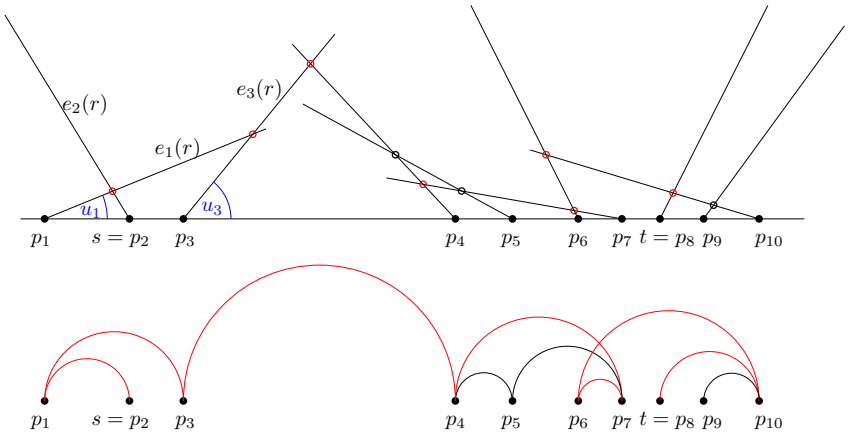


Figure 4 The growing segments problem for an instance where the points are on a line. Top: A possible input consisting of 10 points, including points s and t, and their associated directions, with some common growth value $r > 0$. Bottom: The corresponding graph $G(r)$. The edges in red are those that appear in the shortest path from s to t, which is $(s, p_1, p_3, p_4, p_7, p_6, p_{10}, t)$. These edges correspond to the intersection points, marked in red, between segments. In this example $r^* < r$; more precisely, r^* is the length of the segment with endpoints p_1 and $e_1(r) \cap e_3(r)$.

Variants of growing-segment and segment-proximity graphs. Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$. We generalize the growing-segment graph by considering other shapes (e.g., ellipses) that can be grown around each site p_i. In the simplest version, which we will follow, we assume that each shape has only one degree of freedom of growth. For example, when growing ellipses, we may assume that the ellipse grown at a site p_i is centered at p_i, has fixed directions θ_i and $\theta_i + \frac{\pi}{2}$ of its axes, and has a fixed ratio λ_i between the lengths

of its major and minor axes. We denote this ellipse as $E_i(r)$, where r, the single degree of freedom of growth, is half the length of the major axis. For a pair of points $p_i, p_j \in S$, we now define $\varrho(p_i, p_j) = \min\{r \geq 0 \mid E_i(r) \cap E_j(r) \neq \emptyset\}$. See Figure 5(a) for an illustration.

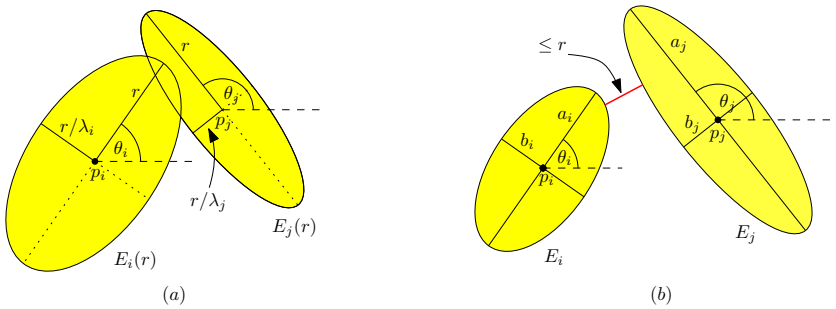


Figure 5 (a) The growing ellipses problem. The ellipse $E_i(r)$ is centered at p_i, the direction of its major axis is θ_i and its length is $2r$; the direction of the minor axis is $\theta_i + \frac{\pi}{2}$ and its length is $2r/\lambda_i$. Since $E_i(r) \cap E_j(r) \neq \emptyset$, (p_i, p_j) is an edge of $G(r)$. (b) The ellipse proximity problem. The length of E_i's major and minor axes are a_i and b_i. Since the distance between E_i and E_j is less than or equal to r, (E_i, E_j) is an edge of $G(r)$.

We present, in Section 3, a randomized algorithm, with $O^*(n^{8/5})$ expected time, for the RSP problem in the growing-ellipse graph. It exploits the fact that each input ellipse has four degrees of freedom (except for the growth parameter r). Here the predicate that determines whether $E_i(r) \cap E_j(r) \neq \emptyset$ is a semi-algebraic predicate of constant complexity, and the associated range-searching problem is with semi-algebraic ranges in four dimensions. The running time here is worse than those in the preceding problems because the associated range-searching problem is in higher dimensions. Currently, we do not see how to reduce the dimension, as we could in the previous problems, and leave this as an open challenge.

Similarly, we can generalize the segment-proximity graph by replacing segments with other shapes. For example, we can have a set $S = \{E_1, \ldots, E_n\}$ of n pairwise-disjoint ellipses in $\mathbb{R}^2$, and we now define $\varrho(E_i, E_j) = \min_{p \in E_i, q \in E_j} \|p - q\|$. See Figure 5(b) for an illustration. As a general ellipse in the plane has five degrees of freedom, our technique solves the RSP problem in the ellipse-proximity graph in $O^*(n^{5/3})$ expected time, by using five-dimensional semi-algebraic range searching data structures. As before, it would be an interesting challenge to improve this bound.

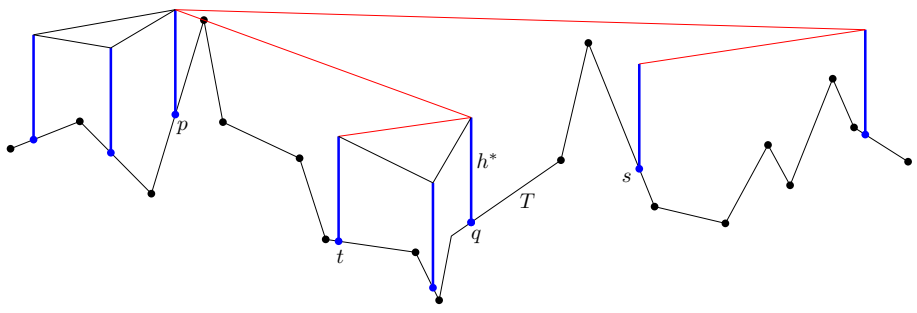


Figure 6 Visibility graph of towers over a terrain. The set Q consists of 8 points, including s and t, and $k = 4$. The height h^* is determined by the pair p, q. The segments between tips of towers correspond to the edges of the visibility graph $V(h^*)$, where the red ones correspond to those that participate in the path, of length 4, between s and t.

Visibility graph of towers over a terrain. Here we face a different setup. We have an x-monotone polygonal line T with n vertices, to which we refer as a 1.5-*dimensional terrain*, and a set Q of m points on T (in general not at its vertices). For a point $q_i \in Q$ and a value $h \geq 0$, let $q_i(h) := q_i + (0, h)$ denote the translate of q_i in the vertical direction by distance h. We can view $q_i(h)$ as the tip of a tower of height h erected on q_i. For a pair $q_i, q_j \in Q$, we now define $\varrho(q_i, q_j)$ to be the minimum value h for which the segment $q_i(h)q_j(h)$ is *visible*, i.e., all the points on the segment lie above or on T. We refer to $G(h)$ as the *visibility graph of towers (of height h) over a terrain*. See Figure 6 for an illustration.

We present (in Section 3.3) a randomized algorithm, with $O^*(n + m^{4/3})$ expected time, for the RSP problem in visibility graphs of towers. As already remarked, this problem is more general than the preceding problems, in that each critical value of h does not depend solely on a pair of points $p, q \in S$ but also on the entire portion of T between p and q. This makes the selection procedure more involved, but we can still make it efficient using a more careful analysis, based on an adaptation of a technique of Agarwal-Varadarajan [6].

2 RSP in Segment-Proximity Graphs

In this section we describe the details of our technique by illustrating it for segment-proximity graphs. Let $S = \{e_1, e_2, \ldots, e_n\}$ be a set of n pairwise-disjoint segments in the plane, let s, t be two segments in S, and let $k \geq 1$ be an integer.

The decision procedure. Consider first the decision problem, in which r is specified and we want to determine whether $r^* \leq r$, i.e., whether $G(r)$ contains a path from s to t of length at most k. We present an algorithm that solves the decision problem in $O^*(n^{4/3})$ time. The solution begins by representing $G(r)$ as the union of bipartite cliques (which are not necessarily edge disjoint). We represent each $e_i \in S$ by the pair of its endpoints, and note that each e_i has four degrees of freedom.

The condition that $\varrho(e_i, e_j) \leq r$ can be expressed as a semi-algebraic predicate of constant complexity, as follows. The distance between two disjoint segments is attained either between two endpoints, one of each segment, or between an endpoint of one segment and the relative interior of the other. We can thus write the condition that a segment $e = uv$ lies at distance at most r from a segment $e' = pq$ as a semi-algebraic predicate $\Pi(e, e'; r)$ which is a disjunction of sub-predicates, each of which is a conjunction of several conditions. The efficiency of the procedure stems from the fact that we can guarantee that each of these sub-conditions involves at most two parameters from the four parameters representing each segment. For example, if the distance between e and e' is attained at their respective endpoints u and p then (i) u and e' lie on different sides of the line ℓ_p^+ that is orthogonal to e' at p, (ii) p and e lie on different sides of the line ℓ_u^+ that is orthogonal to e at u, and (iii) $\varrho(u, p) \leq r$. As another example, if the distance between e and e' is attained at u and a point in the relative interior of e' then (i') u and e' lie on the same side for each of the lines ℓ_p^+ and ℓ_q^+ (the line orthogonal to e' at q), and (ii') $\varrho(u, \ell_{e'}) \leq r$, where $\ell_{e'}$ is the line supporting e'. See Figure 7. The first set of conditions (i)–(iii) are necessary and sufficient for $\varrho(e, e')$ to be attained at p and u (and be at most r), as is easily checked. However, the second set of conditions (i')–(ii') are necessary but not sufficient conditions for $\varrho(e, e')$ to be attained at u and an interior point of e'; see Figure 7(iv). Similar conjunctions arise for all other possible cases. Moreover, whenever one of these conjunctions holds we have $\varrho(e, e') \leq r$.

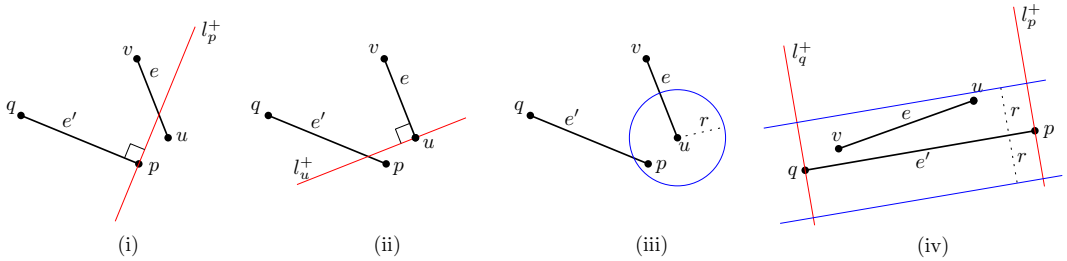


Figure 7 Illustrating conditions (i)–(iii) for the case where the distance between the segments $e = uv$ and $e' = pq$ is attained between their endpoints u and p. (iv) Illustrating conditions (i')–(ii') and the fact that they are not sufficient for $\varrho(e, e')$ to be attained at u and an interior point of e'.

Representing $G(r)$ by bipartite cliques. We turn the problem of constructing $G(r)$ into a batched range searching problem, in which the segments of S serve both as input objects and as queries. The query with a segment e defines the range $Q_e = \{e' \in S \mid \Pi(e, e'; r) \text{ holds}\}$.

We prepare a separate data structure for each of the aforementioned sub-predicates whose disjunction is $\Pi(e, e'; r)$. For example, consider the case where the distance is attained between two endpoints u and p, using the above notation. For sub-condition (i), we prepare a batched halfplane range searching structure, with the endpoints u as input and the halfplanes bounded by the lines ℓ_p^+ and not containing the respective edges e' as ranges. The next level handles sub-condition (ii), in a completely analogous and symmetric manner, and the third level enforces sub-condition (iii), in which the input are the points p and the ranges are disks of radius r around the points u. Similar multi-level structures are constructed for the other sub-predicates in the conjunction. For example, when testing for conditions (i')–(ii'), the last level tests whether the distance from endpoint u to the line $\ell(e')$ supporting e' is at most r. This amounts to testing whether u lies in the strip of width $2r$ centered at $\ell(e')$, and can be implemented using two levels of halfspace range queries (or rather point enclosure queries). Using standard results on multi-level range searching structures (see, e.g., [1]), the whole procedure, with n input objects and n query ranges, takes $O^*(n^{4/3})$ time, and produces $G(r)$ as the union of a collection of (not necessarily edge-disjoint) bipartite cliques, so that the sum of the sizes of their vertex sets is also $O^*(n^{4/3})$.

Finding the path by BFS. With this data available, the rest of the path searching algorithm is relatively easy. We run a BFS through $G(r)$, starting from s. A similar algorithm has been given in [6], and we only briefly sketch the details. At each stage of the BFS we iterate over the points in the current layer, and use the bicliques in which they participate to propagate the path to the next layer. Say we reach a graph $A \times B$, and that the current vertex (i.e., segment) v being processed belongs to A. We then add all the unvisited nodes of B to the next layer of the BFS. When we later process a vertex of B, we repeat the above propagation step, swapping A and B, and then discard the graph completely. In applications where the graph $G(r)$ is directed (see Section 3.1 for examples of such problems), the treatment is somewhat different, with obvious modifications. We keep performing these BFS propagation steps until either t is reached, within the first k layers of the BFS, or we have created k layers without reaching t, or we run out of graphs to process. The overall cost of the BFS is proportional to the overall size of the vertex sets of the graphs, which is $O^*(n^{4/3})$. See [6] for further details.

Solving the optimization problem. The optimization procedure is a variant, or rather an extension, of the distance-selection mechanism of [3], with one notable difference that we replace the parametric search technique used in [3] by a simpler, and as it turns out more efficient, random sampling approach [23]. Our algorithm is based on a procedure, in which, given a threshold value r_0, we want to count the number of critical values that are smaller than or equal to r_0. A value r is *critical* if there exist a pair of segments e_i, e_j in S such that one of the sub-predicates in the disjunction forming $\Pi(e_i, e_j; r)$ holds and its associated critical value is exactly r. Note that the set of critical values is a superset of the set of actual distances between the pairs of segments in S (see, for instance, Figure 7(iv)). This enlargement of the set of critical values is made for technical reasons, to address the fact that the bipartite clique representation of $G(r_0)$ is not edge disjoint; see below for details. (Note that in degenerate configurations the same value r can arise for more than one pair (e_i, e_j), which means that we regard the set of critical values as a multiset.)

Note that we do not count the number of edges of $G(r_0)$, but rather the sum of the weights of these edges, where the *weight* of an edge (e_i, e_j) is the number of satisfied sub-predicates in the disjunction forming $\Pi(e_i, e_j; r_0)$, that is, the number of subgraphs it appears in. We denote this sum of weights as $\mu(G(r_0))$.

We use the quantity $\mu(G(r_0))$ because it is straightforward to compute from the bipartite clique decomposition of $G(r_0)$, in time proportional to the overall size of the vertex sets of the bipartite graphs, that is, in $O^*(n^{4/3})$ time. The quantity $\mu(G(r))$ is (weakly) monotone increasing in r, as easily follows from its definition. As we will see shortly, using $\mu(G(r_0))$ simplifies the optimization procedure and does not affect its performance in any significant manner.

Let $\hat{r}$ be some upper bound on the critical values, over all pairs (e_i, e_j) of segments in S. It is easy to compute $\hat{r}$ in linear time by, e.g., computing the smallest enclosing axis-parallel square of the segments in S. We begin by computing the bipartite clique representation of the graph $G(\hat{r})$. Next, by running a BFS in $G(\hat{r})$, starting from s, as described above, we determine whether $G(\hat{r})$ contains a path between s and t of length $\leq k$. If it does not, then we output that r^* does not exist and stop. Assume therefore, that $G(\hat{r})$ does contain an s-t path of the desired length.

At this point, we know that r^* is in the range $(0, \hat{r}]$, and we narrow down the range using binary search in the set of critical values, where comparisons of the form "$r^* \leq r$?" are resolved by a call to the decision procedure with r. To run the binary search, we define the graph $G(r_1, r_2)$, $r_1 \leq r_2$, whose set of vertices is S and there is an edge between e_i and e_j if one or more of the critical values corresponding to them is in the range $(r_1, r_2]$. Notice that $G(\hat{r}) = G(0, \hat{r})$. Given $r_1 \leq r_2$, we can compute a bipartite clique representation of $G(r_1, r_2)$ in $O^*(n^{4/3})$ time, where the overall size of the vertex sets in this representation is also $O^*(n^{4/3})$, by the following modification of the procedure given above. For each of the multi-level structures used by the preceding procedure, we change its bottom level, so that it collects all pairs of segments e, e' for which the corresponding distance (between two specific endpoints or between an endpoint of one segment and the line supporting the other segment) is in $(r_1, r_2]$. For the case where the considered distance is between two endpoints, we apply range searching with annuli, of the fixed radii r_1 and r_2, instead of the disks used earlier. For the case where the considered distance is between an endpoint and a line, we apply range searching with pairs of strips of width $r_2 - r_1$, obtained as the set differences of the corresponding strips of widths $2r_2$ and $2r_1$. This does not affect the asymptotic performance bounds – it only adds levels to the structures.

We note the following properties: (i) A pair of segments (e, e') in S that has already been encountered while computing $G(r_1)$ may appear again, due to a different pair of features on e and e' being at distance in $(r_1, r_2]$. (ii) Nevertheless, the sum $m = \mu(G(r_1, r_2))$ of the number of edges in the biclique representation of $G(r_1, r_2)$ is exactly $\mu(G(r_2)) - \mu(G(r_1))$, as is implied by the construction.

Assume that we already know that r^* is in the range $I = (r_1, r_2]$. To perform the next binary search step, we compute an approximate median r of the critical values in I, such that the number of critical values in each of the intervals $I_1 = (r_1, r]$ and $I_2 = (r, r_2]$ is, say, at most $2m/3$. This can be done by taking a constant-size random sample of critical values in I (see below for details), and returning the median value r of the sample. It is well-known and easy to see that for a sufficiently large (but still constant) sample size, r is an approximate median as desired with probability greater than $1/2$; see, e.g., [11, 18, 23].

We thus check whether r is indeed an approximate median, and repeat the process with a new random sample, for an expected constant number of times, if it is not.

Once we have found an approximate median r, we run the decision procedure at r to determine whether $r^* \leq r$. If the answer is "YES", we continue with the range I_1, and if it is "NO", we continue with the range I_2. When the number $\mu(G(r_1, r_2))$ of the critical values in the current range is, say, $O(n)$, we compute these values explicitly, by modifying our data structures so that they report critical values instead of counting them, and perform a binary search among them to find r^*.

It remains to describe how to compute a random sample of c critical values in the range $I = (r_1, r_2]$. Consider the representation of $G(r_1, r_2)$ by the union of bicliques, enumerated as $A_1 \times B_1, \ldots, A_\nu \times B_\nu$, where each of the sets A_i, B_i is also enumerated in some arbitrary order. This latter enumeration induces a natural lexicographical order of the edges of each $A_i \times B_i$. Recall that $m = \mu(G(r_1, r_2))$ is the sum $\sum_{i=1}^{\nu} |A_i| \cdot |B_i|$. To pick a random critical value, we draw a random number t in $[1, m]$, and find the index j for which $\sum_{i=1}^{j-1} |A_i| \cdot |B_i| < t \leq \sum_{i=1}^{j} |A_i| \cdot |B_i|$. We then find the t'-th edge of $A_j \times B_j$ in lexicographical order, where $t' := t - \sum_{i=1}^{j-1} |A_i| \cdot |B_i|$, using a similar procedure, and return the associated critical value. (Since we focus on a specific biclique, which corresponds to a single sub-predicate in the disjunction forming Π, the critical value is uniquely defined.)

All this leads to the following summary result.

▶ **Theorem 1.** *The RSP problem in the proximity graph of n pairwise-disjoint segments in the plane can be solved in $O^*(n^{4/3})$ time.*

3 RSP in Other Proximity Graphs

The scheme presented in the previous section applies to many other optimization problems of a similar nature. We discuss in this section and in Appendix A several such problems.

The general approach: A high-level overview of the decision procedure. Generally, the decision procedure for determining whether $G(r)$ has the desired property for a given r, has to construct $G(r)$ as a union of bicliques, which reduces to batched range searching with constant-complexity semi-algebraic ranges in some suitable dimension.

Once $G(r)$ is available, testing for the existence of a path of length at most k is done using BFS, as in the preceding section. The same machinery can also be applied to count the number of critical values that are smaller than or equal to r or that lie in a range $(r, r']$.

The performance of these steps depends on the ambient dimension t, or the dimensions ξ, η, of the parametric spaces that represent the features of the input and query objects that participate in any sub-predicate of the predicate $\Pi(e, e'; r)$ that represents the property that

the distance between objects e and e' is $\leq r$. In the symmetric case, where both the primal space (in which the objects e are stored as points) and the dual space (in which the objects e' are stored as points) have the same dimension t, the recently developed machinery for range searching with semi-algebraic sets [2, 4, 24][4] implies that this cost is $O^*(n^{2t/(t+1)})$, where n is the total number of objects. In the asymmetric case, where the primal and dual parametric spaces have different respective dimensions ξ, η, and we have n objects in the former and m in the latter, the more general bound, developed in Appendix B, applies, and this cost is $O^*(m^{\xi(\eta-1)/(\xi\eta-1)}n^{\eta(\xi-1)/(\xi\eta-1)} + m + n)$. Since we use a multi-level structure, the parameters t, ξ, η may differ at different levels; the overall cost is dominated by the cost of the most expensive level (up to the $O^*(\cdot)$ notation). This machinery uses polynomial partitions. It can be replaced by cuttings, when the parametric dimension t, ξ, or η is at most four or when the predicate only contains linear inequalities.

We now proceed to list the other problems mentioned in the introduction, to which our technique can be applied, and discuss the concrete solution of each of them. Due to lack of space, we delegate some of these problems to Appendix A.

3.1 Reverse shortest paths in communication graphs of directional antennas

Recall that here we are given a set P of n points, including two designated points s and t, a range $\delta > 0$, and an integer $k \geq 1$. Moreover, each point p_i is associated with a direction u_i. The problem is to find the smallest angle α^* such that, if we place at each point p_i a *directional antenna* $A_i(\alpha^*)$ of range δ, symmetry axis u_i and opening angle α^*, then there is a path from s to t of length at most k in the induced communication graph $G(\alpha^*)$. (Alternatively, one can fix the opening angle of the antennas and ask to minimize the range. Our technique works for this case as well.) We can consider either the asymmetric version, where $G(\alpha^*)$ is a directed graph over P, whose edges are all the pairs (p_i, p_j) for which $p_j \in A_i(\alpha^*)$, or the symmetric version, where $G(\alpha^*)$ is undirected and its edges are all the pairs (p_i, p_j) for which both $p_j \in A_i(\alpha^*)$ and $p_i \in A_j(\alpha^*)$. See Figure 2 for an illustration. We present an algorithm for (either version of) this problem that runs in $O^*(n^{4/3})$ time.

Consider, for concreteness, the asymmetric version of the problem (the symmetric version is solved by essentially the same technique). The number of parameters needed to represent an antenna is three, two for p_i and one for u_i (δ is treated as a constant, and α is the growth parameter that we want to minimize). For a fixed pair p_i, p_j, we can write the condition that $p_j \in A_i(\alpha)$ as a semi-algebraic predicate $\Pi(p_i, u_i, p_j; \alpha)$ which is a disjunction of two sub-predicates, each of which is a conjunction of several conditions, each involving only two out of the three parameters representing $A_i(\alpha)$. Namely, $p_j \in A_i(\alpha)$ if and only if (i) p_j lies on the "right" side of both the lines supporting the rays u_i^- and u_i^+, which are obtained from u_i by rotating it by an angle of $\alpha/2$ in clockwise and counterclockwise directions, respectively, and (ii) $\varrho(p_i, p_j) \leq \delta$.

We thus construct two range searching structures, one for each of the sub-predicates of $\Pi(p_i, u_i, p_j; \alpha)$, where each structure consists of three levels, similar to the structures constructed for the segment proximity problem. Finally, we perform n queries, one for each point in P, in each of these structures, to obtain the bipartite clique representation of $G(\alpha)$. As in the segment proximity problem, the overall running time is $O^*(n^{4/3})$, which also bounds

[4] The primal-dual combination of the techniques of [2, 24], which is rather non-trivial, has not been explicitly treated in any previous work, as far as we can tell.

the overall size of the vertex sets in the compact representation of $G(\alpha)$. This follows from the property that at each level of the structure, the features of the antennas that participate in that level have only two degrees of freedom.

Once this representation of $G(\alpha)$ is available, testing for the existence of a path of length k between s and t can be done, similar to Section 2, using a careful implementation of BFS on that graph. We note that in the asymmetric version of the problem the graph is directed. This means that when we process a vertex v that participates in some (now directed) bipartite clique $A \times B$, with $v \in A$, we put in the next layer of the BFS only the still unvisited vertices of B, and discard $A \times B$ from further processing, even though some of its A-vertices might still not have been visited. When such a vertex is visited later, the graph is of no use, as all its B-vertices have already been visited. Except for this difference, the BFS and the decision procedure proceed as before. The bound $O^*(n^{4/3})$ also dominates the overall running time of the entire procedure.

One can also consider the case where each antenna has its own range. Informally, this simply means that we add one extra parameter, namely the range, to the tuple of parameters that specify an input antenna, so each antenna is represented now by four parameters. The case can be treated similarly except that the number of parameters that specify an antenna (ignoring the growth parameter α) is now four, instead of three. The only difference is that now, when performing a query for p_i, we use the disk of radius δ_i around p_i at the bottom level of the structure in which the query is performed. In other words, at the bottom level we apply the algorithm for mixed-dimensional batched range searching with $\xi = 2$ (for the input objects p_j) and $\eta = 3$ (for the query objects $A_i(\alpha)$), which, by Equation (1) of Appendix B, takes a total of $O^*(n^{7/5})$ running time, which also dominates the overall running time of the entire procedure. We thus obtain:

▶ **Theorem 2.** *The RSP problem in the communication network of n directional antennas in the plane can be solved in $O^*(n^{4/3})$ time when all the antennas have the same range, and in $O^*(n^{7/5})$ time when each antenna has a different range. This holds for both symmetric and asymmetric versions of the problem.*

3.2 Reverse shortest paths in intersection graphs of growing segments in the plane

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane, so that each point p_i is associated with a direction u_i. For $p_i \in P$, we denote by $e_i(r)$ the segment of length r that emanates from p_i in direction u_i. Let $G(r)$ be the intersection graph of these segments. That is, $G(r)$ is the graph over P, whose set of edges consists of all the pairs (p_i, p_j) for which $e_i(r) \cap e_j(r) \neq \emptyset$. Let s and t be two designated points of P. In the growing segments problem, we wish to find the smallest value r^* of r, such that there is a path in $G(r^*)$ from s to t of length at most some prescribed value k. See Figure 4 for an illustration of the special case where all the points in P are on the x-axis.

We present an algorithm for this problem that runs in $O^*(n^{4/3})$ time.

As in the directed antennas problem, the number of parameters needed to represent an input object is three, two for p_i and one for u_i (where r is the growth parameter that we wish to minimize). For a fixed pair p_i, p_j, we can write the condition that $e_i(r) \cap e_j(r) \neq \emptyset$ as a semi-algebraic predicate $\Pi(p_i, u_i, p_j, u_j; r)$. Concretely, $e_i(r) \cap e_j(r) \neq \emptyset$ if and only if the endpoints of $e_i(r)$ lie on different sides of the line supporting $e_j(r)$, and vice versa. By trying all possible combinations of sides, we can write $\Pi(p_i, u_i, p_j, u_j; r)$ as a disjunction of several sub-predicates, each of which is a conjunction of four conditions, each requiring some

specific endpoint of one segment to lie on some specific side of the line supporting the other segment. Note that each of these sidedness conditions involves only two out of the three parameters representing each of the objects.

We thus construct several range searching structures of an identical nature, one for each combination of sides. Each structure consists of four levels, each of which is a structure for batched halfplane range searching. We perform n queries, one for each point in P, in each of these structures, to obtain the bipartite clique representation of $G(r)$. Omitting the straightforward details, the overall running time is $O^*(n^{4/3})$, which also bounds the overall size of the vertex sets in the compact biclique representation of $G(\alpha)$. This bound also dominates the overall running time of the entire algorithm, and we get:

▶ **Theorem 3.** *The RSP problem for n growing segments in the plane can be solved in $O^*(n^{4/3})$ time.*

3.3 Visibility graph of towers over a terrain

Recall that here we are given an x-monotone polygonal line T (i.e., a 1.5-*dimensional terrain*) with n vertices, a set Q of m points on T (not necessarily vertices), including two designated points s and t, and an integer parameter $1 \leq k < m$. The problem is to find the minimum height h^* such that if we place a tower of height h^* at each point of Q, then there exists a path of length at most k between the tips of the towers at s and t in the *visibility graph* $G(h^*)$ as defined earlier. See Figure 6 for an illustration. We present an algorithm for this problem that runs in $O^*(n + m^{4/3})$ time.

We note that this problem is just one of several problems whose corresponding decision procedure constructs, as a preliminary stage, a compact representation of the visibility graph $G(h)$, as just defined. This task (i.e., computing the representation of $G(h)$) is different than the analogous tasks in the other problems considered in this paper, since the decision whether an edge (q, q') belongs to $G(h)$ does not depend only on q and q' (and on h) but also on the entire portion of T between q and q'. Nevertheless, adapting a technique proposed by Agarwal and Varadarajan [6], we can compute the desired representation of $G(h)$ in $O^*(n + m^{4/3})$ time.

Constructing a compact representation of $G(h)$

We use a divide-and-conquer approach. Enumerate the vertices of T as $p_1, \ldots, p_n$ from left to right. Partition T into two subpaths $T_L = (p_1, p_2, \ldots, p_{\text{med}})$ and $T_R = (p_{\text{med}}, p_{\text{med}+1}, \ldots, p_n)$, where $\text{med} = \lfloor n/2 \rfloor$. Let $Q_L = Q \cap T_L$ and $Q_R = Q \cap T_R$, and put $m_L = |Q_L|$, $m_R = |Q_R|$. We compute recursively a compact representation of $G(h)_L = \{qq' \mid qq' \in G(h), q, q' \in Q_L\}$ and of $G(h)_R = \{qq' \mid qq' \in G(h), q, q' \in Q_R\}$, and face the problem of computing (a compact representation of) the portion of $G(h)$ consisting of edges that connect a point in Q_L with a point in Q_R.

In the variant of the technique of Agarwal and Varadarajan that we use here, we take each point $q \in Q_L$, and find the upper rightward-directed tangent ray $\tau^+(q)$ from $q(h)$, the tip of the tower of height h at q, to the subchain $(p_k, \ldots, p_{\text{med}})$ of T_L, where p_k is the leftmost vertex of T_L to the right of q. Symmetrically, for each point $q' \in Q_R$, we find the upper leftward-directed tangent ray $\tau^-(q')$ from $q'(h)$ to the subchain $(p_{\text{med}}, \ldots, p_{k'})$ of T_R, where $p_{k'}$ is the rightmost vertex of T_R to the left of q'. Then $q(h)$ and $q'(h)$ are mutually visible if and only if $q'(h)$ lies above $\tau^+(q)$ and $q(h)$ lies above $\tau^-(q')$; see Figure 8.

To find the tangents $\tau^+(q)$, for $q \in Q_L$, we construct a balanced binary tree over the vertices of T_L, and construct, at each node v of the tree, the upper convex hull of the vertices of T_L that are stored at the subtree rooted at v. With some care, the overall cost of these

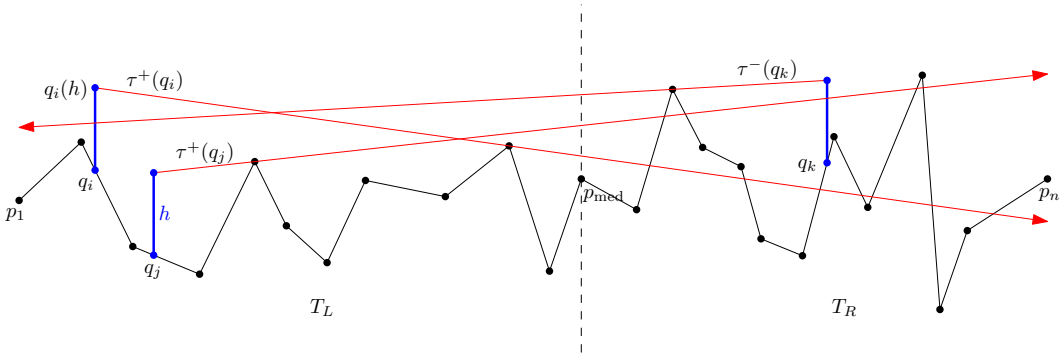


Figure 8 The criterion for visibility between a tower tip from Q_L and a tower tip from Q_R. In this example, $q_i, q_j \in Q_L$ and $q_k \in Q_R$. Moreover, since $q_k(h)$ is above the ray $\tau^+(q_i)$ and $q_i(h)$ is above the ray $\tau^-(q_k)$, q_i and q_k are mutually visible, i.e., $q_iq_k \in G(h)$. On the other hand, since $h(q_j)$ is not above $\tau^-(q_k)$, $q_jq_k \notin G(h)$.

constructions is $O(n \log n)$. Then, for each $q \in Q_L$, we express the portion of T_L to the right of q as the disjoint union of $O(\log n)$ subtrees. We compute the upper tangents from $q(h)$ to each of these hulls, and output the tangent with the largest slope. Again, with some care (including fractional cascading), this takes $O(\log n)$ time for each point of Q_L, for a total cost of $O(m_L \log n)$. A fully symmetric procedure, whose cost is $O(n \log n + m_R \log n)$, computes all the tangents $\tau^-(q')$, for $q' \in Q_R$.

To find the mutually visible pairs in $Q_L \times Q_R$, we use a two-level batched halfplane range searching algorithm, where in the first (resp., second) level the points are those of Q_R (resp., Q_L) and the halfplanes are those that lie above the lines supporting the tangents $\tau^+(q)$, for $q \in Q_L$ (resp., $\tau^-(q')$, for $q' \in Q_R$). Using standard techniques, this can be done in time $O^*(m^{4/3})$. This yields a bipartite clique decomposition of the visible edges in $Q_L \times Q_R$, and the overall collection of these graphs, over all levels of the recursion, is the desired representation of $G(h)$. As is easily checked, the overall size of the vertex sets of these graphs is $O^*(m^{4/3})$, and the overall construction takes $O^*(n + m^{4/3})$ time.

The rest of the algorithm proceeds as in Section 2. We apply BFS to $G(h)$ to determine whether s and t are connected by a path of length at most k, and, depending on the answer, we generate the next value for the binary search. We thus conclude:

▶ **Theorem 4.** *The RSP problem among m towers over a 1.5-dimensional terrain with n vertices can be solved in $O^*(n + m^{4/3})$ time.*

References

1 P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.

2 P. K. Agarwal, B. Aronov, E. Ezra, and J. Zahl. An efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50:760–787, 2021.

3 P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.

4 P. K. Agarwal, J. Matoušek, and M. Sharir. On range searching with semialgebraic sets II. *SIAM J. Comput.*, 42:2039–2062, 2013.

5 P. K. Agarwal, M. H. Overmars, and M. Sharir. Computing maximally separated sets in the plane. *SIAM J. Comput.*, 36(3):815–834, 2006.

6 P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete Comput. Geom.*, 23(2):273–291, 2000.

7 S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics 10. Springer-Verlag, Berlin, 2nd edition, 2006.

8 H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1-2):3–24, 1998.

9 D. Burton and P. L. Toint. On an instance of the inverse shortest paths problem. *Math. Program.*, 53:45–61, 1992.

10 S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom. Theory Appls.*, 48:360–367, 2015.

11 T. M. Chan. On enumerating and selecting distances. *Int. J. Comput. Geom. Appl.*, 11(3):291–304, 2001.

12 T. M. Chan and D. Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *27th Internat. Sympos. on Algorithms and Computation*, pages 24:1–24:13, 2016.

13 T. M. Chan and D. Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019.

14 T. M. Chan and D. Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *J. Comput. Geom.*, 10(2):3–20, 2019.

15 B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

16 T. Cui and D. S. Hochbaum. Complexity of some inverse shortest path lengths problems. *Networks*, 56(1):20–29, 2010.

17 G. D. da Fonseca, V. G. P. de Sá, and C. M. H. de Figueiredo. Shifting coresets: Obtaining linear-time approximations for unit disk graphs and other geometric intersection graphs. *Int. J. Comput. Geom. Appl.*, 27(4):255–276, 2017.

18 M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. *Int. J. Comput. Geom. Appl.*, 2(1):1–27, 1992.

19 A. V. Fishkin. Disk graphs: A short survey. In *First Internat. Workshop on Approximation and Online Algorithms*, volume 2909 of *Lecture Notes in Computer Science*, pages 260–264, 2003.

20 J. Gao and L. Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM J. Comput.*, 35(1):151–169, 2005.

21 M. J. Katz and M. Sharir. Efficient algorithms for optimization problems involving distances in a point set. In arXiv:2111.02052.

22 M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26:1384–1408, 1997.

23 J. Matoušek. Randomized optimal algorithm for slope selection. *Inf. Process. Lett.*, 39(4):183–187, 1991.

24 J. Matoušek and Z. Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54:22–41, 2015.

25 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

26 L. Roditty and M. Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011.

27 H. Wang and J. Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete Comput. Geom.*, 64(4):1141–1166, 2020.

28 H. Wang and Y. Zhao. Reverse shortest path problem for unit-disk graphs. In *17th Internat. Sympos. on Algorithms and Data Structures*, pages 655–668, 2021.

29 H. Wang and Y. Zhao. Reverse shortest path problem in weighted unit-disk graphs. In *16th Internat. Conf. on Algorithms and Computation*, pages 135–146, 2022.

30 J. Zhang and Y. Lin. Computation of the reverse shortest-path problem. *J. Glob. Optim.*, 25(3):243–261, 2003.

A Additional applications

A.1 Reverse shortest paths in proximity graphs of polylines in the plane

This is a generalization of the segment proximity problem. The input consists of a set $\mathcal{T}$ of n pairwise disjoint polylines, each of size (number of vertices) at most some constant l, two designated polylines s and t in $\mathcal{T}$, and an integer parameter $k \leq n$. The problem is to find the smallest r^* such that there exists a path between s and t of length at most k in the graph $G(r^*)$ over $\mathcal{T}$, in which there is an edge between polylines T and T' if and only if the distance between T and T' is at most r^*, where the distance between two polylines T and T' is the length of the shortest segment that connects them. See Figure 3 for an illustration.

The problem can be solved in much the same way as in the case of segments, observing that the distance between two polylines is always attained either between two vertices, one of each polyline, or between a vertex of one of them and the relative interior of an edge of the other. We can therefore write the condition that the distance between two polylines is at most r as the disjoint disjunction of several predicates, each of which has the same structure as in the case of segments. The only difference is that the number of these predicates grows quadratically in l, which does not affect the asymptotic complexity since l is a constant.

Adapting the preceding analysis, we get:

▶ **Theorem 5.** *The RSP problem in the proximity graph of n pairwise-disjoint polylines in $\mathbb{R}^2$, where each polyline is of size at most l, can be solved in $O^*(n^{4/3})$ time, where the constant of proportionality depends (quadratically) on l.*

A.2 Variations of the growing segments and the segment proximity problems

We can generalize the growing segments problem (studied in Section 3.2), and the segment proximity problem (studied in Section 2), by considering various other shapes that can be placed, or grown around each site p_i, such as ellipses. Consider first the growing problem. In the simplest version, we assume that each growing shape has only one degree of freedom of growth. For example, when growing ellipses, we assume, for instance, that the ellipse grown at a site p_i is centered at p_i, has fixed directions θ_i and $\theta_i + \frac{\pi}{2}$ of its axes, and has a fixed ratio λ_i between the lengths of its major and minor axes. We denote this ellipse as $E_i(r)$, where r, the single degree of freedom of growth, is half the length of the major axis. See Figure 5(a) for an illustration.

The problem is to find the smallest value r^* of r for which the graph $G(r^*)$ has a path between two designated points, where the edges of $G(r^*)$ are those pairs (p_i, p_j) for which $E_i(r^*) \cap E_j(r^*) \neq \emptyset$.

Here we need $t = 4$ real parameters to specify an ellipse, namely the coordinates of p_i, λ_i, and θ_i. Our machinery yields an algorithm that runs in $O^*(n^{2t/(t+1)}) = O^*(n^{8/5})$ time.

We can also generalize the proximity problem of Section 2, whose input is a set of segments, by considering other shapes, such as ellipses. That is, each ellipse E_i is given by its center p_i, the lengths of its axes $a_i > b_i$, and the direction of the major axis θ_i, and we assume that these ellipses are pairwise disjoint. Given two designated ellipses s and t in the input set and a parameter k, the goal is to find the smallest value r^* of r, such that there exists a path between s and t of length at most k in the proximity graph $G(r)$ over the ellipses E_i, whose edges are all the pairs (E_i, E_j) of ellipses, such that the distance between E_i and E_j is at most r (this latter property can be expressed as a semi-algebraic predicate of constant complexity, as is easily verified). See Figure 5(b) for an illustration.

Since we need here five real parameters to specify an ellipse, we get that the RSP problem in a proximity graph of n pairwise disjoint ellipses in the plane can be solved in $O^*(n^{5/3})$ time.

In summary, we have:

▶ **Theorem 6.**

(a) *The growing ellipses problem can be solved in $O^*(n^{8/5})$ time.*

(b) *The proximity problem for n pairwise disjoint ellipses in the plane can be solved in $O^*(n^{5/3})$ time.*

Note that this is one application where we have to apply full-blown semi-algebraic range searching machinery. We were mostly able to bypass this using multi-level data structures, each of which had to deal only with halfspace range searching. Nevertheless, other, more involved, applications of our technique will have to use this more general machinery. For example, handling other shapes, for both problems, can be handled in much the same way, using semi-algebraic range searching, whose performance depends on the number of parameters needed to specify an object. Other setups include extensions of some of the problems considered in this work to three (or higher) dimensions. For example, the predicate needed for the segment-proximity problem in $\mathbb{R}^3$ will lead to semi-algebraic ranges (namely, cylinders).

B Mixed-dimensional batched range searching

For the sake of completeness we sketch an algorithm for handling mixed-dimensional batched range searching (in a single level of a multi-level range searching structure).

In general, we have two sets A, B of n and m objects, respectively, where the objects in A have ξ degrees of freedom and those in B have η degrees of freedom, for suitable constant parameters ξ and η. Each pair $a \in A$ and $b \in B$ defines a constant-degree semi-algebraic predicate $\Pi(a, b)$. The goal is either to determine whether there exists a pair $(a, b) \in A \times B$ such that $\Pi(a, b)$ is true, or to count the number of such pairs, or to represent all of them in some compact form, or to report all of them.

Consider the first, simplest task; the other tasks are solved similarly (in the reporting version we also incur an additive cost which is linear, or nearly linear, in the output size). The solution uses a primal-dual approach. The primal space is $\mathbb{R}^\xi$, each object of A is stored as a point, and each object $b \in B$ is stored as the semi-algebraic range $Q_b = \{a \in A \mid \Pi(a, b) \text{ is true}\}$. The dual space is $\mathbb{R}^\eta$, each object of B is stored as a point, and each object $a \in A$ is stored as the semi-algebraic range $Q_a^* = \{b \in B \mid \Pi(a, b) \text{ is true}\}$. Although the problem is symmetric, we view for concreteness the objects of A as data and those of B as queries. In the primal we use a modified version of the techniques of Agarwal, Matoušek and Sharir [4] and of Matoušek and Patáková [24]. These techniques essentially construct a hierarchical polynomial partition in $\mathbb{R}^\xi$ until one reaches nodes with input size x (that we will shortly determine). The overall number of nodes is $O(n/x)$, the preprocessing time to construct the structure is $O^*(n)$, and a query with a range Q_b reaches $O^*((n/x)^{1-1/\xi})$ leaf nodes. At each leaf node we pass to the dual η-dimensional space, and apply the point-enclosure technique of Agarwal et al. [2], which processes the x dual ranges Q_a^* into a data structure of size $O^*(x^\eta)$, in time $O^*(x^\eta)$, so that a point enclosure query (with a dual point $b \in B$) takes $O(\log x)$ time.

The overall storage used by the structure, and the time to construct it, are both $O^*((n/x) \cdot x^\eta) = O^*(nx^{\eta-1})$, and a query takes $O^*((n/x)^{1-1/\xi} \log x) = O^*((n/x)^{1-1/\xi})$ time. We put $nx^{\eta-1} = \sigma$, for a suitable storage parameter σ that we will shortly determine, so $x = (\sigma/n)^{1/(\eta-1)}$. The query time then becomes

$$O^*((n/x)^{1-1/\xi}) = O^*((n^{\eta/(\eta-1)}/\sigma^{1/(\eta-1)})^{1-1/\xi}).$$

The overall cost of answering m queries, plus the preprocessing cost, is thus

$$O^*(mn^{\eta(\xi-1)/(\xi(\eta-1))}/\sigma^{(\xi-1)/(\xi(\eta-1))} + \sigma).$$

We now balance the two terms by choosing $\sigma = m^{\xi(\eta-1)/(\xi\eta-1)} n^{\eta(\xi-1)/(\xi\eta-1)}$, and conclude that the overall running time of the algorithm is

$$O^*\left(m^{\xi(\eta-1)/(\xi\eta-1)} n^{\eta(\xi-1)/(\xi\eta-1)}\right). \tag{1}$$

On the Complexity of Rainbow Vertex Colouring Diametral Path Graphs

Jakob Dyrseth
University of Bergen, Norway

Paloma T. Lima
IT University of Copenhagen, Denmark

Abstract

Given a graph and a colouring of its vertices, a rainbow vertex path is a path between two vertices such that all the internal nodes of the path are coloured distinctly. A graph is rainbow vertex-connected if between every pair of vertices in the graph there exists a rainbow vertex path. We study the problem of deciding whether a given graph can be coloured using k or less colours such that it is rainbow vertex-connected. Note that every graph G needs at least $\mathrm{diam}(G) - 1$ colours to be rainbow vertex connected.

Heggernes et al. [MFCS, 2018] conjectured that if G is a graph in which every induced subgraph has a dominating diametral path, then G can always be rainbow vertex coloured with $\mathrm{diam}(G) - 1$ many colours. In this work, we confirm their conjecture for chordal, bipartite and claw-free diametral path graphs. We complement these results by showing the conjecture does not hold if the condition on *every* induced subgraph is dropped. In fact we show that, in this case, even though $\mathrm{diam}(G)$ many colours are always enough, it is NP-complete to determine whether a graph with a dominating diametral path of length three can be rainbow vertex coloured with two colours.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases rainbow vertex colouring, diametral path graphs, interval graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.43

1 Introduction

Rainbow colouring is a concept that was first introduced by Chartrand et al. [2] in 2008, to study connectivity in edge coloured graphs. A rainbow colouring of a graph is a colouring of its edges such that, between any two vertices, there exists a path no two edges of which have the same colour. The study of this topic garnered a lot of attention and led Krivelevich and Yuster [7] to define a variation which applied to vertex colourings, instead of edge colourings.

In this variant, we are concerned with *rainbow vertex paths*, which are paths between two vertices in which all the *internal* nodes have different colours. We say a graph is *rainbow vertex connected* if between every pair of vertices there exists a rainbow vertex path. Resulting from this we have the decision problem Rainbow Vertex Colouring (RVC) which takes as input a graph G and some integer k with the task to decide if G can be coloured using k or less colours such that it is rainbow vertex-connected. The minimum number of colours needed to make a graph G rainbow vertex-connected is denoted by $\mathbf{rvc}(G)$, and it is called the rainbow vertex-connection number of G.

It is easy to see that any rainbow vertex colouring needs at least $\mathrm{diam}(G) - 1$ many colours, where $\mathrm{diam}(G)$ denotes the diameter of the graph G. On the other hand, the size of a minimum connected dominating set provides an upper bound on **rvc**. Indeed, if we give each vertex of a connected dominating set D a distinct colour, and an arbitrary colour to the remaining vertices, we obtain a rainbow vertex colouring of G, as between any two pairs of vertices there is a path whose all internal vertices lie in D. This relation led Heggernes et

al. [6] to propose the following conjecture on diametral path graphs. A *diametral path* is a shortest path whose length is equal to the diameter of the graph. A graph is a *diametral path graph* if every induced subgraph contains a dominating diametral path.

▶ **Conjecture 1** (Heggernes et al. [6]). *If G is a diametral path graph, then* $\mathbf{rvc}(G) = \mathrm{diam}(G) - 1$.

To support their conjecture, they showed it to be true for two subclasses of diametral path graphs, namely bipartite permutation graphs and interval graphs. Later on, the above conjecture was also shown to be true for permutation graphs [11].

Our results

We show the conjecture to be true for chordal, bipartite and claw-free diametral path graphs. In fact, we show a slightly more general result, as our algorithms require only the input graph to have a dominating diametral path, instead of every induced subgraph.

▶ **Theorem 2.** *If G is a chordal, bipartite or claw-free graph that contains a dominating diametral path, then* $\mathbf{rvc}(G) = \mathrm{diam}(G) - 1$. *Moreover, a rainbow vertex colouring can be computed in time that is polynomial on the size of G.*

Note that our results for chordal graphs, bipartite graphs and claw-free graphs with a dominating diametral path generalize the results for interval graphs, bipartite permutation graphs and unit interval graphs, respectively, obtained by Heggernes et al. [6]. It is also interesting to notice that these results contrast with the fact that Rainbow Vertex Colouring is NP-complete on bipartite graphs for any fixed $k \geq 3$ [6, 9] and on chordal graphs for any fixed $k \geq 2$ [6].

We complement these results by showing that Conjecture 1 does not hold if we drop the condition on every induced subgraph. We give an example of a graph with a dominating diametral path that requires $\mathrm{diam}(G)$ many colours to be rainbow vertex connected. Moreover, we show that, in this graph class, even though $\mathrm{diam}(G)$ many colours always suffice, already when $\mathrm{diam}(G) = 3$, it is NP-complete to decide whether $\mathbf{rvc}(G) = 2$.

▶ **Theorem 3.** *If G is a graph that has a dominating diametral path, then* $\mathbf{rvc}(G) \leq \mathrm{diam}(G)$.

▶ **Theorem 4.** *Let G be a graph with a dominating diametral path of length three. It is* NP*-complete to decide whether* $\mathbf{rvc}(G) \leq 2$.

A summary of these results is presented in Figure 1. In the figure we shorten dominating diametral path with the abbreviation DDP.

A variant of the RVC problem concerned with shortest paths was introduced by Li et al. [10]. We say a graph is *strongly rainbow vertex-connected* if between every pair of vertices there exists a shortest path which is also rainbow vertex path. The resulting decision problem is called Strong Rainbow Vertex Colouring (SRVC) and the analogous parameter is denoted by $\mathbf{srvc}(G)$. Every strong rainbow vertex colouring is also a rainbow vertex colouring, thus $\mathbf{rvc}(G) \leq \mathbf{srvc}(G)$. Heggernes et al. [6] showed that if G is a unit interval graph, then $\mathbf{srvc}(G) = \mathrm{diam}(G) - 1$. Their result for interval graphs, however, applied only to the standard variant of the problem. In this work we strengthen their result by showing the following.

▶ **Theorem 5.** *If G is an interval graph, then* $\mathbf{srvc}(G) = \mathrm{diam}(G) - 1$ *and an optimal strong rainbow vertex colouring can be found in time that is linear in the size of G.*

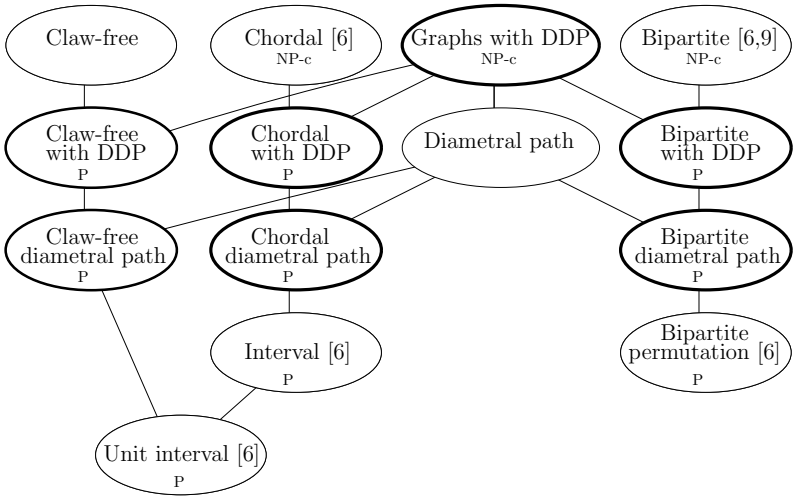


Figure 1 An overview of the complexity of RVC restricted to the graph classes considered here. A thick line represents results obtained in this work. A line connecting two graph classes means the lower one is a subclass of the upper one.

This paper is organized is follows. In Section 2 we define the notation and define basic concepts that will be used throughout the text. In Section 3 we prove Theorems 3 and 4, and provide the example of a graph with a dominating diametral path that requires $\text{diam}(G)$ many colours in any rainbow vertex colouring. The proof of Theorem 2 is split in Sections 4, 5 and 6. Finally, in Section 7, we consider interval graphs and prove Theorem 5. Due to space constraints, statements marked with ♣ have their proofs deferred to the full version of this work.

Other related work

As mentioned above, RVC is NP-complete for any fixed $k \geq 2$ on chordal graphs (in fact, even on split graphs) and any fixed $k \geq 3$ on bipartite graphs [6]. These results also apply to SRVC. In terms of positive results, both problems are polynomial-time solvable on bipartite permutation graphs, unit interval graphs, and block graphs [6], on planar graphs for every fixed k [8, 5], and on split strongly chordal graphs [11]. RVC was furthermore showed to be polynomial-time solvable on interval graphs [6] and on permutation graphs and powers of trees [11].

Although rainbow vertex colouring may seem abstract and almost purely theoretical it does have some quite interesting applications to real world problems especially in terms of encryption and data security. One example of this mentioned in [4] can be seen in onion routing, a technique for anonymously browsing online. In onion routing the goal is to prevent an adversary from knowing what sites you are connecting to. The way this is achieved is by sending the message through a path of intermediaries before accessing the server you requested. This message will be sent using multiple layers of encryption [12], where each node of the path can only decrypt a single layer of this encryption. Assigning decryption keys to the network draws parallels to rainbow vertex colouring.

2 Preliminaries

For basic graph terminology we refer the reader to [1]. We denote a set of consecutive integers from 1 to k as $[k]$. Given some path $P = ux_1x_2...x_mv$ the vertices $x_1x_2...x_m$ are called the *internal vertices* of P. A path is a rainbow vertex path if for every pair of internal vertices x_i, x_j they are coloured such that $c(x_i) \neq c(x_j)$. For some graph G if $\text{diam}(G) = 2$, then $\mathbf{rvc}(G) = \mathbf{srvc}(G) = 1$. The reason each node of the G can have the same colour is that between every pair of vertices there will always exist some path such that there is only one internal node.

Breadth-first search (BFS) is an algorithm for traversing graph structures. The algorithm works by exploring the graph in layers, starting from some arbitrary root node v_0. The layers are indicated L_i, where L_0 is the layer of the root vertex. Thus all vertices in layer $v_i \in L_i$ of the BFS-structure have a distance of i to v_0. Throughout this text we will often, for the sake of convenience, indicate a vertex v belonging to some layer L_i of a BFS-structure with v_i.

A *diametral path* is a shortest path whose length is equal to $\text{diam}(G)$. A dominating diametral path is a path $P = x...y$ such that $\text{dist}(x, y) = \text{diam}(G)$ and the set $V(P) = \{x, ..., y\}$ is a dominating set of G.

We say a graph is a *diametral path graph* if every connected induced subgraph has a dominating diametral path. It has been shown that the existence of a dominating diametral path in a graph can be checked and the path itself can be computed in $\mathcal{O}(n^3m)$ time [3]. Throughout this text we will often run a BFS-search on one end of the dominating diametral path. We denote the vertices of the diametral path p_i, where p_i belongs to layer L_i of the BFS-tree. We will also often refer to vertices as being *dominated to the right*, *dominated to the left* or *dominated in layer*. For some vertex $v \in L_i$ we say it is dominated to the right if $(v, p_{i+1}) \in E(G)$, we say it is dominated to the left if $(v, p_{i-1}) \in E(G)$ and we say it is dominated in layer if $(v, p_i) \in E(G)$.

A graph is *chordal* if it contains no induced cycle of a length which is greater than three. A well known subclass of the chordal graph is that of *interval graphs*. A graph G is an interval graph if its vertices can be associated with a set of intervals in the real line in such a way that two vertices are adjacent if and only if the corresponding intervals intersect. The set of intervals associated with a graph G is called an *interval model for* G. Interval graphs is a subclass of the chordal graphs with a dominating diametral path.

If the vertices of graph can be divided into two disjoint subsets such that each of the subsets are independent the graph is said to be a *bipartite graph*. A *claw* is a bipartite graph on four vertices a, b, c and d, with edges (a, b), (a, c) and (a, d). A graph is *claw-free* if it contains no induced *claw*. When referring to claws throughout this text we will write them $\{abcd\}$ such that $\{b, c, d\} \subset N(a)$, or said in another way the center of the claw will be written first.

Basic properties of graphs with a dominating diametral path

We now present some basic properties of the layers of a BFS performed on a graph with a dominating diametral path. These properties will come in useful in all the algorithms presented in this work. Let G be a graph containing a dominating diametral path to which a BFS is performed on one end of the diametral path. We let $k = \text{diam}(G)$ thus the BFS will have $k + 1$ layers. For each layer L_i, for $0 \leq i \leq k$, we let p_i indicate the vertex of the dominating diametral path in layer i, where p_0 is the root vertex. The following is an easy property of a BFS.

▶ **Observation 6.** *A vertex $x \in L_i$ can only have neighbours in L_{i-1}, L_i and L_{i+1}.*

We will now introduce two new definitions which are used repeatedly throughout the text. Recall that for convenience of notation we indicate vertices by the layer they belong to, i.e. a vertex $v \in L_i$ is denoted v_i.

▶ **Definition 7.** *We say a path between $x \in L_i$ and $y \in L_j$ is a direct path if the length of the path is $j - i$, that is, it intersects every layer from L_i to L_j only once.*

▶ **Definition 8.** *We say a path between two vertices $x \in L_i$ and $y \in L_j$ zig-zags in L_l if it is of the following structure: $x...v_{l-1}v_l v'_{l-1}v'_l...y$, where $v_{l-1} \neq v'_{l-1}$, $v_l \neq v'_l$ and the paths x to v_l and v'_l to y are direct paths.*

This next observation comes naturally as a result of p_0 being the root vertex of the BFS.

▶ **Observation 9.** *The root vertex p_0 of the BFS will have a direct path to every vertex in the graph.*

The following claims concerns colourings in which the vertices of the diametral path are coloured in a specific way.

▷ Claim 10 (♣). Let $c : V \to [k-1]$ be a colouring for G in which each vertex p_i of the diametral path is coloured $c(p_i) = i$, for $1 \leq i < k$. Furthermore, $c(p_0) = k-1$ and $c(p_k) = 1$. The vertices not in the diametral path are coloured with arbitrary colours in $[k-1]$. For pairs of vertices $x \in L_i$ and $y \in L_j$, where $2 < i \leq j \leq k$, a rainbow path can always be found between x and y whose all internal vertices are contained in the diametral path.

The consequence of this claim is that when looking at pairs of vertices $u \in L_i$ and $v \in L_j$, where $0 \leq i \leq j \leq k$ if the colouring of the diametral path is equal to c then we only have to examine cases when $i < 3$. This next claim ensures that when $i = j$ for two vertices we only have to examine the case when the diameter of the graph is three.

▷ Claim 11 (♣). Let $c : V \to [k-1]$ be a colouring for G in which each vertex p_i of the diametral path is coloured $c(p_i) = i$, for $1 \leq i < k$. Furthermore, $c(p_0) = k-1$ and $c(p_k) = 1$. The vertices not in the diametral path are coloured with arbitrary colours in $[k-1]$. For a pair of vertices $x, y \in L_i$ in G there will always be a rainbow path connecting them, unless $\mathrm{diam}(G) = 3$.

3 Graphs with a dominating diametral path

In this section, we first prove Theorem 3, stating that $\mathrm{diam}(G)$ many colours are always enough to rainbow vertex colour a graph G that has a dominating diametral path. We then proceed to give an example of one such graph that in fact *needs* $\mathrm{diam}(G)$ many colours to be rainbow vertex connected. To round off this section, we present the proof of Theorem 4, stating that even when $\mathrm{diam}(G) = 3$, testing whether two colours are enough to make G rainbow vertex connected is an NP-complete problem.

▶ **Theorem 3.** *If G is a graph that has a dominating diametral path, then* $\mathbf{rvc}(G) \leq \mathrm{diam}(G)$.

Sketch of the proof (♣). To construct the colouring $c : V \to [k]$, we run a BFS on some end-point of the diametral path. We let k equal the number of layers of the tree excluding p_0 and let p_i denote the vertex of the diametral in layer L_i. We assign the colouring for $v \in L_i$: $c(v) = k$ if $i = 0$ or $i = k$; and $c(v) = i$ otherwise. For a proof that G is rainbow vertex coloured under c we refer the reader to the full version of this work. ◀

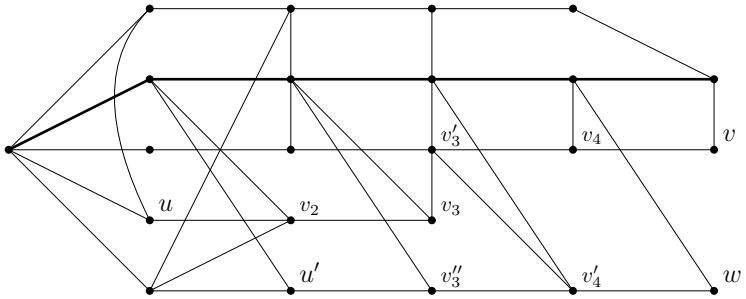


Figure 2 The graph has a diameter of 5, while also needing 5 colours to be rainbow vertex-coloured. The dominating diametral path is highlighted.

We now give an example of a graph with a dominating diametral path that needs $\mathrm{diam}(G)$ many colours to be rainbow vertex connected. The graph is depicted in Figure 2. It should be noted that this does not disprove the conjecture as the example graph in Figure 2 is not a diametral path graph. It is rather a graph *containing a dominating diametral path*. The distinction can be seen by observing that the graph in the figure contains two induced cycles of a length greater than six, which is a forbidden structure in diametral path graphs [3].

▶ **Proposition 12.** *If G is the graph shown in Figure 2, then* $\mathbf{rvc}(G) \geq \mathrm{diam}(G)$

Proof. Note that the diameter of the graph in Figure 2 is five. Assume for a contradiction that G can be rainbow vertex coloured with four colours. First we highlight the three paths of interest in the graph: $uv_2v_3v_3'v_4v$, $uv_2v_3v_3'v_4'w$ and $u'v_3''v_4'v_3'v_4$. What makes these paths interesting for our purposes is that they are the only paths of length less than or equal to the diameter between their two endpoints. Thus, if there exists a rainbow vertex colouring for the graph using 4 colours all of these paths must be rainbow coloured simultaneously. We begin by giving the uv-path an arbitrary rainbow colouring, the internal vertices $v_2v_3v_3'v_4$ receive the colouring $1, 2, 3, 4$ such that $c(v_2) = 1$, $c(v_3) = 2$, $c(v_3') = 3$ and $c(v_4) = 4$. For the uw-path we see that v_4' must be coloured 4 for it to be rainbow, but this means that the $u'v$-path will not be rainbow as there are two nodes of its internal vertices which are coloured 4. Thus we see that there is no way for all of these paths to be rainbow at the same time. ◀

To conclude this section, we prove the following.

▶ **Theorem 4.** *Let G be a graph with a dominating diametral path of length three. It is* NP*-complete to decide whether* $\mathbf{rvc}(G) \leq 2$.

Proof. We give a reduction from Hypergraph 2-Colouring. In this problem, we are given a hypergraph $\mathcal{H}$ and we want to colour its vertices in such a way that no hyperedge is monochromatic. The Hypergraph Colouring problem is known to be NP-hard for any fixed number of colours $k \geq 2$.

Given an instance $\mathcal{H}$ of Hypergraph 2-Colouring with $V(\mathcal{H}) = [n]$ and $E(\mathcal{H}) = \{e_1, \ldots, e_m\}$, we construct an instance (G, k) of Rainbow Vertex Colouring, with $k = 2$, as follows. The graph G has four vertices $p_0, \ldots, p_3$ forming an induced path; two pendant vertices a_1 and a_2 attached to p_1 and p_2, respectively; two independent sets S_1 and S_2, where each vertex of S_i corresponds to a hyperedge of $\mathcal{H}$; and a clique C, each vertex of which corresponds to an element of $V(\mathcal{H})$.

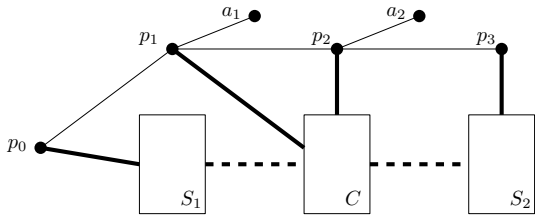


Figure 3 The graph constructed in the proof of Theorem 4. Thick lines indicate a vertex is complete to the set. Dashed lines indicate the two sets are attached.

We now give some definitions to describe the edge set of G. We denote by s_{iq} the vertex of S_i corresponding to the hyperedge e_q of $\mathcal{H}$, and by v_q the vertex of C corresponding to the element $q \in V(\mathcal{H})$. We say a set S_i is *attached* to C if for every $1 \leq q \leq m$, s_{iq} is adjacent to all the vertices of C corresponding to elements of e_q. That is, $(s_{iq}, v_\ell) \in E(G)$ if and only if $\ell \in e_q$ in $\mathcal{H}$. In words, the adjacencies between S_i and C encode the structure of $\mathcal{H}$.

The edges of G can then be described as follows: $p_0p_1 \dots p_3$ is an induced path; $(p_i, a_i) \in E(G)$ for $i \in \{1, 2\}$; C is a clique; S_1 and S_2 are attached to C; p_0 dominates S_1; p_1 and p_2 dominate C; and p_3 dominates S_2.

See Figure 3 for an illustration of the structure of G.

▷ Claim 13 (♣). If $\mathcal{H}$ is 2-colourable, then G has rainbow vertex colouring with two colours.

▷ Claim 14. If G has a rainbow vertex colouring with two colours, then $\mathcal{H}$ is 2-colourable.

Proof. Let c be a rainbow colouring for G with two colours. Since p_1 and p_2 are cut vertices in G, they must have distinct colours under c. So we may assume $c(p_1) = 1$ and $c(p_2) = 2$. To retrieve a 2-colouring for $\mathcal{H}$ from c we consider the restriction of c to C. That is, we define a 2-colouring for $\mathcal{H}$ as follows: for every $q \in V(\mathcal{H})$, $\phi(q) = c(v_q)$. To see that $\mathcal{H}$ has no monochromatic edge of colour 1 under ϕ, consider rainbow paths between a_1 and the vertices of S_2. Since $N(S_2) = C \cup \{p_3\}$ and $\text{dist}(a_1, p_3) = 3$, every such rainbow path has to use a vertex from C. Note also that this path must contain p_1, and $c(p_1) = 1$. If there exists a monochromatic hyperedge e_j with colour 1 under ϕ, then there is no rainbow path between s_{2j} and a_1, a contradiction. Hence ϕ has no monochromatic hyperedge of colour 1. By a similar argument, we see that $\mathcal{H}$ has no monochromatic edge of colour 2. Indeed, since $N(S_1) = C \cup \{p_0\}$ and $\text{dist}(p_0, a_2) = 3$, any path between a vertex of S_1 and a_2 must contain p_2 and a vertex from C. Since $c(p_2) = 2$ and S_1 is attached to C, the claim follows. ◁

To conclude the proof of the theorem, note that $\text{diam}(G) = 3$ and that $p_0p_1p_2p_3$ is a dominating diametral path in G. ◀

4 Bipartite graphs with a dominating diametral path

In this section, we investigate the complexity of RVC on bipartite graphs with a dominating diametral path, and prove Conjecture 1 for this graph class. Recall that for bipartite graphs RVC has already been proven to be NP-complete for any fixed $k \geq 3$. The result we achieve for this graph class is a direct improvement on the earlier result of Heggernes et al. [6] on bipartite permutation graphs, a known subclass of the bipartite graphs with a dominating diametral path.

▶ **Theorem 15 (♣).** *If G is a bipartite graph with a dominating diametral path, then* $\mathbf{rvc}(G) = \mathrm{diam}(G) - 1$ *and the corresponding rainbow vertex colouring can be found in time that is polynomial in the size of G.*

5 Chordal graphs with a dominating diametral path

In a similar vein to the previous section, we are now going to consider the complexity of RVC on chordal graphs with a dominating diametral path. In particular, we show that Conjecture 1 is true for chordal diametral path graphs. Recall that RVC is NP-complete on chordal graphs for $k \geq 2$. Our algorithm for chordal graphs with a dominating diametral path generalizes the existing result for interval graphs of Heggernes et al. [6]. We also point out that an example of a graph that is a chordal diametral path graph and *not* an interval graph is the 3-sun, a graph on vertex set $\{a, b, c, d, e, f\}$ in which $\{a, b, c\}$, $\{b, d, e\}$ and $\{c, d, f\}$ induce triangles.

▶ **Theorem 16.** *If G is a chordal graph with a dominating diametral path, then* $\mathbf{rvc}(G) = \mathrm{diam}(G) - 1$ *and the corresponding rainbow vertex colouring can be found in time that is polynomial in the size of G.*

Proof. Let $G = (V, E)$ be a chordal graph with a dominating diametral path. We run a BFS search on one of the endpoints of the diametral path which we will call p_0. We let k denote the number of layers of the BFS tree and let p_i be the vertex of the diametral path in L_i where $0 \leq i \leq k$. To construct a rainbow colouring $c : V \to [k-1]$ for G we assign the colouring for $v \in L_i$:

$$c(v) = \begin{cases} k-1 & \text{if } i = 0 \\ 1 & \text{if } i = k \text{ or if } i = 2 \text{ and for some } v_2 \in (N(v) \cap L_2) : \mathrm{dist}(v_2, p_k) = k \\ i & \text{otherwise.} \end{cases}$$

Once a BFS is performed on a chordal graph with a dominating diametral path, we will see that some special edges must exist within a layer, otherwise we would have long induced cycles. We will formulate some of these properties before delving into the details of the proof as they will be of great use.

▷ Claim 17 (♣). If we have a path $P_3 = abc$ such that $a, c \in L_i$ and $b \in L_{i+1}$ then the edge (a, c) must be in $E(G)$.

▶ **Observation 18 (♣).** *A vertex $x \in L_i$ in G cannot only be dominated to the right.*

▷ Claim 19 (♣). If we have some $P_4 = abcd$ where $a, d \in L_i$ and $b, c \in L_{i+1}$, then the edge (a, d) must be in $E(G)$.

▷ Claim 20 (♣). If a vertex $x \in L_i$, where $x \neq p_i$, has a neighbour $y \in L_{i+1}$ then x must be dominated in layer.

▶ **Observation 21 (♣).** *A vertex $x \in L_2$ with a direct path to p_k will always be coloured 2.*

▷ Claim 22 (♣). There will never exist a direct path between some vertex $x \in L_2$ coloured 1 and some vertex $y \in L_i$, $2 < i \leq k$, where y is only dominated in layer.

▷ Claim 23 (♣). If for a pair of vertices $x \in L_i$ and $y \in L_j$ where $j > i$ and $\mathrm{dist}(x, y) = j - i + 1$, then there will always exist some path between x and y such that it repeats layers in L_i.

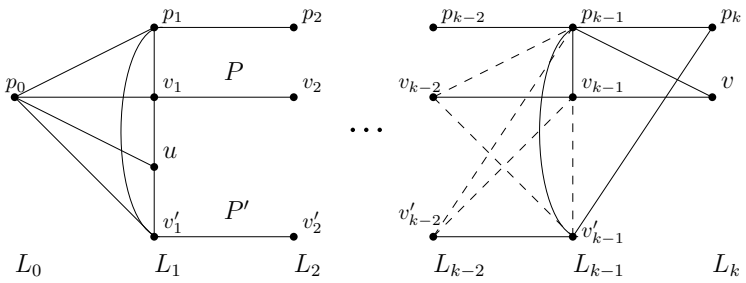


Figure 4 Illustration of the Case 2 in the proof of Theorem 16.

To argue the correctness of the colouring let us look at a pair of arbitrary vertices $u \in L_i$ and $v \in L_j$. Firstly we consider the case when $i = j$. By Claim 11 we know there will be a rainbow path for all cases of G except for when $\text{diam}(G) = 3$. For this instance there are three possibilities for i and j.

1. $i = j = 1$. The path up_0v will always exist and is rainbow.
2. $i = j = 2$. Because of Observation 18 we know u and v must be dominated by either p_1, p_2 or both. If they are dominated by the same vertex there is a rainbow path, while if they are dominated by different vertices either path up_1p_2v and up_2p_1v is rainbow.
3. $i = j = 3$. The vertices are either dominated by p_2, p_3 or both. If they are dominated by the same vertex there is a rainbow path, and if they are dominated by different vertices either path up_2p_3v and up_3p_2v is rainbow.

We will now examine all cases of u and v such that $0 \leq i < j \leq k$. First we can observe by Claim 10 that when $i \geq 3$ we can always use the dominating path. We will look at the cases where the dominating path cannot be arbitrarily used, i.e. when $i < 3$ more closely.

Case 1. $i = 0$. The only case we cannot use the dominating path is if $j = k$ and v is only dominated in layer. We know by Observation 9 that p_0 must have a direct path to every vertex including v. By Claim 22, this path will never intersect some vertex coloured 1 in L_2 and thus is rainbow.

Case 2. $i = 1$. If u is dominated to the right the diametral path will be rainbow for every case of v. This leaves us two cases of u.

1. u is dominated in layer. The only case of v we have to look at is when $j = k$ while v is only dominated in layer. There are two subcases we have to examine for this case.
   a. $\text{dist}(u, v) = k - 1$. Since the path between u and v is direct and v is only dominated in layer we know by Claim 22 that no internal vertex is coloured 1 and thus the path is rainbow.
   b. $\text{dist}(u, v) = k$. Using Claim 23 we know there will be some path repeating in L_1 and from there go directly to v. Since the path from L_1 to v is direct we can again use Claim 22 and thus we know that all internal vertices are distinctly coloured.
2. u is only dominated by p_0. For this case in both instances when $j = k - 1$ and $j = k$ we cannot arbitrarily use the diametral path.
   a. $j = k - 1$. If v is dominated to the left we can use the diametral path, thus let us assume v is only dominated in layer. From p_0 there must exist some direct path to v. We can thus go from u to p_0 and then traverse the direct path to v. By Claim 22, since v is dominated in layer, the vertex in L_2 of the direct path will never be coloured 1.

b. $j = k$. First we observe that the combination of u not being dominated in layer and Claim 20 means that $\text{dist}(u, v) = k$. By Claim 23 we therefore know there must exist some path $P = uv_1v_2...v_{k-1}v$, where $v_1 \neq p_1$. If v is only dominated in layer, Claim 22 states that v_2 must be coloured 2 and thus P is rainbow. Therefore let us assume v is dominated to the left. As argued before, the fact that u is not dominated in layer and Claim 20, imply that by Claim 23 we can identify that the following path $P' = uv'_1v'_2...v'_{k-1}p_k$ also must be in G. By Observation 21 v'_2 is not coloured 1 as this vertex has a direct path to p_k. If for some $i < k$, $v'_i = p_i$ then we have the following rainbow path: $uv'_1...v'_{i-1}p_i...p_{k-1}v$, thus let us assume for all $i < k$, $v'_i \neq p_i$. Using Claim 20 we know both v'_{k-1} and v_{k-1} must be dominated in layer. With P and P' we can as a result see the formation of a large cycle starting at u and meeting at p_{k-1}. In particular in L_{k-1} we have a P_3 on the vetices $v'_{k-1}p_{k-1}v_{k-1}$. We let P'' denote the shortest path between v_{k-1} and v'_{k-1} in $G' = G[\{u\} \cup \{v_1, v'_1\} \cup ... \cup \{v_{k-2}, v'_{k-2}\} \cup \{v_{k-1}, v'_{k-1}\}]$. If $|E(P'')| = 2$ then either v_{k-2} or v'_{k-2} is an internal node of P''. If v_{k-2} is the internal node this means the edge (v_{k-2}, v'_{k-1}) is in $E(G)$, which means there is a direct path from v_2 to p_k and therefore v_2 would be coloured 2, making P a rainbow path. We therefore check for when v'_{k-2} is the internal node. This means that (v'_{k-2}, v_{k-1}) is an edge in $E(G)$, but this results in the rainbow path: $uv'_2...v'_{k-2}v_{k-1}v$. Since $|E(P'')| = 2$ always results in a rainbow path between u and v we check for when $|E(P'')| = 3$. This means that (v_{k-2}, v'_{k-2}) is an edge in $E(G)$. We already established earlier that neither edge (v'_{k-2}, v_{k-1}) or (v_{k-2}, v'_{k-1}) can be in $E(G)$. If (v_{k-2}, p_{k-1}) is in the graph then, again, v_2 will have a direct path to p_k – resulting in v_2 being coloured 2. If (v'_{k-2}, p_{k-1}) is an edge in the graph then there is a rainbow path $uv'_1...v'_{k-2}p_{k-1}v$ between u and v, thus we can omit these two edges. As we currently have a cycle $v_{k-2}v_{k-1}p_{k-1}v'_{k-1}v'_{k-2}$ and no other edges can be added to G' in these layers we see that for $|E(P'')| > 3$ we will find an even longer induced cycle. Therefore $|E(P'')| = 1$ and by applying Claim 19 on the $P_4 = v_{k-2}v_{k-1}v'_{k-1}v'_{k-2}$ we conclude (v_{k-2}, v'_{k-2}) is an edge in G. However as argued above, (v_{k-2}, v'_{k-1}) and (v'_{k-2}, v_{k-1}) are not in G. We then have an induced cycle of length four in G, a contradiction since G is chordal. All the potential edges in the cycle is shown in Figure 4.

The proof of **Case 3**, that is, when $i = 2$, is deferred to the full version of this work.

This proves c is indeed a rainbow colouring for G with $\text{diam}(G) - 1$ colours. Finding the root vertex of the dominating diametral path is a polynomial time operation while the BFS takes linear time. Checking if nodes in L_2 must be coloured 1 is done by checking for each vertex in L_2 their distance to p_k. This is a $\mathcal{O}(n^2)$ time operation, thus when summarizing all these times together we end up with a polynomial time algorithm. ◀

6 Claw-free graphs with a dominating diametral path

In this section, we give a polynomial-time algorithm to optimally rainbow vertex colour claw-free graphs with a dominating diametral path. In particular, this proves Conjecture 1 for claw-free diametral path graphs. This result also improves on the previous algorithm for unit interval graphs [6].

▶ **Theorem 24.** *If G is a claw-free graph with a dominating diametral path, then* $\mathbf{rvc}(G) = \text{diam}(G) - 1$ *and the corresponding rainbow vertex colouring can be found in time that is polyonmial in the size of G.*

Proof. Let G be a claw-free diametral path graph. We run a BFS on one of the ends of the diametral path. We denote this vertex p_0 and we let k denote the number of layers of the BFS-tree excluding p_0. The vertex of the diametral path in each layer we denote p_i, $1 \leq i \leq k$. To construct a colouring $c : V \to [k-1]$ on G for $v \in L_i$ we assign the colouring:

$$c(v) = \begin{cases} k-1 & \text{if } i = 0 \\ 1 & \text{if } i = k \text{ or if } N(v) \cap L_{i+1} = \emptyset \\ i & \text{otherwise.} \end{cases}$$

Before arguing the correctness of our colouring we make the following observations.

▶ **Observation 25 (♣).** *A direct path between vertices $x \in L_i$ and $y \in L_j$, where $j > i$, cannot intersect some vertex $z \in L_l$ coloured 1, $0 < i \leq l < j$.*

▷ Claim 26 (♣). If a vertex $x \in L_i$, with $i > 1$, is dominated to the left, then x is also dominated in layer.

To argue the correctness of the colouring we will look at a pair of arbitrary vertices $u \in L_i$ and $v \in L_j$. We start by looking at the case where $i = j$. By Claim 11 we know that for all cases except for when $\text{diam}(G) = 3$ there will be a rainbow path between u and v. There are three possible cases for i and j:

1. $i = j = 1$. Both u and v are connected to p_0, leading to the rainbow path up_0v.
2. $i = j = 2$. Claim 26 states that neither u nor v can only be dominated to the left, which means they are either dominated by p_2, p_3 or both. If the vertices are dominated by the same vertex there is a rainbow path, and for the case they are dominated by different vertices either path up_2p_3v and up_3p_2v is rainbow.
3. $i = j = 3$. Since u and v are in layer L_3 they cannot be dominated to the right. Claim 26 tells us u and v cannot only be dominated left so we therefore know that both u and v must be dominated in layer. We, as a result, have the rainbow path up_3v.

For the rest of the proof we will assume $0 \leq i < j \leq k$. For cases when $i \geq 3$ we know by Claim 10 that there exists a rainbow path between u and v. When $i = 0$ we know by Observation 9 that there must always be a direct path to v, and this path will never intersect some vertex coloured 1, other than in L_1, as this would by Observation 25 imply the path not being direct. When $i = 2$ we know from Claim 26 that u must be dominated in layer and thus for all cases of j the diametral path will have a rainbow path to v. We are thus left with only one case to examine, which is when $i = 1$ while u is not dominated to the right.

1. $j = k - 1$. Consider the path using the edge (u, p_0) and then a direct path between p_0 and v. The direct path between p_0 and v will by Observation 25 never intersect a vertex coloured 1 in layers other than L_1 and thus the path is rainbow.
2. $j = k$. If $\text{dist}(u, v) = k - 1$ then the path goes directly to v and by Observation 25 this path will not intersect some vertex coloured 1, thus it is rainbow. If $\text{dist}(u, v) = k$ the path must repeat layers at some point. If the repetition of layers happens in L_i or L_j the path will be rainbow. Therefore let us assume this repetition happens in some layer L_l where $i < l < j$. We have the following path: $P = u...v_{l-1}v_lv'_lv'_{l+1}...v$. If v_l has no neighbour in L_{l+1} it will be coloured 1, making P rainbow, thus let us assume P has some neighbour $v_{l+1} \in L_{l+1}$. If $(v_{l-1}, v'_l) \notin E(G)$ and $(v'_l, v_{l+1}) \notin E(G)$ then $\{v_lv_{l-1}v'_lv_{l+1}\}$ induces a claw in G. As a result, either (v_{l-1}, v'_l) or (v'_l, v_{l+1}) must be an edge in $E(G)$. The former case implies a direct path $u...v_{l-1}v'_lv'_{l+1}...v$ between u and v, thus let us assume only $(v'_l, v_{l+1}) \in E(G)$. Since v'_l must have some neighbour v'_{l-1} in L_{i-1}

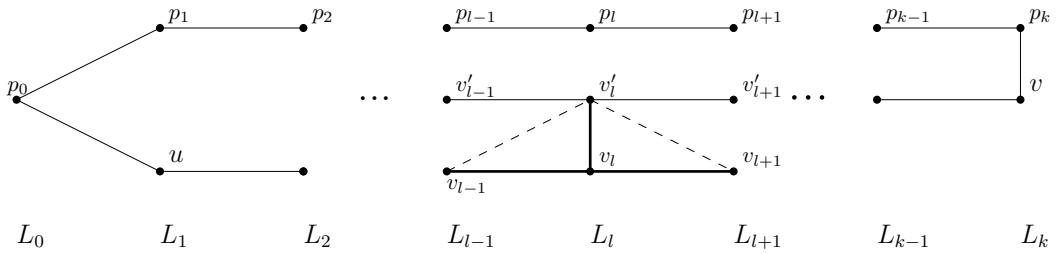


Figure 5 Scenario where the path repeats in layer L_l. If v_l has no neighbours in L_{l+1} it is coloured 1, otherwise one of two possible edges (dashed lines) must be in the graph.

another claw becomes apparent: $\{v'_l v'_{l-1} v'_{l+1} v_{l+1}\}$. In a similar argument to the proof of Claim 26 we have three possible edges to prevent an induced claw, but only one, as seen by Observation 6 is allowed within the BFS-structure – thus (v_{l+1}, v'_{l+1}) must be an edge within $E(G)$. The inclusion of this edge means the repetition of layers can be shifted one step further to the right, to L_{i+1}, with the following path: $u...v_{l-1}v_l v_{l+1}v'_{l+1}...v$. For this new path either v_{l+1} has a neighbour in L_{l+2}, which means the repetition can be shifted a further step to the right, the argument being identical to the one presented just now, or v_{l+1} has no neighbour in L_{i+2} meaning its coloured 1, making this new path rainbow. If the path repeats in L_k, in the scenario where the repetition has shifted all the way to the right, the path will also be rainbow. This scenario where the path between u and v repeats in some layer L_l is shown in Figure 5.

This proves that c is a rainbow colouring for G using $\text{diam}(G) - 1$ colours. It is simple to see that our colouring algorithm only takes polynomial time. The BFS-search is linear, while finding vertices which are not dominated by the following layer is also linear. This added with the time it takes to find the dominating diametral path results in a polynomial time algorithm. ◀

7 Interval graphs

In Section 5, we presented a proof which generalized an earlier result concerning interval graphs. For this section we will expand on the theme of interval graphs as we prove that also SRVC can be efficiently solved on this graph class. Only an algorithm for computing an optimal strong rainbow verex colouring on unit interval graphs was previously known [6].

▶ **Theorem 5.** *If G is an interval graph, then* $\mathbf{srvc}(G) = \text{diam}(G) - 1$ *and an optimal strong rainbow vertex colouring can be found in time that is linear in the size of G.*

8 Conclusion

In this work we have given efficient algorithms for optimally colouring multiple subclasses of graphs with dominating diametral paths. We have shown that for a graph G with a dominating diametral path, and one of the following properties: chordal, claw-free or bipartite, we can find a colouring in polynomial time using $\text{diam}(G) - 1$ colours. But the status of the of the conjecture of Heggernes et al. [6] still remains an open problem.

We believe the conjecture is true because diametral path graphs seem to be a graph class that is more dense than graphs with a dominating diametral path. As we have seen for the claw-free and chordal graphs, the dominating diametral path in conjunction with some

property making for a more dense graph class seems to help in terms of finding a rainbow vertex colouring using $\text{diam}(G) - 1$ colours. A step towards solving Conjecture 1 would be to consider other widely studied subclasses of diametral path graphs, such as co-comparability graphs and AT-free graphs.

On the other hand, we showed Conjecture 1 is not true for a closely related graph class, namely if we assume that only the input graph has a dominating diametral path, as opposed to every induced subgraph of it. We showed an example of a graph with a dominating diametral path that requires $\text{diam}(G)$ many colours in any rainbow vertex colouring. We also showed that RVC becomes NP-complete on this graph class. In the light of our algorithms for chordal, bipartite and claw-free graphs with a dominating diametral path, it would be interesting to understand which other properties ensure efficient algorithms for RVC on graphs with a dominating diametral path.

References

1. J. A. Bondy and U.S.R. Murty. *Graph Theory*. Springer, 2008.
2. Gary Chartrand, Garry L Johns, Kathleen A McKeon, and Ping Zhang. Rainbow connection in graphs. *Mathematica bohemica*, 133(1):85–98, 2008.
3. Jitender S Deogun and Dieter Kratsch. Diametral path graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 344–357. Springer, 1995.
4. Paul Dorbec, Ingo Schiermeyer, Elżbieta Sidorowicz, and Éric Sopena. Rainbow connection in oriented graphs. *Discrete Applied Mathematics*, 179:69–78, 2014.
5. Eduard Eiben, Robert Ganian, and Juho Lauri. On the complexity of rainbow coloring problems. *Discrete Applied Mathematics*, 246:38–48, 2018.
6. Pinar Heggernes, Davis Issac, Juho Lauri, Paloma T Lima, and Erik Jan van Leeuwen. Rainbow vertex coloring bipartite graphs and chordal graphs. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
7. Michael Krivelevich and Raphael Yuster. The rainbow connection of a graph is (at most) reciprocal to its minimum degree. *Journal of Graph Theory*, 63(3):185–191, 2010.
8. Juho Lauri. *Chasing the Rainbow Connection: Hardness, Algorithms, and Bounds*. PhD thesis, Tampere University of Technology, 2016.
9. Shasha Li, Xueliang Li, and Yongtang Shi. Note on the complexity of deciding the rainbow (vertex-) connectedness for bipartite graphs. *Applied Mathematics and Computation*, 258:155–161, 2015.
10. Xueliang Li, Yaping Mao, and Yongtang Shi. The strong rainbow vertex-connection of graphs. *arXiv preprint*, 2012. `arXiv:1201.1541`.
11. Paloma T Lima, Erik Jan van Leeuwen, and Marieke van der Wegen. Algorithms for the rainbow vertex coloring problem on graph classes. *Theoretical Computer Science*, 887:122–142, 2021.
12. Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

On the Complexity of Tree Edit Distance with Variables

Tatsuya Akutsu[1]
Bioinformatics Center, Institute for Chemical Research, Kyoto University, Japan

Tomoya Mori
Bioinformatics Center, Institute for Chemical Research, Kyoto University, Japan

Naotoshi Nakamura
The Thomas N. Sato BioMEC-X Laboratories, Advanced Telecommunications Research Institute International (ATR), Kyoto, Japan
Karydo TherapeutiX, Inc., Tokyo, Japan
Interdisciplinary Biology Laboratory (iBLab), Division of Natural Science, Graduate School of Science, Nagoya University, Japan

Satoshi Kozawa
The Thomas N. Sato BioMEC-X Laboratories, Advanced Telecommunications Research Institute International (ATR), Kyoto, Japan
Karydo TherapeutiX, Inc., Tokyo, Japan

Yuhei Ueno
The Thomas N. Sato BioMEC-X Laboratories, Advanced Telecommunications Research Institute International (ATR), Kyoto, Japan
Karydo TherapeutiX, Inc., Tokyo, Japan
V-iCliniX Laboratory, Nara Medical University, Japan

Thomas N. Sato[2]
The Thomas N. Sato BioMEC-X Laboratories, Advanced Telecommunications Research Institute International (ATR), Kyoto, Japan
Karydo TherapeutiX, Inc., Tokyo, Japan
V-iCliniX Laboratory, Nara Medical University, Japan

Abstract

In this paper, we propose *tree edit distance with variables*, which is an extension of the tree edit distance to handle trees with variables and has a potential application to measuring the similarity between mathematical formulas. We analyze the computational complexity of several variants of this model. In particular, we show that the problem is NP-complete for ordered trees. We also show for unordered trees that the problem of deciding whether or not the distance is 0 is graph isomorphism complete but can be solved in polynomial time if the maximum outdegree of input trees is bounded by a constant. We also present parameterized and exponential-time algorithms for ordered and unordered cases, respectively.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Tree edit distance, unification, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.44

Related Version *Full Version*: https://arxiv.org/abs/2105.04802

Funding *Tatsuya Akutsu*: Partially supported by JSPS KAKENHI #JP22H00532 and #JP22K19830.
Naotoshi Nakamura: Partially supported by JSPS KAKENHI #JP17H06003 and #JP19H05422.

[1] Corresponding author
[2] Corresponding author

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park; Article No. 44; pp. 44:1–44:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Thomas N. Sato: Partially supported by JST ERATO Grant Number JPMJER1303, Nakatani Foundation, and AMED under Grant Number JP21he2102002.

Acknowledgements We are grateful to the members of Sato lab at ATR and Karydo TherapeutiX, Inc. for advice and discussion throughout the course of this work.

1 Introduction

Measuring the similarity of tree structured data is a fundamental problem in computer science because various kinds of data are represented as trees. *Tree edit distance* is one of the most extensively studied measures for dissimilarity between two rooted trees. It is known that tree edit distance can be computed in polynomial time for ordered trees [7, 14, 16, 18], whereas its computation is NP-hard for unordered trees [19].

Mathematical formulas are one of the widely used tree structured data. Indeed, many methods have been developed for retrieving similar mathematical formulas [1, 10, 15, 20]. Comparison of mathematical formula is also important for analysis of biological systems [17]. However, most of the developed search methods are heuristic ones and are not studied from a viewpoint of the computational complexity. *Unification* is a basic technique to evaluate the identify of logic formulas, and the computational complexity of various variants (e.g., allowing associative and/or commutative laws) has been studied [3, 11, 12]. However, unification does not give a similarity or distance measure. Although a combination of unification and tree edit distance was proposed under the name of "tree edit distance with variables", only a quite restricted case (each variable can occur only once) was studied [3].

In order to compare mathematical formulas, it is important to consider trees with variables. For example, consider two functions $f(x, y, z)$ and $g(x, y, z)$ defined by:

$$\begin{aligned} f(x, y, z) &= (x + y) \times z, \\ g(x, y, z) &= (x + z) \times y. \end{aligned}$$

These two functions are essentially the same: the former one is identical to the latter one by replacing y and z with z and y, respectively. In addition, consider a function $h(x, y, z)$ defined by:

$$h(x, y, z) = z \times (x + y).$$

This function is also essentially the same as f and g because multiplication satisfies the commutative law. Functions f, g, and h can be respectively represented as T_1, T_2, and T_3 shown in Figure 1. If we ignore variable names assigned to leaves, these trees are identical as unordered rooted trees. However, considering variable names is important. For example, consider a function k defined by

$$k(x, y) = (x + y) \times x.$$

This function can be represented as a rooted tree T_4 in Figure 1. Although (unordered) tree structures of $T_1, \ldots, T_4$ are identical, k is clearly different from f, g, and h. Therefore, variable names assigned to leaves should be taken into account.

Based on the above discussion, we introduce *tree edit distance with variables* in this paper. Before giving this new distance measure, we briefly review the standard *tree edit distance* [6]. Let T_1 and T_2 be two rooted trees in which each node has a label from some alphabet. We consider two cases: both T_1 and T_2 are ordered trees, and both T_1 and T_2 are unordered trees. This distinction can be taken into account only when we consider whether or not two trees

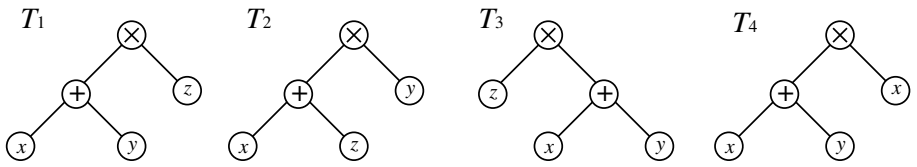


Figure 1 Tree representations of mathematical expressions.

Table 1 Summary of Results.

	$d_0(T_1, T_2)$	iso	iso-BD	$d(T_1, T_2)$	$d(T_1, T_2)$-BD
ordered	P [16]	P (Prop. 3)	P (Prop. 3)	NPC (Thm. 4) $O((\sqrt{3})^M \cdot poly(n_1, n_2))$ time (Thm. 7)	NPC (Thm. 4)
unordered	NPC [19]	GIC (Thm. 9)	P (Thm. 9)	NPC [19] $O(\left(\frac{M}{e}\right)^{(\frac{1}{2}+\delta)M} \cdot 1.26^{n_1+n_2})$ time (Prop. 10)	NPC [19]

are identical (i.e., isomorphic) after tree editing operations. The tree edit distance $d_0(T_1, T_2)$ between T_1 and T_2 is defined as the cost of the minimum cost sequence of edit operations that transforms T_1 to T_2, where an operation is one of deletion of a node, insertion of a node, and change of the label of a node. Then, we define the tree edit distance between two trees in which leaves can have variables as labels T_1 and T_2 by $dist(T_1, T_2) = \min_\theta dist_0(T_1\theta, T_2\theta)$, where θ is a substitution (i.e., a set of assignments of constants to variables). See Section 2 for the precise definitions.

In this paper, we analyze the computational complexity of several variants/subcases of the tree edit distance problem with variables, with focusing on the unit cost model (i.e., the cost of each edit operation is 1). When discussing the complexity classes, we consider a decision version of the problem: whether or not $dist(T_1, T_2) \leq d$ for given T_1, T_2, and a given non-negative real number d. The main results are summarized in Table 1, where "iso" asks whether $d(T_1, T_2) = 0$, "BD" means that the maximum *outdegree* (i.e., the maximum number of children) of both T_1 and T_2 is bounded by a constant, M denotes the number of occurrences of variables in T_1 and T_2, n_i denotes the number of nodes in T_i, and δ is any positive constant. In this table, P, NPC, and GIC mean that the target problem is polynomial-time solvable, NP-complete, and Graph Isomorphism complete (i.e., as hard as the graph isomorphism problem under polynomial-time reduction), respectively. It is interesting to see that the complexity substantially changes according to introduction of variables.

2 Preliminaries

In this section, we review the precise definition of the tree edit distance and then formally define the tree edit distance with variables.

Let T_1 and T_2 be two rooted trees in which each node has a label from an alphabet Σ. We use $\ell(v)$ to denote the label of a node v. As mentioned in Section 1, we consider two cases: both T_1 and T_2 are ordered trees, and both T_1 and T_2 are unordered trees, and this distinction can be taken into account only when we consider whether or not two trees are identical after tree edit operations. We consider three kinds of *edit operations* (see also Figure 2):

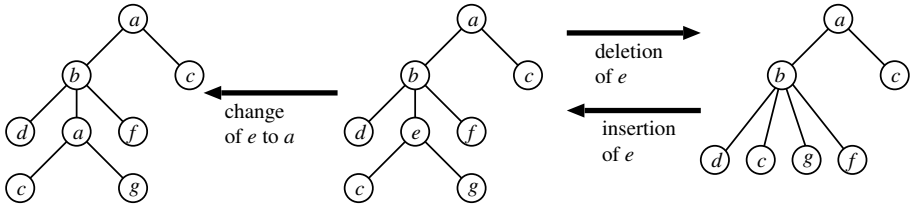


Figure 2 Tree edit operations.

Deletion: Delete a non-root node v in a tree T with parent u, making the children of v become children of u. The children are inserted in the place of v into the set of the children of u.

Insertion: Inverse of delete. Insert a node v as a child of u in T, making v the parent of some of the children of u.

Change-Label: Change the label of a node v in T.

We assign a *cost* for each editing operation: $\gamma(a, b)$ denotes the cost of changing a node with label a to label b, $\gamma(a, \epsilon)$ denotes the cost of deleting a node labeled with a, $\gamma(\epsilon, a)$ denotes the cost of inserting a node labeled with a. We assume that $\gamma(x, y)$ satisfies the conditions of distance metric: $\gamma(x, x) = 0$, $\gamma(x, y) = \gamma(y, x)$, $\gamma(x, y) \geq 0$, and $\gamma(x, z) \leq \gamma(x, y) + \gamma(y, z)$. Then, the *edit distance* between T_1 and T_2 is defined as the cost of the minimum cost sequence of edit operations that transforms T_1 to T_2 (precisely, transforms T_1 to a tree identical to T_2). It is well-known that this distance satisfies the conditions of distance measure, in both ordered and unordered cases.

In this paper, we focus on the *unit cost model* in which the cost of each edit operation is 1 (i.e., $\gamma(x, y) = 1$ for any $x \neq y$). Note that all hardness results hold for a general cost model because the unit cost model is a special case. For positive results, we need to consider mapping costs between variables in T_1 and T_2. Since it is difficult to define appropriate general costs for such cases, we only consider the unit cost model in this paper.

Here we define the tree edit distance with variables. Let Σ be a set of constant symbols, where each constant is denoted by a lower-case letter (e.g., $a, b, c, x, y, z, a_1, a_2$). Let Λ be a set of variables, where each variable is denoted by an upper-case letter (e.g., X, Y, Z, X_1, X_2). A substitution is a set of variable-constant pairs, $\theta = \{(X_1, x_1), (X_2, x_2), \ldots, (X_k, x_k)\}$, where $X_i \neq X_j$ holds for all $i \neq j$ but $x_i = x_j$ can hold for some (i, j). For a rooted tree T and a substitution θ, $T\theta$ denotes the tree obtained by changing variables appeared in T to constants according to θ (each X_i is replaced with x_i). Let $dist_0(T_1, T_2)$ be the standard tree edit distance between T_1 and T_2 (i.e., distance between trees without variables). We reasonably assume the following:

- Variable symbols appear only in leaves.
- The sets of variables appearing T_1 and T_2 are disjoint.
- Distinct variables in the same tree must be substituted to distinct constants by θ.
- Every variable appearing in T_1 (resp., T_2) is substituted to a constant symbol not appearing in T_1 or T_2 (because otherwise the cost of substituting a variable to a constant would be 0, which is not appropriate for measuring the distance between two mathematical expressions).

Then, we define the tree edit distance with variables as follows.

▶ **Definition 1.** *The tree edit distance with variables between T_1 and T_2 is*

$$dist(T_1, T_2) = \min_{\theta} dist_0(T_1\theta, T_2\theta).$$

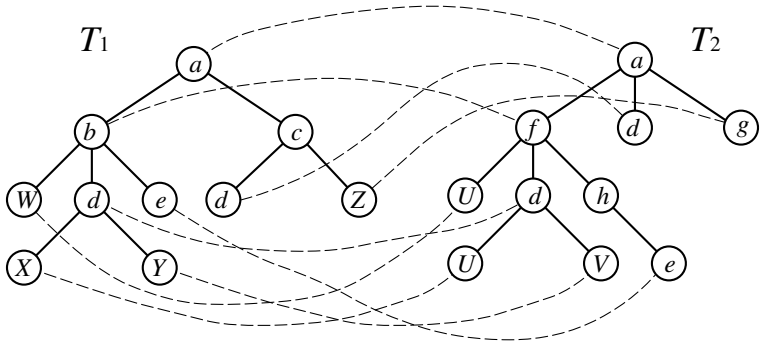


Figure 3 Example of a tree pair. In this case, $dist(T_1, T_2) = 5$ under the unit cost model. Dashed curves show a tree mapping corresponding to the minimum cost sequence of edit operations.

For example, consider trees T_1 and T_2 shown in Figure 3 and the unit cost model (i.e., $\gamma(x, y) = 1$ for any $x \neq y$). Then, $dist(T_1, T_2) = 5$ (in both ordered and unordered cases) by $\theta = \{(X, x), (Y, y), (Z, z), (W, w), (U, x), (V, y)\}$ and the following sequence of editing operations: change the label of node w to x, insert node h, change the label of node b to f, delete node c, and change the label of node z to g, where we identify nodes by their labels.

As the basic property, the following holds.

▶ **Proposition 2.** *For both ordered and unordered cases, tree edit distance with variables satisfies the conditions of a distance measure.*

Proof. Two trees that are isomorphic by one-to-one renaming of variables are regarded as identical. Clearly, $dist(T_1, T_2) = 0$ if and only if T_1 and T_2 are identical. Since $dist_0(T_1, T_2) = dist_0(T_2, T_1)$ holds for variable-free trees T_1 and T_2, $dist(T_1', T_2') = dist(T_2', T_1')$ holds for trees with variables T_1' and T_2'. Let $\theta_{1,2} = \mathrm{argmin}_\theta dist_0(T_1\theta, T_2\theta)$ and $\theta_{2,3} = \mathrm{argmin}_\theta dist_0(T_2\theta, T_3\theta)$. We assume without loss of generality (w.l.o.g.) that $\theta_{1,2}$ and $\theta_{2,3}$ give the same substitutions for variables appearing in T_2. Let $\theta_{1,3}$ be the union of $\theta_{1,2}$ and $\theta_{2,3}$. Since $T_1\theta_{1,2} = T_1\theta_{1,3}$, $T_2\theta_{1,2} = T_2\theta_{2,3} = T_2\theta_{1,3}$, and $T_3\theta_{2,3} = T_3\theta_{1,3}$ hold, we have

$$\begin{aligned} dist(T_1, T_3) &\leq dist_0(T_1\theta_{1,3}, T_3\theta_{1,3}) \\ &\leq dist_0(T_1\theta_{1,2}, T_2\theta_{1,2}) + dist_0(T_2\theta_{2,3}, T_3\theta_{2,3}) \\ &= dist(T_1, T_2) + dist(T_2, T_3). \end{aligned}$$

◀

There is a close relationship between the tree edit distance and the tree mapping [6]. For an unordered tree, a bijective mapping $\mathcal{M} \subseteq V(T_1) \times V(T_2)$ is called a *tree mapping* if for every $(u_1, v_1), (u_2, v_2) \in \mathcal{M}$, it holds that: (i) $u_1 = u_2$ if and only if $v_1 = v_2$; and (ii) u_1 is an ancestor of u_2 if and only if v_1 is an ancestor of v_2. Condition (i) states that each node appears at most once in $\mathcal{M}$, and condition (ii) states that ancestor-descendant relations must be preserved in $\mathcal{M}$. For ordered trees, the following condition is needed in addition to (i) and (ii): (iii) u_1 is left to u_2 if and only if v_1 is left to v_2. See [6] for the precise definition of "left". It is known that any edit sequence can be modified without changing the total cost such that change-label operations follow deletion operations and insertion operations follow change-label operations. Then, an edit sequence gives a tree mapping: the nodes not deleted or inserted correspond to each other. Conversely, a tree mapping gives an edit sequence: nodes in T_1 (resp., T_2) that do not appear in $\mathcal{M}$ are regarded as deleted (resp., inserted), and any $(u, v) \in \mathcal{M}$ is regarded as change-labeled if their labels are different. Therefore, we use such words as "u is mapped to v" when discussing about tree edit distance.

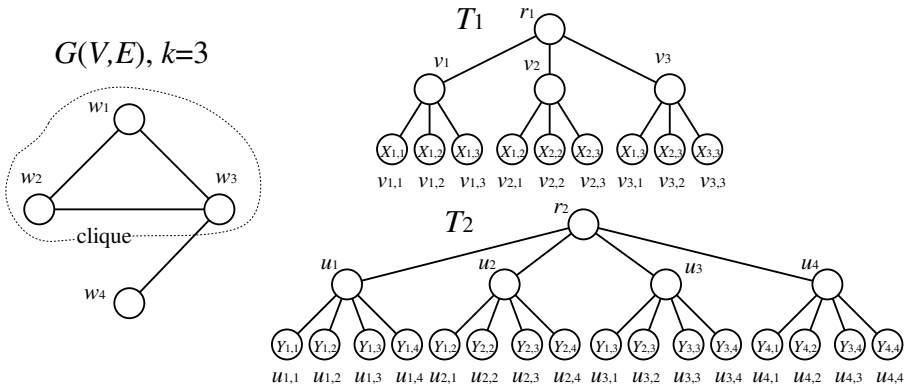


Figure 4 Reduction from maximum clique to ordered tree edit distance with variables, where only relevant labels are shown.

3 Ordered Trees

In this section, all trees are ordered trees, which means that the children of each node are ordered from left to right and that this ordering must be preserved among isomorphic trees. For each tree T, $V(T)$ and $E(T)$ denote the sets of nodes and edges, respectively. We let $n_1 = |V(T_1)|$ and $n_2 = |V(T_2)|$. For each node (resp., vertex) v in a tree (resp., in a graph), $\ell(v)$ denotes the label of v.[3] We may use the label of a node to denote the node itself when there is no confusion.

▶ **Proposition 3.** *For ordered trees, whether or not $dist(T_1, T_2) = 0$ can be determined in polynomial time.*

Proof. We construct an Euler string $str(T_i)$ [2] for each of the trees T_i using depth first search (DFS). In constructing $str(T_i)$, we assign a unique integer number from $1, 2, \cdots$ as the label of a variable node every when we first encounter the variable. Then, it is straightforward to see $str(T_1) = str(T_2)$ if and only if $dist(T_1, T_2) = 0$. ◀

▶ **Theorem 4.** *For ordered trees, the tree edit distance problem with variables is NP-complete.*

Proof. It is clear that the problem is in NP. Then, we show a polynomial-time reduction from the maximum clique problem (see also Figure 4). The maximum clique problem is, given an undirected graph $G(V, E)$ and an integer k, to decide whether or not there exists a complete subgraph (clique) of size (#vertices) k in $G(V, E)$, where all vertices have the same label. It is well-known that the problem is NP-complete.

From a given k, we construct T_1 as follows:

$$V(T_1) = \{r_1\} \cup \{v_1, \ldots, v_k\} \cup \left(\bigcup_{i \in \{1,\ldots,k\}} \{v_{i,1}, \ldots, v_{i,k}\} \right),$$

$$E(T_1) = \left(\bigcup_{i \in \{1,\ldots,k\}} \{(r_1, v_i)\} \right) \cup \left(\bigcup_{i \in \{1,\ldots,k\}} \{(v_i, v_{i,1}), \ldots, (v_i, v_{i,k})\} \right),$$

[3] We mainly use "nodes" for trees and "vertices" for graphs.

$$
\begin{aligned}
\ell(r_1) &= a,\\
\ell(v_1) &= \ell(v_2) = \cdots = \ell(v_k) = b,\\
\ell(v_{i,i}) &= X_{i,i} \text{ for all } i,\\
\ell(v_{i,j}) &= \ell(v_{j,i}) = X_{i,j} \text{ for all } i < j,
\end{aligned}
$$

where $X_{i,j} \neq X_{i',j'}$ for any $i \neq i'$ or $j \neq j'$.

From a given $G(V,E)$ with $V = \{w_1, \ldots, w_n\}$, we construct T_2 as follows:

$$
\begin{aligned}
V(T_2) &= \{r_2\} \cup \{u_1, \ldots, u_n\} \cup \left(\bigcup_{i \in \{1,\ldots,n\}} \{u_{i,1}, \ldots, u_{i,n}\} \right),\\
E(T_2) &= \left(\bigcup_{i \in \{1,\ldots,n\}} \{(r_2, u_i)\} \right) \cup \left(\bigcup_{i \in \{1,\ldots,n\}} \{(u_i, u_{i,1}), \ldots, (u_i, u_{i,n})\} \right),\\
\ell(r_2) &= a,\\
\ell(u_1) &= \cdots = \ell(u_n) = b,\\
\ell(u_{i,j}) &= \ell(u_{j,i}) = Y_{i,j} \text{ for all } \{w_i, w_j\} \in E \text{ with } i < j,\\
\ell(u_{i,j}) &= Y_{i,j} \text{ for all other } u_{i,j}\text{s},
\end{aligned}
$$

where $Y_{i,j} \neq Y_{i',j'}$ holds for any $i \neq i'$ or $j \neq j'$.

Here, we note that $n_1 = 1 + k + k^2$ and $n_2 = 1 + n + n^2$. We show that $G(V,E)$ has a clique of size k if and only if $dist(T_1, T_2) = n_2 - n_1$ (i.e., T_1 is obtained by deletion operations from T_2 and renaming of variables). We say that a tree mapping $\mathcal{M}$ is an *inclusion mapping* if $\mathcal{M}$ corresponds to the sequence of edit operations with cost $n_2 - n_1$ (i.e., $\mathcal{M}$ contains all nodes in T_1).

Suppose that $G(V,E)$ has a clique of size k. We assume w.l.o.g. that $\{w_1, w_2, \ldots, w_k\}$ be the set of nodes in that clique. Then, the following mapping gives an inclusion mapping from T_1 to T_2:

$$
\mathcal{M} = \{(r_1, r_2)\} \cup \{(v_i, u_i) \mid i = 1, \ldots, k\} \cup \{(X_{i,j}, Y_{i,j}) \mid 1 \leq i \leq j \leq k\}.
$$

Conversely, suppose that there exists an inclusion mapping $\mathcal{M}$ from T_1 to T_2. We assume w.l.o.g. that $\mathcal{M}$ includes the following mappings:

$$
\{(r_1, r_2)\} \cup \{(v_i, u_i) \mid i = 1, \ldots, k\}
$$

Then, for any (i,j) such that $1 \leq i < j \leq k$, $X_{i,j}$ must be mapped to $Y_{i,j}$ because v_i is mapped to u_i, v_j is mapped to u_j, and $X_{i,j}$ (resp., $Y_{i,j}$) is only one variable appearing in children of both v_i and v_j (resp., u_i and u_j). It means that for all (i,j) such that $1 \leq i < j \leq k$, there exists an edge between w_i and w_j. Therefore, there exists a clique of size k in $G(V,E)$. Note that although $X_{i,i}$ may not be necessarily mapped to $Y_{j,j}$, it does not cause a problem.

Finally, we consider the bounded degree case. In this case, it is enough to encode each non-leaf node as in Figure 5. Let $\hat{T}$ be the tree obtained from T by this encoding. Then, it is straightforward to see that there exists an inclusion mapping from T_1 to T_2 if and only if there exists an inclusion mapping from $\hat{T}_1$ to $\hat{T}_2$. ◀

▶ **Proposition 5.** *The tree edit distance problem with variables can be solved in polynomial time for ordered trees if each variable occurs once in input trees.*

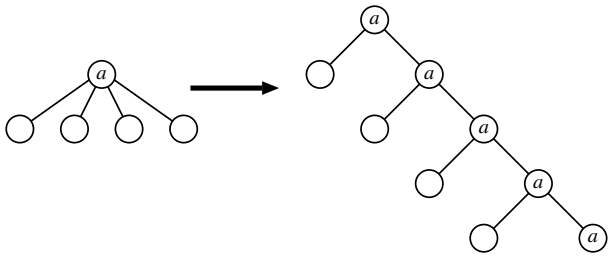


Figure 5 Encoding of the root node, where other non-leaf nodes are encoded in the same way except that label a is replaced with label b.

Proof. Let F_i denote an ordered forest (i.e., an ordered set of rooted trees). For each F_i, $V(F_i)$ denotes the set of nodes in F_i. For the root v of the rightmost tree in F_i, $F_i - T_i(v)$ denotes the forest obtained by removing the rightmost tree of F_i, and $F_i - v$ denotes the forest obtained by removing v (i.e., each child u of v becomes the root of the subtree induced by u and its descendants).

Recall that the edit distance for ordered trees (without variables) $dist_0(T_1, T_2)$ can be computed in $O(n_1^2 n_2^2)$ time by using the following dynamic programming algorithm [6, 18]:

$$
\begin{aligned}
D_0(F_1, \epsilon) &= \sum_{u \in V(F_1)} \gamma(\ell(u), \epsilon), \\
D_0(\epsilon, F_2) &= \sum_{v \in V(F_2)} \gamma(\epsilon, \ell(v)), \\
D_0(F_1, F_2) &= \min \begin{cases} D_0(F_1 - u, F_2) + \gamma(\ell(u), \epsilon), \\ D_0(F_1, F_2 - v) + \gamma(\epsilon, \ell(v)), \\ D_0(F_1 - T_1(u), F_2 - T_2(v)) \\ \qquad + D_0(T_1(u) - u, T_2(v) - v) \\ \qquad + \gamma(\ell(u), \ell(v)), \end{cases}
\end{aligned}
$$

where ϵ in $D_0(F_1, \epsilon)$ and $D_0(\epsilon, F_2)$ denotes the empty forest, u and v in the third recursion are the roots of the rightmost trees in F_1 and F_2, respectively, and $D_0(T_1, T_2)$ gives $dist_0(T_1, T_2)$. Then, it is enough to redefine $\gamma(x, y)$ function as follows:

$$
\begin{aligned}
\gamma(X_i, X_j) &= 0, \quad \text{for any variable pair } (X_i, X_j) \text{ such that } X_i \neq X_j, \\
\gamma(a, a) &= 0, \quad \text{for any constant symbol } a, \\
\gamma(x, y) &= 1, \quad \text{for any other pair } (x, y).
\end{aligned}
$$

Note that $\gamma(X_i, X_j) = 0$ always holds in this dynamic programming algorithm because X_i and X_j always appear in F_1 and F_2, respectively. Then, it is straightforward to see that this algorithm correctly computes the edit distance with variables and works in polynomial time. ◀

Let $DP_{SO}(T_1, T_2)$ denote the algorithm for two input trees T_1 and T_2 given in the proof. The above result and proof are very similar to those in Theorem 11 of [3]. However, each variable can match a subtree in [3], whereas each variable can match a variable or constant here. Hereafter, M denotes the total number of occurrences of variables, and $O^*(f(\cdots))$ denotes $O(f(\cdots) \cdot poly(n_1, n_2))$ time, where $poly(n_1, n_2)$ denotes some polynomial function of n_1 and n_2.

▶ **Proposition 6.** *The tree edit distance problem with variables for ordered trees can be solved in $O^*(2^M)$ time.*

Proof. Let $\mathcal{X} = (X^1, X^2, \ldots, X^{m_1})$ and $\mathcal{Y} = (Y^1, Y^2, \ldots, Y^{m_2})$ be the lists of occurrences of variables in the DFS ordering on T_1 and T_2, respectively, where $M = m_1 + m_2$. We examine all 0-1 assignments σ on $\mathcal{X}$ and $\mathcal{Y}$, where 1 (resp., 0) means that the corresponding variable is mapped (resp., is not mapped) to a variable in the other tree. Let $\phi(X^i) = |\{j \mid j \leq i,\ \sigma(X^j) = 1\}|$ and $\phi(Y^i) = |\{j \mid j \leq i,\ \sigma(Y^j) = 1\}|$. Since any edit operation does not change the ordering, X^i is mapped to Y^j such that $\phi(X^i) = \phi(Y^j)$. In some cases, X_i is mapped to multiple variables (e.g., Y_j and Y_k). We ignore such an assignment σ because of the constraint on θ. Then, each σ gives a matching (i.e., partial one-to-one mapping) between $\overline{\mathcal{X}}$ and $\overline{\mathcal{Y}}$, where $\overline{\mathcal{X}}$ (resp., $\overline{\mathcal{Y}}$) denotes the set of variables appearing in $\mathcal{X}$ (resp., $\mathcal{Y}$). Then, we assign a unique constant symbol to each variable in $\overline{\mathcal{X}}$. We assign the same symbol (e.g., b_k) as X_i to Y_j if X_i is mapped to Y_j. If a variable X_i (resp., Y_j) is not mapped to a variable, we assign a unique constant symbol to the variable (e.g., c_k for X_i, and d_h for Y_j). Let θ_σ denote the resulting substitution.

For example, let $\mathcal{X} = (X_1, X_2, X_3, X_2, X_4, X_5)$, $\mathcal{Y} = (Y_1, Y_2, Y_3, Y_4, Y_4, Y_5)$. For $\sigma(\mathcal{X}) = (1, 0, 1, 1, 1, 0)$ and $\sigma(\mathcal{Y}) = (1, 1, 1, 0, 1, 0)$, we have a mapping of $\{(X_1, Y_1), (X_2, Y_3), (X_3, Y_2), (X_4, Y_4)\}$, Then, we have a substitution θ_σ such that

$$\begin{aligned} \mathcal{X}\theta_\sigma &= (b_1, b_2, b_3, b_2, b_4, c_1), \\ \mathcal{Y}\theta_\sigma &= (b_1, b_3, b_2, b_4, b_4, d_1). \end{aligned}$$

If $\sigma(\mathcal{X}) = (1, 1, 0, 1, 1, 0)$ and $\sigma(\mathcal{Y}) = (1, 1, 1, 0, 1, 0)$, we ignore this assignment because X_2 should be mapped to both Y_2 and Y_3.

By applying substitution θ_σ to T_1 and T_2, we obtain variable-free trees $T_1\theta_\sigma$ and $T_2\theta_\sigma$. For each assignment σ, we compute $dist_0(T_1\theta_\sigma, T_2\theta_\sigma)$. Since all possible substitutions are examined by testing all σ, $\min_\sigma dist_0(T_1\theta_\sigma, T_2\theta_\sigma)$ gives $dist(T_1, T_2)$. Since 2^M assignments are examined and $dist_0(T_1\theta_\sigma, T_2\theta_\sigma)$ can be computed in polynomial time, the proposition holds. ◀

In the above, we consider all 0-1 assignments to all occurrences of variables. However, it is enough to find a mapping between X_is and Y_js and thus we need not consider all 0-1 assignments. Based on this idea, we have the following theorem.

▶ **Theorem 7.** *The tree edit distance problem with variables for ordered trees can be solved in $O^*((\sqrt{3})^M)$ time.*

Proof. As in the proof of Proposition 6, let $\mathcal{X}$ and $\mathcal{Y}$ be the lists of occurrences of variables in the DFS ordering for T_1 and T_2, respectively. For each variable X_i occurring h times in $\mathcal{X}$, we consider the following $2h - 1$ assignments: $(1, 0, 0, \cdots, 0, 0, 0)$, $(0, 1, 0, \cdots, 0, 0, 0)$, $(0, 0, 1, \cdots, 0, 0, 0)$, $\cdots$, $(0, 0, \cdots, 0, 0, 1)$, and $(1, 1, 1, \cdots, 1, 1, 1)$, $(0, 1, 1, \cdots, 1, 1, 1)$, $(0, 0, 1, \cdots, 1, 1, 1)$, $\cdots$, $(0, 0, 0, \cdots, 0, 1, 1)$. The first h cases mean that at most one occurrence of X_i is mapped to some variable Y_j. In this case, X_i is called a *single occurrence variable*, and the occurrences of X_i corresponding to "0" are replaced by a unique constant (e.g., c_k) not appearing in the other parts whereas the occurrence of X_i corresponding to "1" is kept as it is. The remaining $h - 1$ cases mean that the first "1" corresponds to the first occurrence of X_i that is mapped to some Y_j and that at least two occurrences of X_i are mapped to the same number of occurrences of Y_j. In this case, X_i is called a *multi occurrence variable*, and a unique constant (e.g., b_k) is shared by X_i and Y_j. For each multi occurrence variable, only the position of the first "1" is relevant. For example, suppose that X_i is the

multi occurrence variable to which "1" is assigned first in $\mathcal{X}$. Then, all occurrences of X_i are substituted by b_1. Suppose also that $X_{i'}$ is the next multi occurrence variable to which "1" is assigned. Then, all occurrences of $X_{i'}$ are substituted by b_2. Y_js are handled in an analogous way to X_is except that d_k is used in place of c_k.

From 0-1 assignments on variables given as above, we obtain substituted sequences of $\mathcal{X}$ and $\mathcal{Y}$, which are denoted by $\lambda(\mathcal{X})$ and $\lambda(\mathcal{Y})$, respectively. For example, let

$$\begin{aligned}\mathcal{X} &= (X_1, X_2, X_3, X_2, X_3, X_4, X_5, X_3, X_4, X_2, X_5),\\ \mathcal{Y} &= (Y_2, Y_1, Y_3, Y_4, Y_3, Y_4, Y_3, Y_2, Y_5, Y_4, Y_1).\end{aligned}$$

Suppose that (1), $(0,1,1)$, $(1,1,1)$, $(1,1)$, and $(0,1)$ are assigned to X_1, X_2, X_3, X_4, and X_5, respectively. Furthermore, suppose that $(1,0)$, $(1,1)$, $(0,1,1)$, $(1,1,1)$, and (1) are assigned to Y_1, Y_2, Y_3, Y_4, and Y_5, respectively. Then, we have

$$\begin{aligned}\lambda(\mathcal{X}) &= (X_1, b_2, b_1, b_2, b_1, b_3, c_1, b_1, b_3, b_2, X_5),\\ \lambda(\mathcal{Y}) &= (b_1, Y_1, b_3, b_2, b_3, b_2, b_3, b_1, Y_5, b_2, d_1).\end{aligned}$$

Note that X_1 (and any other variable) can match another variable in $\lambda(\mathcal{Y})$ with cost 0 and can match a constant symbol with cost 1. Note also that X_3's and Y_2's are substituted by b_1 because these are the multi occurrence variables to which "1" is assigned first in $\mathcal{X}$ and $\mathcal{Y}$, respectively. Note also that X_2's are substituted by b_2 because it is the multi occurrence variable that receives "1" after X_3 receives it.

Then, we consider the following procedure.

(i) $dmin \leftarrow 0$.

(ii) For all $\lambda(\mathcal{X})$ and $\lambda(\mathcal{Y})$, do step (iii).

(iii) $dmin \leftarrow \min(dmin, DP_{SO}(\lambda(\mathcal{X}), \lambda(\mathcal{Y})))$.

(iv) Output $dmin$.

The correctness of this procedure follows from the following observation. Let $\mathcal{M}$ be the tree mapping corresponding to the minimum cost edit sequence. If X_i and Y_j match to each other at two or more position pairs (i.e., both are multi occurrence variables) in $\mathcal{M}$, then there must exist a λ such that the same constant b_k is assigned to X_i and Y_j because b_ks are used only for multi occurrence variables. If X_i and Y_j match to each other at exactly one position pair in $\mathcal{M}$, both X_i and Y_j are treated as single occurrence variables and 1's in the 0-1 assignments correspond to the matching position pair. The other occurrences of variables correspond to deletions, insertions, or change-labels because b_i, c_i, and d_i are constant symbols not appearing in the original input trees.

Here, we analyze the number of combinations of 0-1 assignments, which gives the exponential factor of the algorithm. Let $\alpha_l M$ be the total number of occurrences of variables X_i and Y_j each of which occur l times. For example, $\alpha_1 = \frac{2}{22}$, $\alpha_2 = \frac{8}{22}$, $\alpha_3 = \frac{12}{22}$, and $\alpha_l = 0$ for $l \geq 4$, for the above mentioned $\mathcal{X}$ and $\mathcal{Y}$. Note that $\sum_{l=1}^{M} \alpha_l = 1$ holds. For each variable occurring h times ($h = 1, \ldots, M$), the number of examined 0-1 assignments is $2h-1$. Since $2h - 1 = 1$ for $h = 1$, the total number of combinations of 0-1 assignments for $\mathcal{X}$ and $\mathcal{Y}$ is

$$L_2(\alpha_2, \ldots, \alpha_M) = \prod_{h=2}^{M} (2h-1)^{\frac{\alpha_h M}{h}}.$$

▷ Claim 8. $f(h) = (2h-1)^{\frac{1}{h}}$ is decreasing with respect to $h = 2, 3, \cdots$.

Proof. It is seen by a simple numerical calculation that $(2 \cdot 2 - 1)^{\frac{1}{2}} > (2 \cdot 3 - 1)^{\frac{1}{3}}$. For $h \geq 3$, by taking the derivative of $\ln(f(h))$, we have

$$\frac{d\ln(f(h))}{dh} = \frac{d(\frac{1}{h}\ln(2h-1))}{dh} = -\frac{\ln(2h-1)}{h^2} + \frac{2}{(2h-1)h} < 0. \quad \triangleleft$$

Therefore, we have

$$\begin{aligned} L_2(\alpha_2, \ldots, \alpha_M) &= \prod_{h=2}^{M} (2h-1)^{\frac{\alpha_h M}{h}} \leq \prod_{h=2}^{M} (2 \cdot 2 - 1)^{\frac{\alpha_h M}{2}} \\ &= (3)^{\frac{\left(\sum_{h=2}^{M} \alpha_h\right) M}{2}} \leq 3^{\frac{M}{2}} < (\sqrt{3})^M. \end{aligned}$$

Since the other parts can be clearly done in polynomial time, the theorem holds. ◀

4 Unordered Trees

In this section, all trees are unordered rooted trees. The graph isomorphism problem is, given two undirected graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, to decide whether or not there exists a bijection ϕ from V_1 to V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(u), \phi(v)\} \in E_2$. It is unclear whether graph isomorphism is in P or NP-complete [5]. However, it is known that graph isomorphism can be solved in polynomial time if the maximum degree of input graphs is bounded by a constant [9, 13].

▶ **Theorem 9.** *For unordered trees, the problem of deciding $dist(T_1, T_2) = 0$ is graph isomorphism complete. Furthermore, the problem can be solved in polynomial time if the maximum outdegree of T_1 and T_2 is bounded by a constant.*

Proof. First, we show that graph isomorphism can be reduced to the problem in polynomial time. For each of G_1 and G_2, we construct trees as for T_2 in the proof of Theorem 4. Then, it is straightforward to see that G_1 and G_2 are isomorphic if and only if $dist(T_1, T_2) = 0$.

Next, we show that the problem can be reduced to graph isomorphism in polynomial time (see also Figure 6). Here, we consider w.l.o.g. graph isomorphism over labeled graphs (because it is obvious that labeled cases can be reduced to unlabeled cases in polynomial time). We show how to construct $G_1(V_1, E_1)$ from T_1, where an identical construction can be used for T_2. We construct $G_1(V_1, E_1)$ by adding vertices and edges to T_1 as follows. For each variable X_i, we create a new vertex v_{X_i} with constant label a, connect v_{X_i} to all leaves in T_1 having label X_i, and change the labels of these leaves to b, where a and b are constant symbols not appearing in T_1 or T_2 (we use the same a and b for all variables in T_1 and T_2). Then, it is straightforward to see that G_1 and G_2 are isomorphic if and only if $dist(T_1, T_2) = 0$.

Finally, we prove the last claim. We modify the reduction shown above (see also G'_1 in Figure 6). For each variable X_i, we make a copy $\tilde{T}_1$ of T_1 and then delete all of the following nodes in $\tilde{T}_1$:

- a node which is not a node with label X_i or its ancestor,
- a node which is an ancestor of the lowest common ancestor of all nodes with label X_i.

Then, we apply the deletion operation to the nodes in $\tilde{T}_1$ each of which has a single child and change labels of all internal nodes to a. Denote the resulting tree by T_{X_i}. Finally, we identify leaves of T_{X_i} with the corresponding leaves in T_1. Let G'_1 be the graph obtained by applying this procedure to all variables. We construct G'_2 in the same way. Clearly, this construction can be done in polynomial time. Furthermore, the maximum degree of the resulting graphs is

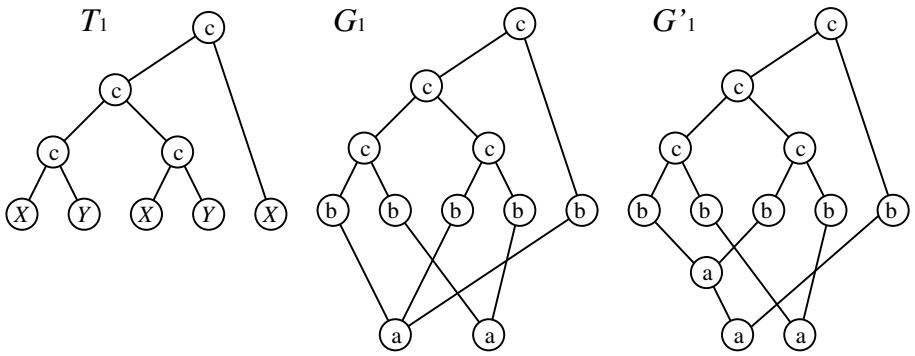


Figure 6 Transformation from tree T_1 to graph G_1 of unbounded degree and graph G'_1 of bounded degree.

bounded by the maximum degree of the input trees (if the maximum outdegree of the input trees is no less than 2). Since the structure of each T_{X_i} does not depend on the ordering of nodes, G'_1 and G'_2 are isomorphic if and only if $dist(T_1, T_2) = 0$. Since isomorphism of graphs of bounded degree can be tested in polynomial time [9, 13]. the last claim holds. ◀

As in Section 3, let M be the number of occurrences of variables in T_1 and T_2.

▶ **Proposition 10.** *$dist(T_1, T_2)$ can be computed in $O((\frac{M}{e})^{(\frac{1}{2}+\delta)M} \cdot 1.26^{n_1+n_2})$ time for unordered trees, where δ is any small positive constant.*

Proof. Recall $dist(T_1, T_2) = \min_\theta dist_0(T_1\theta, T_2\theta)$. Therefore, the problem can be solved by computing $dist_0(T_1\theta, T_2\theta)$ for all essentially different θ, where "essentially different" θ_1 and θ_2 mean that θ_1 and θ_2 give distinct correspondences between variables in T_1 and those in T_2. Let h_1 and h_2 be the numbers of variables in T_1 and T_2, respectively. Since we consider an upper bound, we assume w.l.o.g. that $h_1 = \alpha M$ and $h_2 = (1-\alpha)M$, where $0 < \alpha < \frac{1}{2}$. The number of one-to-one mappings from the variables in T_1 to the variables in T_2 is bounded by

$$\frac{h_2!}{(h_2 - h_1)!} = \frac{((1-\alpha)M)!}{((1-2\alpha)M)!}. \tag{1}$$

Note that some variable in T_1 may not be mapped to a variable in T_2 in some substitution θ. However, the distance would not be decreased and thus such a substitution can be ignored. By using upper and lower bounds of Stirling's approximation $\sqrt{2\pi n}(\frac{n}{e})^n \le n! \le e\sqrt{n}(\frac{n}{e})^n$, we have

$$\begin{aligned}\frac{((1-\alpha)M)!}{((1-2\alpha)M)!} &\le \frac{e\sqrt{(1-\alpha)M}\left(\frac{(1-\alpha)M}{e}\right)^{(1-\alpha)M}}{\sqrt{2\pi(1-2\alpha)M}\left(\frac{(1-2\alpha)M}{e}\right)^{(1-2\alpha)M}} \\ &= e\sqrt{\frac{(1-\alpha)}{2\pi(1-2\alpha)}}\left(\frac{(1-\alpha)^{(1-\alpha)}}{(1-2\alpha)^{(1-2\alpha)}}\right)^M \cdot \left(\frac{M}{e}\right)^{\alpha M}\end{aligned}$$

Since $\frac{(1-\alpha)^{(1-\alpha)}}{(1-2\alpha)^{(1-2\alpha)}} < 1.15$ holds for $0 < \alpha < \frac{1}{2}$ (using numerical calculations. Note that $\lim_{\alpha \to \frac{1}{2}}(1-2\alpha)^{(1-2\alpha)} = 1$), the above term is $O\left(1.15^M \cdot \left(\frac{M}{e}\right)^{\frac{M}{2}}\right)$ for a constant α. Note that if α is very close to $\frac{1}{2}$, we need to consider a factor of $\sqrt{\frac{(1-\alpha)}{2\pi(1-2\alpha)}}$ because α is not constant. In such a case, we use $((\frac{1}{2}+\epsilon)M)!$ to bound Eq.(1), where $\epsilon = \frac{1}{2} - \alpha$ and we can use

arbitrary small constant $\epsilon > 0$. This term is smaller than $O\left((\frac{M}{2e})^{(\frac{1}{2}+\delta)M}\right)$ for $\delta > \epsilon$. Since $O\left(1.15^M \cdot (\frac{M}{e})^{\frac{M}{2}}\right) \leq O\left((\frac{M}{2e})^{(\frac{1}{2}+\delta)M}\right)$ holds too, Eq.(1) is bounded by $O\left((\frac{M}{e})^{(\frac{1}{2}+\delta)M}\right)$ for any constant $\delta > 0$.

Since the tree edit distance between two unordered trees can be computed in $O(1.26^{n_1+n_2})$ time [4], the proposition holds. ◀

In the above theorem, M is defined as the number of occurrences of variables (in order to use the same parameter as in Theorem 7). However, M can be defined as the total number of variables in T_1 and T_2 in this theorem because we only consider the number of variables in the proof.

5 Concluding Remarks

In this paper, we have introduced and studied the tree edit distance problem with variables. We showed that the problem (decision problem version) is NP-complete even for ordered trees, whereas it is well-known that edit distance for ordered tree can be computed in polynomial time. We presented parameterized and exponential-time algorithms for the ordered and unordered cases, respectively. Since these algorithms are not necessarily optimal, improvements of these algorithms are left as open problems. As for the formalization, the unit cost model is assumed mainly because defining an appropriate cost model via substitutions on variables is difficult. Giving such costs and developing the corresponding algorithms would benefit the future practical applications

In this paper, we assumed that mathematical formulas are given as rooted trees. However, such formulas may be represented more efficiently by directed acyclic graphs (DAGs) with reusing identical sub-trees. Since it is not straight-forward to extend the algorithms for the tree edit distance to those for the graph edit distance for DAGs [8], it would be interesting to study such extensions with variables.

References

1 Akiko Aizawa and Michael Kohlhase. Mathematical information retrieval. *The Information Retrieval Series (Springer)*, 43:169–185, 2021. doi:10.1007/978-981-15-5554-1_12.

2 Tatsuya Akutsu. A relation between edit distance for ordered trees and edit distance for Euler strings. *Information Process. Letters*, 100:105–109, 2006. doi:10.1016/j.ipl.2006.06.002.

3 Tatsuya Akutsu, Jesper Jansson, Atsuhiro Takasu, and Takeyuki Tamura. On the parameterized complexity of associative and commutative unification. *Theoretical Computer Science*, 660:57–74, 2017. doi:10.1016/j.tcs.2016.11.026.

4 Tatsuya Akutsu, Takeyuki Tamura, Daiji Fukagawa, and Atsuhiro Takasu. Efficient exponential-time algorithms for edit distance between unordered trees. *Journal of Discrete Algorithms*, 25:79–93, 2014. doi:10.1016/j.jda.2013.09.001.

5 Lázló Babai. Canonical form for graphs in quasipolynomial time: preliminary report. In *51st ACM Symp. Theory of Computing*, pages 1237–1246, 2019. doi:10.1145/3313276.3316356.

6 Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:217–239, 2005. doi:10.1016/j.tcs.2004.12.030.

7 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms*, 6(1):2, 2009. doi:10.1145/1644015.1644017.

8 Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13:113–129, 2010. doi:10.1007/s10044-008-0141-y.

9 Martin Grohe, Daniel Neuen, and Pascal Schweitzer. A faster isomorphism test for graphs of small degree. In *59th IEEE Symp. Foundations of Computer Science*, pages 89–199, 2018. doi:10.1109/FOCS.2018.00018.

10 Shahab Kamali and Frank W. Tompa. A new mathematics retrieval system. In *19th ACM Int. Conf. Information and Knowledge Management*, pages 1413–1416, 2010. doi:10.1145/1871437.1871635.

11 Deepak Kapur and Paliath Narendran. Complexity of unification problems with associative-commutative operators. *Journal of Automated Reasoning*, 9:261–288, 1992. doi:10.1007/BF00245463.

12 Kevin Knight. Unification: a multidisciplinary survey. *ACM Computing Surveys*, 21:93–124, 1989. doi:10.1145/62029.62030.

13 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

14 Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. In *62nd IEEE Symp. Foundations of Computer Science*, pages 792–803, 2021. doi:10.1109/FOCS52979.2021.00082.

15 Tam T. Nguyen, Kuiyu Chang, and Siu Cheung Hu. A math-aware search engine for math question answering system. In *21st ACM Int. Conf. Information and Knowledge Management*, pages 724–733, 2012. doi:10.1145/2396761.2396854.

16 Kuo Chung Tai. The tree-to-tree correction problem. *Journal of ACM*, 26:422–433, 1979. doi:10.1145/322139.322143.

17 Sean T. Vittadello and Michael P. H. Stumpf. Model comparison via simplicial complexes and persistent homology. *Royal Society Open Science*, 8(10):211361, 2020. doi:10.1098/rsos.211361.

18 Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problem. *SIAM Journal on Computing*, 18:1245–1262, 1989. doi:10.1137/0218082.

19 Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, 1992. doi:10.1016/0020-0190(92)90136-J.

20 Wei Zhong and Richard Zanibbi. Structural similarity search for formulas using leaf-root paths in operator subtrees. In *41st European Conference on IR Research*, pages 116–129, 2019. doi:10.1007/978-3-030-15712-8_8.

On the Cop Number of String Graphs

Sandip Das
Indian Statistical Institute, Kolkata, India

Harmender Gahlawat
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Cops and Robber is a well-studied two-player pursuit-evasion game played on a graph, where a group of cops tries to capture the robber. The *cop number* of a graph is the minimum number of cops required to capture the robber. We show that the cop number of a string graph is at most 13, improving upon a result of Gavenčiak et al. [Eur. J. of Comb. 72, 45–69 (2018)]. Using similar techniques, we also show that four cops have a winning strategy for a variant of Cops and Robber, named Fully Active Cops and Robber, on planar graphs, addressing an open question of Gromovikov et al. [Austr. J. Comb. 76(2), 248–265 (2020)].

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Cop number, string graphs, intersection graphs, planar graphs, pursuit-evasion games

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.45

Funding This research was financed by the IFCAM project "Applications of graph homomorphisms" (MA/IFCAM/18/39).
Harmender Gahlawat: Supported by the Israel Science Foundation (ISF) grant no. 1176/18

Acknowledgements We thank Uma kant Sahoo, Dibyayan Chakraborty, and Florent Foucaud for initial discussions on the topic of this paper.

1 Introduction

Cops and Robber is a two-player perfect information pursuit-evasion game played on a graph. One player is referred as *cop player*, who controls a set of *cops*, and the other player is referred as *robber player* and controls a single *robber*. The game starts with the cop player placing each cop on some vertex of the graph, and multiple cops may simultaneously occupy the same vertex. Then, the robber player places the robber on a vertex of the graph. Afterwards, the cop player and the robber player make alternate moves, starting with the cop player. In a cop player move, each cop either moves to an adjacent vertex (along an edge) or stays on the same vertex. In the robber player move, the robber does the same. For simplicity, we will say that the cop (resp. robber) moves in a cop (resp. robber) move instead of saying that the cop (resp. robber) player moves the cop (resp. robber).

A state in the game where one of the cops occupies the same vertex as the robber is called the *capture*. If the cops can capture the robber in a graph, then the cops *win*, and if the robber can evade the capture forever, then the robber *wins*. The *cop number* of a graph G, denoted as $\mathsf{c}(G)$, is the minimum number of cops that can ensure the capture against all the strategies of the robber. For a family $\mathcal{F}$ of graphs, $\mathsf{c}(\mathcal{F}) = \max\{\mathsf{c}(G) \mid G \in \mathcal{F}\}$. In this paper, we consider finite, connected[1], and simple undirected graphs. We denote the robber by $\mathcal{R}$.

[1] The cop number of a disconnected graph is the sum of the cop numbers of its components; hence we assume connectedness.

The game of Cops and Robber was independently introduced by Quilliot [27] and by Nowakowski and Winkler [22], both in 1983, with just one cop. Aigner and Fromme [3] generalized the game to multiple cops and defined the cop number for a graph. The notion of cop number and some fundamental techniques introduced by Aigner and Fromme [3] have resulted in a plethora of rich results on this topic. For more details, we refer the reader to the book by Bonato and Nowakowski [8].

The computational complexity of finding the cop number of a graph is a challenging question in itself. On the positive side, Berarducci and Intrigila [6] provided a backtracking algorithm that decides whether the cop number of a graph is at most k in $O(n^{2k+1})$ time; hence, this is a polynomial-time algorithm for a fixed k. On the negative side, Fomin et al. [11] proved that determining the cop number of a graph is NP-hard as well as W[2]-hard. Moreover, the game was shown to be PSPACE-hard by Mamino [21] and EXPTIME-complete by Kinnersley [18]. Recently, Brandt et al. [10] provided the fine-grained lower bounds, and proved that the time complexity of any algorithm for Cops and Robber is $\Omega(n^{k-o(1)})$ conditioned on SETH, and $2^{\Omega(\sqrt{n})}$ conditioned on ETH.

A *string representation* of a graph is a collection of simple curves on the plane such that each curve corresponds to a vertex of the graph, and two curves intersect if and only if the vertices they represent are adjacent in the graph. The graphs that have string representations are called *string graphs*. Many important graph families like *planar graphs*, *chordal graph*, and *disk graphs* are subfamilies of string graphs [5, 15]. Pach and Toth [23] proved that the number of string graphs on n labeled vertices is at least $2^{\frac{3}{4}\binom{n}{2}}$, arguing that many graphs are string graphs. Cops and Robber is well-studied on graphs having a representation either on the plane or on some surface of higher genus [3, 28, 20, 29]. Gavenčiak et al. [13] studied the cop number of various families of intersection graphs and showed that the cop number for the class of string graphs is at most 15. We improve this result by giving a winning strategy using 13 cops for any string graph.

Several variations of Cops and Robber have been studied, and they vary mainly depending on the capabilities of the cops and the robber. Some of these variations are shown to have correspondence with various width measures of graphs like treewidth [30], pathwidth [24], tree-depth [14], hypertree-width [2], cycle-rank[14], and directed tree-width [17]. Moreover, Abraham et al. [1] defined *cop-decomposition*, which is based on the cop strategy in Cops and Robber game on minor-free graphs provided by Andreae [4], and showed that it has significant algorithmic applications in theory.

Gromovikov et al. [16] studied a variation of the game, called Fully Active Cops and Robber, where the cops, as well as the robber, are forced to move to an adjacent vertex on their respective turns. We say that a cop (or robber) is *active* if it has to move to an adjacent vertex in its every turn. Similarly, we say that a cop (or robber) is *flexible* if, in its turn, it can either move to an adjacent vertex or stay on the same vertex. In Fully Active Cops and Robber, the cops, as well as the robber, are active. The *active cop number* of a graph G, denoted $\mathsf{c_a}(G)$, is the minimum number of cops required to ensure capture in Fully Active Cops and Robber. Gromovikov et al. [16] studied this game on various graph classes and suggested determining the active cop number of planar graphs as an open question. We address this question and show that the active cop number for the class of planar graphs is at most four. We also consider a variation of this game where only the cops are forced to be active. Let $\mathsf{c_A}(G)$ be the minimum number of active cops required to ensure the capture of a flexible robber in G. Observe that, for any graph G, $\mathsf{c_a}(G) \leq \mathsf{c_A}(G)$. We rather show that for a planar graph G, $\mathsf{c_A}(G) \leq 4$.

1.1 Preliminaries

Let u be a vertex of graph G. We define the *open neighbourhood* of u, denoted by $N(u)$, as $\{v : uv \in E(G)\}$. We define the *closed neighbourhood* of u, denoted by $N[u]$, as $N(u) \cup \{u\}$. For a subgraph H of G, define the *closed neighbourhood* of H, denoted by $N[H]$, as $\bigcup_{v \in V(H)} N[v]$. We also define $G - H$ as the graph induced by the vertices that are in G but not in H. For a vertex $x \in V(G)$, we define $G - x$ as the graph induced by vertices in $V(G) \setminus \{x\}$.

Consider two arbitrary vertices $u, v \in V(G)$. By $d(u, v)$, we denote the distance between vertices u and v in G. Let P be a path of G with endpoints u and v. We say that P is a u, v-path. Path P is said to be *isometric* if P is a shortest u, v-path. Moreover, path P is said to be *convex* if every u, v-path $Q \neq P$ is longer than P. We remark that not every pair of vertices is guaranteed to have a convex path between them. In this article, we do a lot of index manipulations on path vertices. Therefore, whenever we mention an isometric v_i, v_j-path P, for $i < j$, then the path is of the form $v_i, v_{i+1}, \ldots, v_{j-1}, v_j$.

Let H be a subgraph of G. We say that some cops are *guarding* H if $\mathcal{R}$ cannot enter a vertex of H without getting captured. Similarly, we define a subset $T \subseteq V(G)$ as the *robber territory* if $\mathcal{R}$ cannot move to a vertex $v \notin T$, without getting captured in the next move.

Let $T \subseteq V(G)$ be the robber territory. A u_0, u_k-path P is *isometric relative* to T, if there is no shorter u_0, u_k-path containing at least one vertex of T. An isometric u_0, u_k-path P relative to T is a *convex path relative* to T, if there exists no vertex $x \in T$ such that $d(u_0, x) = i - 1$ and $x \in N(u_i)$.

G is an *intersection graph* if each vertex $v \in V(G)$ corresponds to a set $\psi(v)$, and $uv \in E(G)$ if and only if $\psi(u) \cap \psi(v) \neq \emptyset$. A string graph G is an intersection graph of *strings*, where each string $\psi(v)$ is a continuous image of the interval $[0, 1]$ into $\mathbb{R}^2$. Given a string graph G, we can generate strings corresponding to each vertex of V such that two strings intersect if and only if the corresponding two vertices are adjacent in G. These strings are said to be a *representation* of graph G. It is a standard assumption that for any string graph G, we can get a representation where the strings are non self-intersecting. So, we assume we have a representation where the strings are non self-intersecting.

1.2 Our Results and Techniques Used

It is well established that the cop number of a graph is well related to the geometry of the graph. This relation was first established by Aigner and Fromme [3], who proved that the cop number of a planar graph is at most three. To show this, they proved the following result, which we will also use in this paper.

▶ **Proposition 1** ([3]). *Let P be an isometric u_0, u_k-path in G. Then one cop can guard P after at most k cop moves.*

A similar idea was used by Beveridge et al. [7], who showed that three cops can prevent the robber from crossing an isometric path in any unit disk graph, and using this proved that nine cops have a winning strategy for unit disk graphs. Later, Gavenčiak et al. [13] proved that the cop number of string graphs is at most 15. To establish this, they proved the following result, which we will also use in this paper.

▶ **Proposition 2** ([13]). *Let u and v be two distinct vertices of G and P be an isometric u, v-path relative to the robber territory $T \subseteq V(G)$. Then five cops can guard $N[P]$ after at most k cop moves.*

At a very high level, the idea of the above strategies is the following. The cops play the game assuming a fixed representation of the graph. The cop player employs three teams of cops, where each team can prevent the robber from crossing an isometric path. The cops

begin by using one team to guard an isometric path, say P_1. Next, the cop player finds another isometric path, say P_2, such that the endpoints of P_1 and P_2 are the same, and guard it using the second team. Now, $\mathcal{R}$ cannot cross the paths P_1 and P_2, and hence is restricted to one of the faces formed by the boundary $P_1 \cup P_2$ in the embedding. Now, we can delete the part of the graph not accessible to $\mathcal{R}$. In the remaining graph, the cop player finds another isometric path, say P_3, such that the endpoints of P_3 are the same as the endpoints of P_1 and P_2, and guard P_3 using the third team of the cops. This further restricts $\mathcal{R}$ to either in a face formed by the boundary $P_1 \cup P_3$ or in a face formed by the boundary $P_2 \cup P_3$. In either case, we can free one team of cops and keep repeating this process, and in each iteration, the robber territory is strictly reduced. Since the robber territory is initially the graph G, which is finite, these three teams eventually capture the robber.

Our main observation is that if an isometric path P is the *convex path*, then in some cases, we can employ a smaller number of cops to prevent the robber from crossing P. More specifically, we have the following lemma, which we prove in Section 2.

▶ **Lemma 3.** *Let u_0 and u_k be two distinct vertices of G and P be a convex u_0, u_k-path relative to the robber territory $T \subseteq V(G)$. Then four cops can guard $N[P]$, after at most k cop moves.*

Using Lemma 3, we prove that four cops can prevent $\mathcal{R}$ from crossing a convex path in a string graph representation. Then, using some novel techniques, we give a strategy such that whenever two teams of cops are employed to guard two isometric paths, one of the teams is guarding a convex path. This directly gives a cop winning strategy using 14 cops for string graphs. We further use some techniques to improve this bound to 13 cops. We have the following result, which we prove in Section 3.

▶ **Theorem 4.** *If G is a string graph, then $\mathsf{c}(G) \leq 13$.*

Petr et al. [26] gave an algorithm that, given a graph G, can decide in $\mathcal{O}(kn^{k+2})$ time if $\mathsf{c}(G) \leq k$. Therefore, for any graph family $\mathcal{F}$, if $\mathsf{c}(\mathcal{F}) \leq \ell$, where $\ell \in \mathbb{N}$, then for any graph $G \in \mathcal{F}$, $\mathsf{c}(G)$ can be computed in $\mathcal{O}(n^{\ell+2})$ time. Thus, we have the following corollary.

▶ **Corollary 5.** *If G is a string graph, then $\mathsf{c}(G)$ can be computed in $\mathcal{O}(n^{15})$ time.*

Aigner and Fromme [3] also showed that for a graph G with girth[2] at least five and minimum degree $\delta(G)$, $\mathsf{c}(G) \geq \delta(G)$. Inspired by this, Gavenčiak et al. [13] established the following interesting relation between the cop number of a graph G, its degeneracy, and hence its chromatic number.

▶ **Proposition 6** ([13])**.** *Let $\mathcal{F}$ be a hereditary class of graphs such that $\mathsf{c}(\mathcal{F}) \leq k$, for $k \in \mathbb{N}$. Then, every graph $G \in \mathcal{F}$ with girth at least five is k-degenerate and therefore, $k+1$-colorable.*

Using Proposition 6, they established that every string graph with girth at least five is 16-colorable. Although the results of Fox and Pach [12] imply that the chromatic number of girth five string graphs is bounded, their results do not mention an explicit numerical bound. Moreover, it is known that the chromatic number of string graphs with girth four is unbounded [25]. We also note here that the chromatic number of girth (at least) five 1-string graphs is at most six [19], where 1-string graphs are the graphs with a string representation where two strings intersect at most once. We have the following corollary on the chromatic number of string graphs using Proposition 6 and Theorem 4.

[2] The *girth* of a graph G is the length of a shortest cycle contained in G.

▶ **Corollary 7.** *If G is a string graph with girth at least five, then G is 13-degenerate and hence, 14-colorable.*

Let *2-BOX* be the family of intersection graphs of axis-parallel rectangles in $\mathbb{R}^2$. It is known that $2 \leq \mathsf{c}$*(2-BOX)*≤ 15 [13]. We improve this result in the following theorem.

▶ **Theorem 8.** *Let 2-BOX be the family of rectangle intersection graphs. Then* $3 \leq \mathsf{c}$*(2-BOX)*≤ 13

Proof. Since *2-BOX* is a subclass of string graphs, the upper bound follows from Theorem 4. To prove the lower bound (i.e., $3 \leq \mathsf{c}$*(2-BOX)*), we observe that the dodecahedron graph, having cop number three [3], is a boxicity 2 graph. For completeness, we give a rectangle intersection representation of the dodecahedron graph in Figure 2 in the Appendix. ◀

We further show that our technique can be used to attain better bounds for different variations of the game on other graph classes as well. In particular, we study Fully Active Cops and Robber on planar graphs. It is known that $\mathsf{c_a}(G) \leq 2 \cdot \mathsf{c}(G)$[16]. Let $\mathcal{P}$ be the class of planar graphs, then trivially $\mathsf{c_a}(\mathcal{P}) \leq 6$. Gromovikov et al. [16] asked as open question what is the value of $\mathsf{c_a}(\mathcal{P})$. We answer this question partially by showing that $\mathsf{c_a}(\mathcal{P}) \leq 4$. To show this, we prove the following lemma in Section 2.

▶ **Lemma 9.** *Let v_0 and v_k be two distinct vertices of G and P be a convex v_0, v_k-path relative to the robber territory $T \subseteq V(G)$. Then, one active cop can guard P against a flexible robber, after at most k cop moves.*

In Lemma 9, the active cop can guard a convex path even if the robber is flexible. Therefore, using Lemma 9 and techniques similar to the ones we use for string graph, we prove the following result in Section A (in the Appendix)[3], a corollary of which is that $\mathsf{c_a}(G) \leq 4$.

▶ **Theorem 10.** *If G is a planar graph, then $\mathsf{c_A}(G) \leq 4$.*

2 Guarding Convex Paths

In this section, we prove Lemma 3 and Lemma 9. We recall that, an isometric u_0, u_k-path P is a convex path relative to the robber territory $T \subseteq V(G)$ if there exists no vertex $x \in T$ such that $d(u_0, x) = i - 1$ and $x \in N(u_i)$.

▶ **Lemma 3.** *Let u_0 and u_k be two distinct vertices of G and P be a convex u_0, u_k-path relative to the robber territory $T \subseteq V(G)$. Then four cops can guard $N[P]$, after at most k cop moves.*

Proof. We mark one cop as the *sheriff*, and the other three cops are said to be its *deputies*. The deputies follow the movements of the sheriff such that when the sheriff is at a vertex u_i, for $0 \leq i \leq k$, the deputies are at vertices u_{i-2}, u_{i-1} and u_{i+1}. Let the vertex u_{k+1} refer to the vertex u_k, and let vertices u_{-1} and u_{-2} refer to the vertex u_0. Let $D_j = \{v \mid d(u_0, v) = j, \text{ if } j < k; \text{ and } d(u_0, v) \geq j, \text{ if } j = k\}$.

[3] In respect of the space constraints, this section has been moved to the Appendix.

Since P is an isometric path relative to T, the sheriff can guard P in at most k steps using Proposition 1. Moreover, it is worth mentioning that the sheriff can do so by staying on the vertices of P. More specifically, after each move of the sheriff, if $\mathcal{R}$ is at a vertex $v \in D_j$, then the sheriff is at vertex u_j. We claim that once the sheriff guards P, these four cops guard $N[P]$.

To prove the above claim, we show that if $\mathcal{R}$ moves to a vertex $x \in N[P]$ (also $x \in T$), then $\mathcal{R}$ gets captured by one of the cops. If $\mathcal{R}$ moves to a vertex in P, then the sheriff will capture the robber as it is guarding P. Let $\mathcal{R}$ moves to a vertex $x \notin V(P)$, $x \in N[P]$, and $x \in D_j$.

Let $1 < j < k$. Since $x \in N[P]$, x is adjacent to at least one vertex of P. Now x cannot be adjacent to a vertex y from $\{u_0, \ldots, u_{j-2}\}$, as through path $u_0, \ldots, y, x$ the distance $d(u_0, x) < j$, which is not possible since $x \in D_j$. Moreover, x cannot be adjacent to a vertex y from $\{u_{j+2}, \ldots u_k\}$, as the path $u_0, \ldots x, y, \ldots u_k$ becomes a shorter u_0, u_k-path than P, which is a contradiction to the fact that P is an isometric path relative to T. Also, x cannot be adjacent to u_{j+1} by the definition of the convex path. Hence, x can only be adjacent to u_{j-1} and u_j, and is adjacent to at least one of them. Since the sheriff is guarding $\mathcal{R}$, it can reach u_j in this cop move, and hence is at one of the vertices from $\{u_{j-1}, u_j, u_{j+1}\}$. In any case, there are cops on both u_j and u_{j-1}. Therefore, one of these cops will capture $\mathcal{R}$ whenever $\mathcal{R}$ enters x.

Similar arguments hold for $j \in \{0, 1, k\}$. If $j = k$, then observe that x can only be adjacent to u_k and u_{k-1}, and both these vertices would be occupied by cops. If $j = 1$, then x can only be adjacent to u_0 and u_1, and both these vertices would be occupied by cops. If $j = 0$, then $x = u_0$ and hence x in on P, and since the sheriff is guarding P, it will capture $\mathcal{R}$.

Hence, these four cops can guard $N[P]$ in at most k steps. ◀

Next, we show that for a convex path P relative to the robber territory T, one active cop can guard P against a flexible robber.

▶ **Lemma 9.** *Let v_0 and v_k be two distinct vertices of G and P be a convex v_0, v_k-path relative to the robber territory $T \subseteq V(G)$. Then, one active cop can guard P against a flexible robber, after at most k cop moves.*

Proof. Let $D_j = \{v \mid d(v_0, v) = j, \text{ if } j < k; \text{ and } d(v_0, v) \geq j, \text{ if } j = k\}$. We claim that if the cop $\mathcal{C}$ can ensure the following invariant, then $\mathcal{C}$ successfully guards P: *after each move of the cop, if the robber is at a vertex in D_i, then $\mathcal{C}$ is at either v_i or v_{i-1}*. Assume that this invariant holds and $\mathcal{R}$ moves to enter a vertex v_r of P from a vertex $u \notin V(P)$. Observe that, since P is a convex path, $u \in D_r \cup D_{r+1}$. Consider the game state just before this move of $\mathcal{R}$. Due to the invariant condition, since $\mathcal{R}$ is at vertex u, the cop $\mathcal{C}$ is either at v_r, v_{r-1}, or v_{r+1}. In any of these case, $\mathcal{C}$ can move to capture $\mathcal{R}$ if $\mathcal{R}$ moves to v_r.

Thus, if $\mathcal{C}$ can maintain this invariant, $\mathcal{C}$ guards P. Now, it remains to show that $\mathcal{C}$ can always reach this invariant and, once achieved, can always maintain it. $\mathcal{C}$ starts at vertex v_0. If $\mathcal{R}$ is at vertex $u \in D_i$, then $\mathcal{C}$ assumes $image(\mathcal{R})$ at vertex v_i. Since $image(\mathcal{R})$ is restricted to P, $\mathcal{C}$ can capture $image(\mathcal{R})$ in at most k cop moves. Once $\mathcal{C}$ captures $image(\mathcal{R})$, observe that we get the invariant. After that, $\mathcal{C}$ follows the following strategy:

1. If $\mathcal{R}$ moves from a vertex $u \in D_i$ to a vertex $w \in D_{i-1}$, then $\mathcal{C}$ moves to vertex v_{j-1} from vertex v_j.
2. If $\mathcal{R}$ moves from a vertex $u \in D_i$ to a vertex $w \in D_{i+1}$, then $\mathcal{C}$ moves to vertex v_{j+1} from vertex v_j.
3. If $\mathcal{R}$ moves from a vertex $u \in D_i$ to a vertex $w \in D_i$:
   a. If $\mathcal{C}$ is at v_i, then it moves to v_{i-1}.
   b. If $\mathcal{C}$ is at v_{i-1}, then it moves to v_i.

Since the above strategy maintains the invariant, this completes our proof. ◀

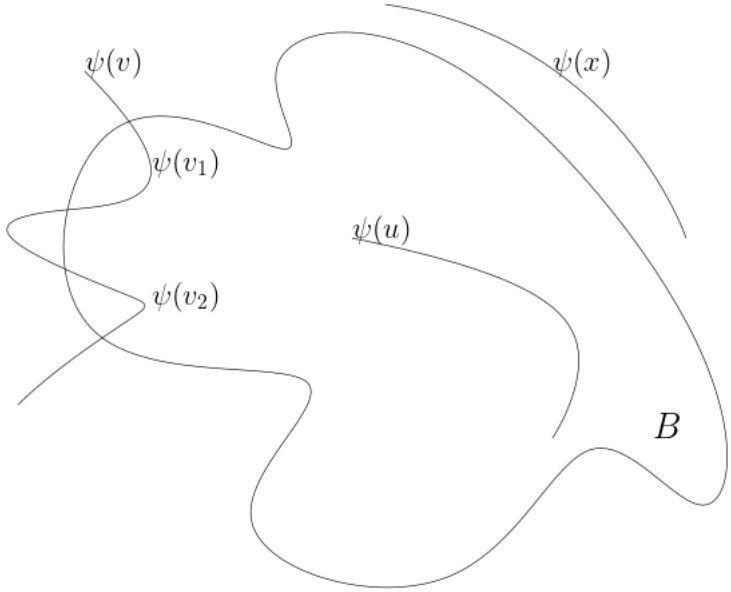


Figure 1 Here $\psi(u)$ is in Ψ_B, $\psi(x)$ is not in Ψ_B, and for $\psi(v)$, strings $\psi(v_1)$ and $\psi(v_2)$ are in Ψ_B.

3 Cops and Robber on String Graphs

3.1 Definitions and Preliminaries

Segments, Faces and Regions. A set $A \subset \mathbb{R}^2$ is *arc-connected* if for any two points $a, b \in A$, the set A contains a curve with endpoints a and b. Consider a fixed string representation Ψ of G. If two strings π and π' intersect at a point p, then we call p as an *intersection point*. In a fixed representation of a string graph G, a string can have multiple intersection points, and two strings can have multiple intersection points in common. A *segment* s of a string π is a maximal continuous part of the string π that does not contain any intersection point other than its endpoints. A string containing k intersection points has $k + 1$ segments.

A *region* is an arc-connected area bounded by some segments of a set of strings in a string representation. A region also includes its boundary. Whenever we mention a region, it should satisfy our region definition. A *face* is a region not containing any intersection point between two strings except on the boundary, and each continuous part of a string in the region intersects the boundary of the region at most once. It is a standard assumption that for a finite string graph G, we can have a representation such that the number of segments, intersection points, and faces is finite.

Representation Restricted to a Region. Consider a region B of representation Ψ. We define the *representation restricted to* B, denoted by Ψ_B, in the following manner. If a string $\psi(v)$ is completely inside B, then we have $\psi(v)$ in Ψ_B also. If $\psi(v)$ is completely outside B, then $\psi(v)$ is not in Ψ_B. If a string $\psi(v)$ is such that some portion of $\psi(v)$ is outside B and some portion of $\psi(v)$ is inside B, then we do the following. Let $s_1, \ldots, s_k$ be the portions of the string $\psi(v)$ such that each endpoint of s_i, for $0 < i \leq k$, is either on the boundary of B or is an endpoint of the string $\psi(v)$, and $s_i \in \Psi_B$. Then, instead of the string $\psi(v)$, we include k new strings. We consider each portion s_i, for $0 < i \leq k$, as a new string $\psi(v_i)$ in Ψ_B. See Figure 1 for an illustration. Let G_B be the string graph corresponding to the representation Ψ_B. Here $V(G_B)$ is defined by the strings in Ψ_B, and $E(G_B)$ is defined by the intersections

between these strings. Observe that, though G_B might contain more vertices than G, the number of vertices in G_B remains finite. Moreover, the number of faces and segments in Ψ_B is not more than that in Ψ. Here, we also say that G_B is the graph G *restricted* to region B.

Relating Curves to Paths. Let Ψ be a fixed representation of a string graph G. Consider a curve π in the representation Ψ. π is composed of some of the segments of the strings from Ψ. Let π be composed of segments $s_1, \ldots, s_\ell$. Furthermore, consider a u_1, u_k-path P in G. We say that the curve π is *related* to path P if each segment $s \in \{s_1, \ldots, s_\ell\}$ is a segment of some string $\psi(u)$, $u \in \{u_1, u_2, \ldots, u_k\}$, and for each string $\psi(u)$, $u \in \{u_1, u_2, \ldots, u_k\}$, there is a segment $s \in \{s_1, \ldots, s_\ell\}$ such that s is a segment of $\psi(u)$. Observe that $\ell \geq k$. Note that multiple curves may relate to the same path, and a curve may be related to multiple paths. For example, consider a complete graph K_n (which is a string graph) and a string representation of K_n, denoted by $\Psi(K_n)$. If we choose a curve that contains at least one segment from each string of $\Psi(K_n)$, then this curve corresponds to every path of length $n-1$ in K_n. We would also like to mention here that the order of segments in the curve might not correspond to the order of vertices in the path.

An *isometric curve* in Ψ is a curve that is related to an isometric path in G. We have the following observation that we use implicitly in our arguments.

▶ **Observation 11.** *Although multiple isometric curves can be related to an isometric path, an isometric curve cannot be related to multiple isometric paths.*

Proof. Consider an isometric u_1, u_k-path P and a curve π related to P. Let $s_1, \ldots, s_l$ be the order of segments in π. Since P is an isometric path, a segment of string $\psi(u_i)$ can only be adjacent to a segment of string $\psi(u_{i-1})$, $\psi(u_i)$, or of string $\psi(u_{i+1})$ in π.

Let $z_1, \ldots, z_k$ be natural numbers such that $z_1 = 1$, $z_k = l$, and $z_1 < z_2 < \cdots < z_k$. Then there exists a sequence $z_1, \ldots, z_k$ such that each segment $s \in \{s_{z_i}, \ldots, s_{z_{i+1}}\}$, for $1 \leq i \leq k-2$, is a segment of either the string $\psi(u_i)$ or the string $\psi(u_{i+1})$, and the segment $s_{z_{i+1}}$ is a segment of the string $\psi(u_{i+1})$. For $i = k-1$, each segment $s \in \{s_{z_i}, \ldots, s_{z_{i+1}}\}$, is a segment of either the string $\psi(u_i)$ or the string $\psi(u_{i+1})$. Thus, π can be related to only one path $u_1, u_2, \ldots, u_k$. Therefore, an isometric curve relates to a convex path. ◀

For ease of arguments in later proofs, we define *monotone curves* in the following manner. Let π be a curve related to an isometric u_1, u_k-path P, and let $s_1, \ldots, s_\ell$ be the order of segments of π. Let $z_1, \ldots, z_{k+1}$ be natural numbers such that $z_1 = 1$, $z_{k+1} = \ell$, and $z_1 < z_2 < \cdots < z_{k+1}$. Then, curve π is said to be a *monotone curve related* to P if there exists a sequence $z_1, \ldots, z_{k+1}$ such that each segment $s \in \{s_{z_i}, \ldots, s_{z_{i+1}}\}$, for $1 \leq i \leq k-1$, is a segment of the string $\psi(u_i)$. For our results, whenever we consider an isometric curve related to an isometric path, we always consider a monotone curve, without mentioning it explicitly.

A curve with endpoints a and b is referred to as an *a, b-curve*. Two curves are said to be *internally disjoint* if they can intersect only at their respective endpoints. Let π be a curve in a fixed representation Ψ. A curve π' is said to be a *sub-curve of* π if π' can be formed by some segments of π. We borrow the following topological lemmas by Gavenčiak et al. [13] that we will use.

▶ **Lemma 12** ([13]). *Let B be a region. If π is an isometric curve and $\pi' \subseteq \pi$ is a sub-curve with $\pi' \subseteq B$, then π' is an isometric curve in Ψ_B.*

▶ **Lemma 13** ([13]). *Let π_1 and π_2 be two internally disjoint isometric a, b-curves, with $a \neq b$, bounding a region R. For any simple a, b-curve $\pi_3 \subseteq R$ containing at least one interior point of R, we have that every arc-connected component of $R \backslash (\pi_1 \cup \pi_2 \cup \pi_3)$ is bounded by two simple and internally disjoint curves π'_i and π'_3 with $\pi'_i \subseteq \pi_i$, $\pi'_3 \subseteq \pi_3$ and $i \in \{1, 2\}$.*

If a curve π is related to an isometric (resp. convex) path P relative to T, then π is referred to as an *isometric* (resp. *convex*) *curve relative to* T. We extend Lemma 12 to accommodate the convex curves in the following lemma.

▶ **Lemma 14.** *Let B be a region of Ψ. If π is a convex curve relative to T and $\pi' \subseteq \pi$ is a sub-curve with $\pi' \subseteq B$, then π' is a convex curve relative to T in Ψ_B.*

Proof. First, we prove that π' is a convex curve in Ψ relative to T. Let the curve π be related to the convex u_0, u_k-path P in G. Then any sub-curve $\pi' \subseteq \pi$ would relate to a u_i, u_j-path $P' = u_i, u_{i+1}, \ldots, u_j$, where $0 \leq i \leq j \leq k$. For contradiction, let us assume that π' is not a convex curve relative to T in Ψ, and hence P' is not a convex path relative to T in G. Thus, there exists a vertex $v \in (V(T) \setminus V(P))$ and some u_ℓ (where $i \leq \ell \leq j$) such that $d(u_i, v) = d(u_i, u_\ell) - 1$ and $u_\ell \in N[v]$. Therefore, $d(u_0, u_i) + d(u_i, v) = d(u_0, u_i) + d(u_i, u_\ell) - 1$. Hence, we have a vertex $v \in (V(T) \setminus V(P))$ such that $d(u_0, v) = d(u_0, u_\ell) - 1$ and $u_\ell \in N[v]$. This contradicts the fact that P is a convex path related to the convex curve π, both relative to T. Hence, π' is a convex curve in Ψ and P' is a convex path in G, both relative to T.

Next, we show that if a curve π' is a convex curve in Ψ relative to T and $\pi' \subseteq \Psi_B$ (for some B of Ψ), then π' is a convex curve in Ψ_B relative to T. Consider two vertices x and y of G corresponding to strings $\psi(x)$ and $\psi(y)$ in Ψ, respectively. Let x' and y' be two vertices in G_B such that $\psi(x')$ is a portion of $\psi(x)$ and $\psi(y')$ is a portion of $\psi(y)$. Then observe that $d(x', y')$ in G_B cannot be less than $d(x, y)$ in G.

Now consider a vertex v' in G_B such that $v' \notin V(P')$, corresponding to string $\psi(v')$ in Ψ_B, such that $u_\ell \in N[v']$. Let $\psi(v)$ be a string in Ψ, corresponding to vertex v such that $v \notin V(P)$, such that $\psi(v')$ is a portion of string $\psi(v)$ in Ψ. Hence u_ℓ is also a neighbour of v in G. Since P' is a convex path in G, either $d(u_i, v) = d(u_i, u_\ell) + 1$ or $d(u_i, v) = d(u_i, u_\ell)$ in G. Hence, $d(u_i, v) \geq d(u_i, u_\ell)$ in G. Since $d(x', y')$ in G_B cannot be less than $d(x, y)$ in G, $d(u_i, v') \geq d(u_i, u_\ell)$. Thus, there cannot be any vertex v' in $V(G_B) \setminus V(P')$ such that $u_\ell \in N[v']$ and $d(u_i, v') = d(u_i, u_\ell) - 1$. Hence, P' is a convex path in G_B and π' is a convex curve in Ψ_B, both relative to T. ◀

We note here that if a path P is an isometric (resp. convex) path relative to $T \subseteq V(G)$, then P is an isometric (resp. convex) path relative to every subset $T' \subseteq T$. Similarly, if a curve π is an isometric (resp. convex) curve relative to T, then π is an isometric (resp. convex) curve relative to every subset $T' \subseteq T$. Moreover, for a curve π related to a path P, we say that π is guarded if $N[P]$ is guarded.

3.2 Bounding the Robber Region

Geometric Robber Territory. Consider a fixed string representation Ψ of a string graph G. We extend the definition of robber territory to the representation in the following manner. Consider a region B. Let $\mathcal{R}$ be on a vertex u such that all points of the string $\psi(u)$ are inside the region B and $\psi(u)$ does not intersect the boundary of B. Then we say that Ψ_B is the *geometric robber territory* if $\mathcal{R}$ cannot move to a vertex v such that the string $\psi(v)$ intersects the boundary of B, without getting captured. We note that although the strings that intersect the boundary of B might be in Ψ_B, they are not accessible to $\mathcal{R}$ when we say that Ψ_B is the geometric robber territory. Below, we show three ways to restrict the geometric robber territory.

Let B be a region bounded by two internally disjoint a, b-curves π_1 and π_2. Then we also denote Ψ_B by Ψ_{π_1, π_2}. Note that, here B is the geometric robber territory if $\mathcal{R}$ is on a vertex $v \in V(G_B) \setminus (N[P_1] \cup N[P_2])$ and $\mathcal{R}$ cannot move to a vertex $u \in N[P_1 \cup P_2]$. Hence, we have the following observation.

▶ **Observation 15.** *Let π_1 and π_2 be two disjoint a, b-curves and $\mathcal{R}$ is in the region Ψ_{π_1,π_2}. If both curves π_1 and π_2 are guarded, then $\mathcal{R}$ cannot leave the region Ψ_{π_1,π_2} without getting captured, and Ψ_{π_1,π_2} becomes the geometric robber territory.*

Consider a fixed string representation Ψ of a string graph G in $\mathbb{R}^2$. We say that a string $\psi(v)$ is a *top-most string* (*bottom-most string*) if some point on $\psi(v)$ has the highest (lowest) *y-coordinate* in Ψ. Here, we also say that v is a *top-most vertex* (*bottom-most vertex*). Let u and v be two distinct vertices of G such that u is a top-most and v is a bottom-most vertex. Let a and b be points on strings $\psi(u)$ and $\psi(v)$, respectively, such that a and b has the highest and lowest *y-coordinate* in Ψ, respectively. Then, an a, b-curve π, related to an isometric u, v-path P, is referred to as a *top-bottom* curve. Note that this curve may not be unique.

Observe that if a vertex $x \notin N[P]$, then $\psi(x)$ lies either completely on the left of π or completely on the right of π, and $\psi(x)$ does not intersect with π. If a string $\psi(x)$ lies on the left (or right) of the curve π, then we also say that vertex x lies on the left (or right) of P. We say that the robber *crosses* the curve π (or the path P) if $\mathcal{R}$ moves (in some finite rounds) from a vertex u, completely on the left of π, to a vertex v, completely on the right of π, or vice versa.

We extend this idea of bounding a region B with two internally disjoint curves π_1 and π_2, to bounding the region on the *left* or the *right* of a top-bottom curve π. The region B on the left (right) of the curve π contains the points both on π and on the left (right) of π. Here Ψ_B is defined analogously and is also denoted by $\Psi_{\pi,L}$ ($\Psi_{\pi,R}$). We have the following observation.

▶ **Observation 16.** *Let π be a top-bottom curve related to an isometric path P. Then five cops can restrict the geometric robber territory to either $\Psi_{\pi,L}$ or $\Psi_{\pi,R}$.*

Proof. The curve π is a continuous curve from a top-most point a to a bottom-most point b in the string representation. Hence, if a vertex x is on left of P and a vertex y is on right of P, every path from x to y passes through a vertex of $N[P]$. Now, five cops can guard $N[P]$ using Proposition 2. This restricts $\mathcal{R}$ to cross the curve π. Hence, if $\mathcal{R}$ was on a vertex x such that $\psi(x)$ is on the left (right) of π, then $\Psi_{\pi,L}$ ($\Psi_{\pi,R}$) becomes the geometric robber territory. ◀

The following observation provides one more way to bound the robber territory.

▶ **Observation 17.** *Let x be a cut vertex of G such that $G - x$ gives a connected component G'. If $\mathcal{R}$ is on a vertex $v \in V(G')$ and a cop is occupying the vertex x, then $\mathcal{R}$ is restricted to $V(G')$ and $V(G')$ becomes the robber territory.*

We also define isometric (resp. convex) paths relative to geometric robber territories. We say that a path P is *isometric* (resp. *convex*) *path relative* to $\Psi_B = \Psi_{\pi_1,\pi_2}$ if P is an isometric (resp. convex) path relative to $T = V(G_B) \setminus V(P_1 \cup P_2)$. Similarly, a curve π is *isometric* (resp. *convex*) *curve relative* to Ψ_{π_1,π_2} if π is an isometric (resp. convex) curve relative to $T = V(G_B) \setminus V(P_1 \cup P_2)$.

3.3 Extending an Isometric Path/Curve

Informally speaking, in this section, we show that if one team of cops is guarding an isometric path and we want to employ a new team of cops to guard another isometric path, then the new team can always find an isometric path such that in the region bounded by some specific curves of these paths, the first path is a convex path relative to the region bounded.

Consider a fixed representation Ψ of a string graph G. Let $\Psi_{\pi_1,z}$, where $z \in \{\pi_2, L, R\}$, be the geometric robber territory. Let π_1 be an isometric curve relative to $\Psi_{\pi_1,z}$. Then we say that a curve $\pi \in \Psi_B$ is an *extended curve* of π_1 if in the region Ψ_{π',π'_1} bounded by any internally disjoint curves $\pi' \subseteq \pi$ and $\pi'_1 \subseteq \pi_1$, the curves π'_1 and π' are a convex curve and an isometric curve relative to Ψ_{π',π'_1}, respectively; and in the region $\Psi_{\pi,z}$, the curve π is an isometric curve relative to $\Psi_{\pi,z}$. We also say that the path P related to π is an extended path of the path P_1 related to π_1. We would like to note here that there might be multiple extended paths of an isometric path. In this section, we show that if 5 cops are guarding $N[P_1]$, then using at most 4 extra cops (total 9 cops), we can reduce the geometric robber territory to either Ψ_{π',π'_1} or to $\Psi_{\pi,z}$. We have the following lemma.

▶ **Lemma 18.** *Consider a fixed representation Ψ of a string graph G. Let $\Psi_{\pi_1,z}$, where $z \in \{\pi_2, L, R\}$, be the geometric robber territory and let π_1 be an isometric curve relative to $\Psi_{\pi_1,z}$ guarded by 5 cops. If π_1 is not a convex curve in $\Psi_{\pi_1,z}$, then we can find an extended curve π of π_1, and restrict the geometric robber territory to either $\Psi_{\pi,z}$ or to Ψ_{π',π'_1} where $\pi' \subseteq \pi$ and $\pi'_1 \subseteq \pi_1$, using at most four extra cops.*

Proof. In the first part of this proof, we show how to find an extended curve of π_1, if it exists. Let P_1 and P_2 be the isometric paths related to the a, b-curves π_1 and π_2, respectively; and let u_0 and u_k be the endpoints of P_1 and P_2. If $z \in \{L, R\}$, then let $P_2 = \phi$. If there is no u_0, u_k-path in $G_{\pi,x}$ other than P_1 and P_2, then we say that the curve π cannot be extended. Note that here π_1 is a convex path in $\Psi_{\pi_1,x}$.

Let G_B be the graph corresponding to the geometric robber territory $\Psi_B = \Psi_{\pi_1,x}$. If P_1 is a convex path relative to $\Psi_{\pi_1,x}$, then we find a shortest u_0, u_k-path P in G_B other than P_1 and P_2. Observe that P is an isometric path relative to $\Psi_{\pi_1,x}$, and also an extended path of P_1. Here, we can simply free one cop from P_1 (since P_1 is convex path relative to $\Psi_{\pi_1,x}$) and use it along with 4 new cops to guard $N[P]$, and we are done. Thus, we can fix any a, b-curve $\pi \subseteq \Psi_B$ related to P as an an extended curve of π_1.

If P_1 is not a convex path relative to $\Psi_{\pi_1,x}$, then we do the following. Find the least i such that there is a vertex $x \in V(G_B) \setminus V(P_1 \cup P_2)$ with $d(u_0, x) = i - 1$ and $u_i \in N(x)$. Now consider the string $\psi(u_i)$ in $\Psi_{\pi_1,x}$. Let p_u be the intersection point of strings $\psi(u_i)$ and $\psi(u_{i-1})$ and let p_b be the intersection point of strings $\psi(u_i)$ and $\psi(u_{i+1})$, in the curve π_1. We note here that these intersection points p_u and p_b are well defined since we are considering monotone curves. Next, we define sub-curves π_u, π_p, and π_b of the curve $\psi(u_i)$. Let $\pi_p = \pi_1 \cap \psi(u_i)$. Let π_u be the maximal continuous sub-curve of $\psi(u_i)$ such that $\pi_u \cap \pi_1 = p_u$. Similarly, let π_b be the maximal continuous sub-curve of $\psi(u_i)$ such that $\pi_b \cap \pi_1 = p_b$. We note here that one or both of π_u and π_b might be empty.

Let $X \subseteq V(G_B) \setminus V(P_1 \cup P_2)$ such that $X = \{x \mid d(u_0, x) = i - 1 \text{ and } u_i \in N(x)\}$. Let $X_\epsilon \subseteq X$, where $\epsilon \in \{u, p, b\}$, such that $X_\epsilon = \{x \mid \psi(x) \cap \pi_\epsilon \neq \phi\}$. Now we find a suitable vertex $v \in X$ in the following manner.

1. If X_u is not empty, then we find a vertex $v \in X_u$ such that an intersection point of the string $\psi(v)$ and curve π_u is closest to the point p_u along the curve π_u. We mark this intersection point as p and we set $p' = p_u$.
2. If X_u is empty and X_p is not empty, then we find a vertex $v \in X_p$ such that an intersection point of the string $\psi(v)$ and curve π_p is closest to the point p_u along the curve π_p. We mark this intersection point as p and we set $p' = p$.
3. If X_u and X_p are empty, then we find a vertex $v \in X_b$ such that an intersection point of the string $\psi(v)$ and curve π_b is closest to the point p_b along the curve π_b. We mark this intersection point as p and we set $p' = p_b$.

Now consider an isometric u_0, v-path P_q relative to $\Psi_{\pi_1,x}$, and let π_q be an isometric a, p-curve related to path P'. Moreover, let π_r denote the p, p'-curve along $\psi(u_i)$, and π_s denote the p', b-subcurve of the curve π_1 We compose the curve $\pi = \pi_q \cup \pi_r \cup \pi_s$. Observe that π is an extended curve of π_1, and P (path related to π) is an extended path of P_1.

In the next part of the proof, we show that if five cops are guarding the closed neighborhood of P_1 (with Ψ_{π_1,π_2} being the geometric robber territory), then a total of nine cops can guard both $N[P]$ and $N[P_1]$ with Ψ_{π_1,π_2} being the geometric robber territory.

We would like to remind here that to guard a convex w_0, w_l-path Q, four cops can guard $N[Q]$ by the sheriff guarding the path Q and the deputies being at vertices w_{j-2}, w_{j-1} and w_{j+1}, when the sheriff is at vertex w_j. Similarly, five cops can guard the closed neighbourhood of an isometric w_0, w_l-path Q, by the sheriff guarding the path Q and the deputies being at vertices $w_{j-2}, w_{j-1}, w_{j+1}$ and w_{j+2}, when the sheriff is at vertex w_j. Let us denote the deputy that moves to vertex w_{j+2} when the sheriff moves at w_j as the *special deputy*.

Consider the paths P and P_1. Note that both paths have the same length. Let the vertices of path P be denoted by $v_0, \ldots, v_k$. Note that $u_0 = v_0$ and for $l \geq i$, $u_l = v_l$. Now, we have that five cops are guarding P_1. The main idea is that in the u_0, u_i-subpath of path P_1, only four cops are required, and if we keep the special deputy at v_{j+2} when the sheriffs are at u_j and v_j, then it serves as the special deputy for both paths P and P_1 (because in path P_1, the special deputy is required only at vertices from $u_i, \ldots, u_k$, and note that these vertices are same as vertices $v_i, \ldots, v_k$). The extra four cops move on P such that the sheriff guards P and the deputies are at v_{j-2}, v_{j-1}, and v_{j+1}, when the sheriff is at v_j. When the sheriff successfully guards P, let it be at the vertex v_l. If $l \geq i$, then we have already achieved the goal as the special deputy is already at v_{l+2} (since $v_{l+2} = u_{l+2}$ here). Otherwise $l < i$, and in this case, the special deputy is at u_{l+2}. Now, the special deputy starts moving towards u_i irrespective of the moves of the sheriffs. Once it reaches u_i, it moves aiming to reach v_{l+2} when the deputies are at u_l and v_l. Once the special deputy reaches such a vertex, we have the desired state.

Now, if $\mathcal{R}$ is in the region $\Psi_{\pi,z}$, then we can free the cops from π_1 and $\mathcal{R}$ is restricted to $\Psi_{\pi,z}$. Otherwise $\mathcal{R}$ is restricted to the region Ψ_{π',π'_1} where $\pi' \subseteq \pi$ and $\pi'_1 \subseteq \pi_1$. ◀

3.4 Algorithm for String Graphs

In this section, we show that for a string graph G, $\mathsf{c}(G) \leq 13$, by giving a winning strategy using 13 cops for any string graph. Let Ψ be a fixed representation of G. First, we provide the intuition for our strategy. We define three "favorable" game states. Then we show that whenever we are in a favorable game state, 13 cops can force the game to another favorable game state such that the geometric robber territory gets reduced.

Let $\mathcal{R}$ be restricted to Ψ_B. Moreover, let u and v be two distinct vertices in G_B. For our strategy, first we define three game states, *state 1*, *state 2*, and *state 3* as follows:

1. *State 1*: Let u be a top-most and v be a bottom-most vertex in G_B. Then five cops are guarding the closed neighbourhood of an isometric u, v-path P in G_B. Note that, in Ψ_B this restricts $\mathcal{R}$ to either $\Psi_{\pi,L}$ or $\Psi_{\pi,R}$, where π is a curve related to P (Observation 16).
2. *State 2*: The region B is bounded by two internally disjoint curves π_1 and π_2 such that π_1 is a convex curve in Ψ_{π_1,π_2} and π_2 is an isometric curve in Ψ_{π_1,π_2}, both relative to Ψ_{π_1,π_2}. Then five cops are guarding π_2 and four cops are guarding π_1 (total 9 cops).
3. *State 3*: Let x be a vertex in G such that G_B is a connected component of $G - x$. If a cop is occupying the vertex x and $\mathcal{R}$ is in G_B, then observe that $\mathcal{R}$ is restricted to G_B (Observation 17). Let u be a top-most vertex and v be a bottom-most vertex in G_B, and

P be an isometric u, v-path relative to $T = V(G_B)$. Then one cop is occupying vertex x and five cops are guarding $N[P]$. Moreover, $\mathcal{R}$ and $\psi(x)$ are on the same side of each curve π related to P.

State 1, state 2, and state 3 are referred to as the *safe states*. We have the following lemma, which is central to our algorithm.

▶ **Lemma 19.** *Consider a fixed representation Ψ of a string graph G. Let $\mathcal{R}$ be in a region B of Ψ, and Ψ_B be the geometric robber territory. Let the game be in a safe state S. Then 13 cops can force the game to a safe state S' and the geometric robber territory to $\Psi_{B'} \subset \Psi_B$, in a finite number of moves.*

Proof. Depending upon the state S of the game, we do the following:

1. **S = state 1**: Let u be a top-most and v be a bottom-most vertex in G_B, and P be the isometric u, v-path such that $N[P]$ is guarded by 5 cops. Let π be a curve related to path P and let π defines B. Observe that the geometric robber territory is either $\Psi_{\pi,L}$ or $\Psi_{\pi,R}$. Without loss of generality, let us assume that $\mathcal{R}$ is restricted to the right of π and hence $\Psi_{\pi,R}$ is the geometric robber territory. We extend the curve π in $\Psi_{\pi,R}$ and let π', related to a path P', be an extended curve of π. Now, one of the following scenarios is possible:
   - **a.** Curve π cannot be extended: It is possible only if there is no u, v-path in G_B other than P. Let $\mathcal{R}$ be in a connected component G' of $G_B - P$. In this case, we claim that there is a unique vertex $x \in V(P)$ such that x has a neighbor in G'. For contradiction, assume that there is some other vertex $y \neq x$ in P such that y has some neighbor in G'. Then consider the path Q formed by the vertices of u, x-path along P, followed by a shortest x, y-path in $G' \cup \{u, v\}$, followed by the y, v-path along P. Here Q is a path other than P, and thus we have a contradiction. Thus x is a cut vertex such that $G_B - x$ gives G' as a component.
     We guard x using one cop and free other cops from P. Now, find a top-most vertex u' and a bottom-most vertex v' in G' and an isometric u', v'-path in G'. Now, consider a top-down curve π' corresponding to path P' and guard π' using five cops. If $\mathcal{R}$ and x are on the same side of P', then we are in the safe state 3. If $\mathcal{R}$ and x are on opposite sides of π', then we can free cop on x, and we are in the safe state 1. In both cases, at least the segments corresponding to the vertices of $V(P) - \{x\}$ will be removed from the geometric robber territory.
   - **b.** $\mathcal{R}$ is on the same side of π and π': Since π' is a top-bottom curve and π' is guarded by five cops, we can free the cops on curve π. Hence, the geometric robber territory is now $\Psi_{\pi',R}$ (since $\mathcal{R}$ is in the right of both π and π'). Also, $\Psi_{\pi',R} \subset \Psi_{\pi,R}$ since the region bounded between π and π' is in $\Psi_{\pi,R}$ but not in $\Psi_{\pi',R}$.
   - **c.** $\mathcal{R}$ is in the region bounded by two curves π_1 and π_1' such that $\pi_1 \subseteq \pi$ and $\pi_1' \subseteq \pi'$: By the definition of extended curve, we know that π_1 is a convex curve and π_1' is an isometric curve, both relative $\Psi_{\pi_1,\pi_1'}$. Hence using Lemma 18, we can restrict the geometric robber territory to $\Psi_{\pi_1,\pi_1'}$ using at most 9 cops. Hence we are in the safe state 2. For the sake of simplicity, to prove that the geometric robber territory decreases in this case, we prove it for state 2, and whenever this case occurs, we execute this Lemma again for state 2.
2. **S = state 2**: Let B be bounded by two internally disjoint a, b-curves π and π', and $\Psi_{\pi,\pi'}$ be the geometric robber territory. Let π and π' are related to u, v-paths P and P', respectively. Moreover, let π be a convex curve in $\Psi_{\pi,\pi'}$ and π' be an isometric curve in $\Psi_{\pi,\pi'}$, both relative to $\Psi_{\pi,\pi'}$. Also, four cops are guarding π and five cops are guarding π'.

Now, if the curve π' can be extended, then we extend the curve π' using Lemma 18. Let π_1 be an extended curve of π'. Also, let P_1 be the path related to π_1 is an extended curve of π'. Now, using Lemma 18, we can guard both π_1 and π' using at most nine cops. Now, if $\mathcal{R}$ is restricted in Ψ_{π,π_1}, then we can free the cops guarding π', and we reach safe state 2. Note that the region bounded by curves π' and π_1 is removed from the geometric robber territory. If $\mathcal{R}$ is restricted in $\Psi_{\pi'',\pi_1'}$ such that $\pi'' \subseteq \pi'$ and $\pi_1' \subseteq \pi_1$, then note that we can free the cops guarding π. Observe that the curve π is removed from the geometric robber territory in this case.
Suppose we cannot extend the curve π' (that is, there is no u, v-path in G_B other than P and P'). Then observe that the vertices of the connected component of $G_B - (P \cup P')$ containing $\mathcal{R}$ can be connected to only one vertex x of $P \cup P'$ (Proof is similar to the argument in case 1(a)). We move one cop to vertex x and free all other cops. Now, we are in a situation similar to that of step 1(a). Hence we follow the same steps. Note that we also reduce the geometric territory of $\mathcal{R}$ in this step.

3. **S = state 3**: Let x be a vertex such that G_B is a connected component of $G - x$. Consider the representation $\Psi' \subset \Psi$ such that $\Psi' = \{\psi(u) \mid u \in G_B\}$. Let u and v be a top-most and bottom-most vertex of G_B, respectively. Also, let P be the isometric u, v-path such that $N[P]$ is guarded by five cops, and one cop is occupying the vertex x. Moreover, both $\mathcal{R}$ and x are on the same side of P. Without loss of generality, let us assume that they are on the right of P. Since x is occupied by a cop and $N[P]$ is guarded by cops, observe that the geometric robber territory is $\Psi'_{\pi,R}$, where π is a curve related to P. Now, if the curve π can be extended, then we extend the curve π in $\Psi'_{\pi,R}$ (using Lemma 18) and let π_1 be the extended curve of π. Also let P_1 be the path related to π_1. If $\mathcal{R}$ and x are on the same side of π_1, then we can free cops from P, and we are in the safe state 3. Here, the geometric robber territory is reduced by the region bounded between π and π_1.
If $\mathcal{R}$ is in the region bounded by two internally disjoint curves π' and π_1' such that $\pi' \subseteq \pi$ and $\pi_1' \subseteq \pi_1$, then we are in the safe state 2 (by the definition of extended curves). Now, we free the cop guarding x. Here, the geometric robber territory is reduced by some segments of $\psi(x)$, at least.
If the curve π cannot be extended in $\Psi'_{\pi,R}$, then there exists a vertex $y \in V(P)$ such that vertices in $G_B - y$ gives a connected component $G_{B'}$ containing $\mathcal{R}$. If x is not adjacent to any vertex in $V(G_{B'})$, then we are in a situation similar to 1(a), and we follow the same steps. If x is adjacent to some vertex in $V(G_{B'})$, then We place one cop on y and free other cops from P. Now, we find a top-most vertex u' and bottom-most vertex v' in $G_{B'}$ and find an isometric u', v'-path P_1 in $G_{B'}$. Now, five cops guard $N[P']$. Consider a top-bottom curve π' related to P'. Now, either x and $\mathcal{R}$ lie on the same side of π' or y and $\mathcal{R}$ lie on the same side of π'. In both cases, we are in the safe state 3. Also, observe that each segment s such that s is a segment of path P and s is not a segment of string $\psi(y)$ is reduced from the geometric robber territory. Hence the geometric robber territory reduces in this step.

This completes the proof of our lemma. ◀

Now we prove the main result of this section.

▶ **Theorem 4.** *If G is a string graph, then $\mathsf{c}(G) \leq 13$.*

Proof. We give a cop strategy to prove our claim. Consider a fixed representation Ψ of a string graph G. We first show that at most 13 cops can force the robber to a safe state. Initially, let the robber territory be Ψ and $G_B = G$. Cops find a top-most vertex u and a

bottom-most vertex v in G_B and find an isometric u, v-path P in G_B. Now, five cops guard $N[P]$. This restricts the robber either to the left or to the right of P. Now we are in the safe state 1.

After this, until the robber is captured, we use Lemma 19 to reduce the geometric robber territory. Since we have a finite graph with a finite representation and cops can reduce the geometric robber territory in every iteration of Lemma 19 using at most 13 cops, these 13 cops will eventually capture the robber. ◀

4 Final Remarks and Future Directions

We proved that the cop number of a string graph is at most 13. But currently, we do not know any string graph having cop number at least four. Thus, for the class of string graphs $\mathcal{S}$, $3 \leq \mathsf{c}(\mathcal{S}) \leq 13$. One immediate open question is to improve this bound by either giving a strategy for fewer cops or by giving an explicit construction of a string graph having cop number at least four. It might also be interesting to tighten the bounds on the active cop number of planar graphs.

Cops and Robber is also well-studied with regard to graph genus. Quillot [28] showed that for a graph G having genus g, $\mathsf{c}(G) \leq 2g + 3$. He used an "unfolding" technique where cops find and guard two isometric paths such that "removing" these paths from the graph reduces the genus of the graph by one. Let g-GENUS STRING be the class of graphs admitting a string representation on an orientable surface of genus g. Gavenčiak et al. [13] also used similar unfolding techniques to show that $\mathsf{c}(g\text{-GENUS STRING}) \leq 10g + 15$. For this purpose, they use 10 cops to unfold a genus by guarding the closed neighborhood of two appropriate isometric paths, and then finally capturing the robber in a genus 0 string graph using 15 cops. Theorem 10, along with their unfolding techniques, gives the following immediate corollary.

▶ **Corollary 20.** $\mathsf{c}(g\text{-}\textit{GENUS STRING}) \leq 10g + 13$.

For graphs having a planar representation on a surface of genus g, better unfolding techniques have been used. Let G be a graph having genus g. Schroeder [29] showed that $\mathsf{c}(G) \leq \lfloor \frac{3g}{2} \rfloor + 3$. Later, Bowler et al. [9] improved the upper bound further and proved that $\mathsf{c}(G) \leq \frac{4g}{3} + \frac{10}{3}$. It will be interesting to see if similar techniques can be used to improve the bounds on $\mathsf{c}(g\text{-GENUS STRING})$. Moreover, we propose the following question.

▶ **Question 21.** *Let C be an isometric cycle in G. What is the least number of cops that can guard $N[C]$ in G?*

Observe that if the answer to above question is some constant $c < 10$, then we can unfold a genus by c cops and therefore, we have $\mathsf{c}(g\text{-GENUS STRING}) \leq c \cdot g + 13$.

Another interesting direction would be to see if the techniques used in this paper can be used to improve the cop number of unit disk graphs from 9 (Beveridge et al. [7]) to 7. Moreover, another interesting question can be to improve the bounds on $\mathsf{c}(\textit{2-Box})$.

References

1 I. Abraham, C. Gavoille, A. Gupta, O. Neiman, and K. Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM Journal on Computing*, 48(3):1120–1145, 2019. doi:10.1137/17M1112406.

2 I. Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, 2004.

3 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.

4 T. Andreae. On a pursuit game played on graphs for which a minor is excluded. *Journal of Combinatorial Theory Series B*, 41(1):37–47, 1986.

5 A. Asinowski, E. Cohen, M.C. Golumbic, V. Limouzy, M. Lipshteyn, and M. Stern. Vertex intersection graphs of paths on a grid. *Journal of Graph Algorithms and Applications*, 16(2):129–150, 2012.

6 A. Berarducci and B. Intrigila. On the cop number of a graph. *Advances in Applied mathematics*, 14(4):389–403, 1993.

7 A. Beveridge, A. Dudek, A. Frieze, and T. Müller. Cops and robbers on geometric graphs. *Combinatorics, Probability and Computing*, 21(6):816–834, 2012.

8 A. Bonato and R. Nowakowski. *The Game of Cops and Robbers on Graphs*. American Mathematical Society, 2011.

9 N. Bowler, J. Erde, F. Lehner, and M. Pitz. Bounding the cop number of a graph by its genus. *SIAM Journal on Discrete Mathematics*, 35(4):2459–2489, 2021.

10 S. Brandt, S. Pettie, and S. Uitto. Fine-grained lower bounds on cops and robbers. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.9.

11 F. Fomin, P. Golovach, J. Kratochvíl, N. Nisse, and K. Suchan. Pursuing a fast robber on a graph. *Theoretical Computer Science*, 41(7-9):1167–1181, 2010.

12 J. Fox and J. Pach. A separator theorem for string graphs and its applications. In S. Das and Uehara R., editors, *WALCOM: Algorithms and Computation*, volume 5431 of *Lecture Notes in Computer Science(LNCS)*, pages 1–14, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-00202-1_1.

13 T. Gavenčiak, P. Gordinowicz, V. Jelínek, P. Klavík, and J. Kratochvíl. Cops and robbers on intersection graphs. *European Journal of Combinatorics*, 72:45–69, 2018.

14 A. C. Giannopoulou, P. Hunter, and D. M. Thilikos. Lifo-search: A min–max theorem and a searching game for cycle-rank and tree-depth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.

15 D. Gonçalves, L. Isenmann, and C. Pennarun. Planar graphs as l-intersection or l-contact graphs. In *SODA*, pages 172–1844, 2018.

16 I. Gromovikov, W. B. Kinnersleyb, and B. Seamone. Fully active cops and robbers. *Australian Journal of Combinatorics*, 76(2):248–265, 2020.

17 T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory Series B*, 82(1):138–154, 2001.

18 W. B. Kinnersley. Cops and robbers is exptime-complete. *Journal of Combinatorial Theory Series B*, 111:201–220, 2015.

19 A. V. Kostochka and J. Nešetřil. Coloring relatives of intervals on the plane, i: Chromatic number versus girth. *European Journal of Combinatorics*, 19(1):103–110, 1998.

20 F. Lehner. On the cop number of toroidal graphs. *Journal of Combinatorial Theory, Series B*, 151:250–262, 2021.

21 M. Mamino. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science*, 477:48–56, 2013.

22 R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3):235–239, 1983.

23 J. Pach and G. Toth. How many ways can one draw a graph? In *GD*, pages 47–58. Springer, 2003.

24 T. D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs, LNCS*, volume 642, pages 426–441. Springer, 1978.

25 A. Pawlik, J. Kozik, T. Krawczyk, M. Lasoń, P. Micek, W. T. Trotter, and B. Walczak. Triangle-free intersection graphs of line segments with large chromatic number. *Journal of Combinatorial Theory Series B*, 105:6–10, 2014.

26 J. Petr, J. Portier, and L. Versteegen. A faster algorithm for cops and robbers. *arXiv*, 2021. `arXiv:2112.07449v2`.

27 A. Quilliot. *Thése d'Etat*. PhD thesis, Université de Paris VI, 1983.

28 A. Quilliot. A short note about pursuit games played on a graph with a given genus. *Journal of Combinatorial Theory Series B*, 38(1):89–92, 1985.

29 B. S. W. Schroeder. The copnumber of a graph is bounded by$\lfloor 3/2 * genus(g) \rfloor + 3$. In *Categorical perspectives. Trends in mathematics*, pages 243–263, 2001.

30 P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory Series B*, 58(1):22–33, 1993.

A Fully Active Cops and Robber on Planar Graphs

In this section, we show that for a planar graph G, $\mathsf{c_a}(G) \leq 4$. We rather consider the game where only the cops are forced to be active and $\mathcal{R}$ is flexible. In this section, we show that for a planar graph G, $\mathsf{c_A}(G) \leq 4$. It is easy to see that for a graph G, $\mathsf{c}(G) \leq \mathsf{c_A}(G)$. Hence, there exist a planar graph G such that $\mathsf{c_A}(G) \geq 3$. Here, we argue that four active cops have a winning strategy for any planar graph. First, we present a straightforward result.

▶ **Lemma 22.** *Let P be an isometric path of a graph G. Then, two active cops can guard P, after a finite number of cop moves.*

Proof. Let the two cops be denoted as *sheriff* and *deputy*. The two cops stay on adjacent vertices of P. The cops move such that the sheriff can guard P using the strategy to guard P in Cops and Robber setting using Proposition 1. At some point during the game, if in the classical game strategy, the sheriff has to stay at a vertex on the cop move, the two cops switch positions and also switch the role of sheriff and deputy. This way, the cop that currently is the sheriff guards P. ◀

Using Lemma 22 and the strategy of Aigner and Fromme [3], it is easy to see that $\mathsf{c_A}(G) \leq 6$. We use Lemma 9 and Lemma 22 to improve this bound in the following theorem.

▶ **Theorem 10.** *If G is a planar graph, then $\mathsf{c_A}(G) \leq 4$.*

Proof. We can use Lemma 9 and techniques similar to the techniques used for string graphs in Section 3 to show that four active cops have a winning strategy against a flexible robber. ◀

Since $\mathsf{c_a}(G) \leq \mathsf{c_A}(G)$, we have the following immediate corollary of Theorem 10.

▶ **Corollary 23.** *Let P be a planar graph. Then $\mathsf{c_a}(G) \leq 4$.*

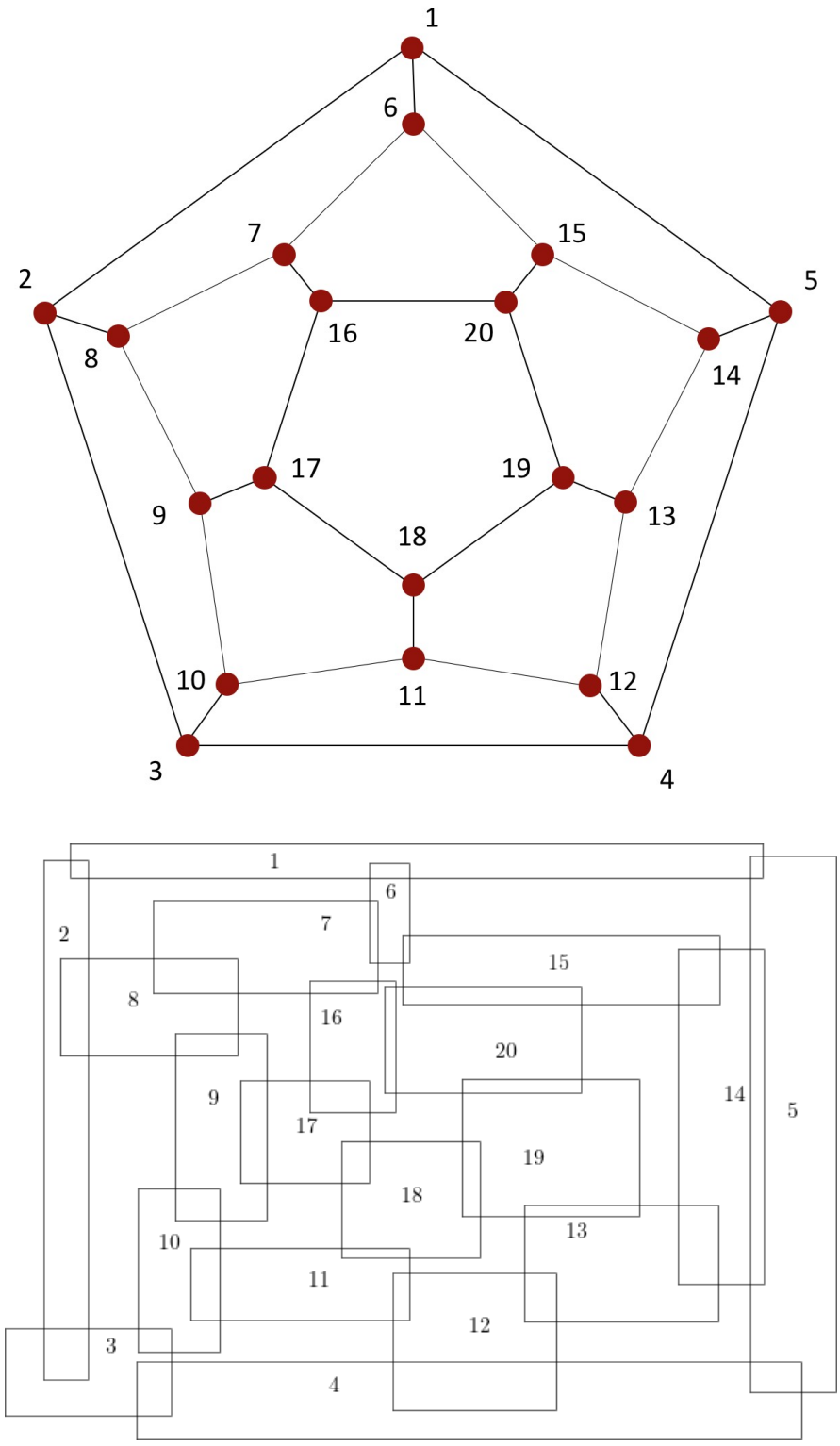


Figure 2 The dodecahedron and its boxicity 2 representation. Here each vertex i corresponds to rectangle i.

On the Parameterized Intractability of Determinant Maximization

Naoto Ohsaka
CyberAgent, Inc., Tokyo, Japan

Abstract

In the Determinant Maximization problem, given an $n \times n$ positive semi-definite matrix $\mathbf{A}$ in $\mathbb{Q}^{n\times n}$ and an integer k, we are required to find a $k \times k$ principal submatrix of $\mathbf{A}$ having the maximum determinant. This problem is known to be NP-hard and further proven to be W[1]-hard with respect to k by Koutis [26]; i.e., a $f(k)n^{\mathcal{O}(1)}$-time algorithm is unlikely to exist for any computable function f. However, there is still room to explore its parameterized complexity in the *restricted case*, in the hope of overcoming the general-case parameterized intractability. In this study, we rule out the fixed-parameter tractability of Determinant Maximization even if an input matrix is extremely sparse or low rank, or an approximate solution is acceptable. We first prove that Determinant Maximization is NP-hard and W[1]-hard even if an input matrix is an *arrowhead matrix*; i.e., the underlying graph formed by nonzero entries is a star, implying that the structural sparsity is not helpful. By contrast, we show that Determinant Maximization is solvable in polynomial time on *tridiagonal matrices*. Thereafter, we demonstrate the W[1]-hardness with respect to the *rank* r of an input matrix. Our result is stronger than Koutis' result in the sense that any $k \times k$ principal submatrix is singular whenever $k > r$. We finally give evidence that it is W[1]-hard to approximate Determinant Maximization parameterized by k within a factor of $2^{-c\sqrt{k}}$ for some universal constant $c > 0$. Our hardness result is conditional on the *Parameterized Inapproximability Hypothesis* posed by Lokshtanov, Ramanujan, Saurab, and Zehavi [30], which asserts that a gap version of Binary Constraint Satisfaction Problem is W[1]-hard. To complement this result, we develop an ε-additive approximation algorithm that runs in $\varepsilon^{-r^2} \cdot r^{\mathcal{O}(r^3)} \cdot n^{\mathcal{O}(1)}$ time for the rank r of an input matrix, provided that the diagonal entries are bounded.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Determinant maximization, Parameterized complexity, Approximability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.46

Related Version *Full Version*: https://arxiv.org/abs/2209.12519 [38]

1 Introduction

Background. We study the following Determinant Maximization problem: Given an $n \times n$ positive semi-definite matrix $\mathbf{A}$ in $\mathbb{Q}^{n\times n}$ and an integer k in $[n]$ denoting the solution size, find a $k \times k$ principal submatrix of $\mathbf{A}$ having the maximum determinant; namely, maximize $\det(\mathbf{A}_S)$ subject to $S \in \binom{[n]}{k}$. One motivating example for this problem is a *subset selection task*. Suppose we are given n items (e.g., images or products) associated with feature vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ and required to select a "diverse" set of k items among them. We can measure the diversity of a set S of k items using the principal minor $\det(\mathbf{A}_S)$ of the Gram matrix $\mathbf{A}$ defined by feature vectors such that $A_{i,j} \triangleq \langle \mathbf{v}_i, \mathbf{v}_j \rangle$ for all $i, j \in [n]$, resulting in Determinant Maximization. This formulation is justified by the fact that $\det(\mathbf{A}_S)$ is equal to the squared volume of the parallelepiped spanned by $\{\mathbf{v}_i : i \in S\}$; that is, a pair of vectors at a large angle is regarded as more diverse. In artificial intelligence and machine learning communities, Determinant Maximization is also known as MAP inference on a *determinantal point process* [6, 31], and has found many applications over the past decade, including tweet timeline generation [43], object detection [29], change-point

detection [44], document summarization [8, 27], YouTube video recommendation [42], and active learning [5]. See the survey of Kulesza and Taskar [28] for further details. Though Determinant Maximization is known to be NP-hard to solve exactly [25], we can achieve an e^{-k}-factor approximation in polynomial time [36], which is nearly optimal because a 2^{-ck}-factor approximation for some constant $c > 0$ is impossible unless P = NP [11, 14, 26].

Having known a nearly tight hardness-of-approximation result in the polynomial-time regime, we resort to *parameterized algorithms* [13, 16, 19]. We say that a problem is *fixed-parameter tractable* (FPT) with respect to a parameter $k \in \mathbb{N}$ if it can be solved in $f(k)|\mathcal{I}|^{\mathcal{O}(1)}$ time for some computable function f and instance size $|\mathcal{I}|$. One very natural parameter is the *solution size* k, which is expected to be small in practice. By enumerating all $k \times k$ principal submatrices, we can solve Determinant Maximization in $n^{k+\mathcal{O}(1)}$ time; i.e., it belongs to the class XP. Because FPT $\subsetneq$ XP [16], it is even more desirable if an FPT algorithm exists. Unfortunately, Koutis [26] has already proven that Determinant Maximization is W[1]-hard with respect to k. Therefore, under the widely-believed assumption that FPT $\neq$ W[1], an FPT algorithm for Determinant Maximization does not exist.

However, there is still room to explore the parameterized complexity of Determinant Maximization in the *restricted case*, in the hope of circumventing the general-case parameterized intractability. Here, we describe three possible scenarios. One can first assume an input matrix $\mathbf{A}$ to be *sparse*. Of particular interest is the structural sparsity of the *symmetrized graph* of $\mathbf{A}$ [9, 12] defined as the underlying graph formed by nonzero entries of $\mathbf{A}$, encouraged by numerous FPT algorithms for NP-hard graph-theoretic problems parameterized by the treewidth [13, 20]. For example, in change-point detection applications, Zhang and Ou [44] observed a small-bandwidth matrix and developed an efficient heuristic for Determinant Maximization. In addition, one may adopt a strong parameter. The *rank* of an input matrix $\mathbf{A}$ is such a natural candidate. We often assume that $\mathbf{A}$ is low-rank in applications; for instance, the feature vectors $\mathbf{v}_i$ are inherently low-dimensional [7] or the largest possible subset is significantly smaller than the ground set size n. Since any $k \times k$ principal submatrix of $\mathbf{A}$ is singular whenever $k > \operatorname{rank}(\mathbf{A})$, we can ensure that $k \leqslant \operatorname{rank}(\mathbf{A})$; namely, parameterization by $\operatorname{rank}(\mathbf{A})$ is considered stronger than that by k. Intriguingly, the partition function of product determinantal point processes is FPT with respect to rank while #P-hard in general [40]. The last possibility to be considered is *FPT-approximability*. Albeit W[1]-hardness of Determinant Maximization with parameter k, it could be possible to obtain an approximate solution in FPT time. It has been demonstrated that several W[1]-hard problems can be approximated in FPT time, such as Partial Vertex Cover and Minimum k-Median [22] (refer to the survey of Marx [34] and Feldmann, Karthik, Lee, and Manurangsi [17]). One may thus envision the existence of a $1/\rho(k)$-factor FPT-approximation algorithm for Determinant Maximization for a *small* function ρ. Alas, we refute the above possibilities under a plausible assumption in parameterized complexity.

Our Results. We improve the W[1]-hardness of Determinant Maximization due to Koutis [26] by showing that it is still W[1]-hard even if an input matrix is extremely sparse or low rank, or an approximate solution is acceptable, along with some tractable cases.

We first prove that Determinant Maximization is NP-hard and W[1]-hard with respect to k even if the input matrix $\mathbf{A}$ is an *arrowhead matrix* (Theorem 3.1). An arrowhead matrix is a square matrix that can include nonzero entries only in the first row, the first column, or the diagonal; i.e., its symmetrized graph is a star. Our hardness result implies that the "structural sparsity" of input matrices is not helpful; in particular, it follows from Theorem 3.1 that this problem is NP-hard even if the treewidth, pathwidth, and vertex cover number of the

symmetrized graph are all 1. The proof is based on a parameterized reduction from k-Sum, which is a parameterized version of Subset Sum known to be W[1]-complete [1, 15], and involves a structural feature of the determinant of arrowhead matrices. On the other hand, we show that Determinant Maximization is solvable in polynomial time on *tridiagonal* matrices (Observation 3.9), whose symmetrized graph is a path graph.

Thereafter, we demonstrate that Determinant Maximization is W[1]-hard when parameterized by the *rank* of an input matrix (Corollary 4.3). In fact, we obtain the stronger result that it is W[1]-hard to determine whether an input set of n d-dimensional vectors includes k pairwise orthogonal vectors when parameterized by d (Theorem 4.2). Unlike the proof of Theorem 3.1, we are allowed to construct only a $f(k)$-dimensional vector in a parameterized reduction. Note that a straightforward parameterized reduction from a canonical W[1]-complete k-Clique problem fails (see Remark 4.4). Therefore, we reduce from a different W[1]-complete problem called Grid Tiling due to Marx [33, 35]. In Grid Tiling, we are given k^2 nonempty sets of integer pairs arranged in a $k \times k$ grid, and the task is to select k^2 integer pairs such that the vertical and horizontal neighbors agree respectively in the first and second coordinates (see Problem 4.5 for the precise definition). Grid Tiling is favorable for our purpose because the constraint consists of simple equalities, and each cell is adjacent to (at most) four cells. To express the consistency between adjacent cells using only a $f(k)$-dimensional vector, we exploit Pythagorean triples. It is essential in Theorem 4.2 that the input vectors can include *both* positive and negative entries in a sense that we can find k d-dimensional nonnegative vectors that are pairwise orthogonal in FPT time with respect to d (Observation 4.7).

Our final contribution is to give evidence that it is W[1]-hard to determine whether the optimal value of Determinant Maximization is equal to 1 or at most $2^{-c\sqrt{k}}$ for some universal constant $c > 0$; namely, Determinant Maximization is FPT-inapproximable within a factor of $2^{-c\sqrt{k}}$ (Theorem 5.1). Our result is conditional on the *Parameterized Inapproximability Hypothesis* (PIH), which is a conjecture posed by Lokshtanov, Ramanujan, Saurab, and Zehavi [30] asserting that a gap version of Binary Constraint Satisfaction Problem is W[1]-hard when parameterized by the number of variables. PIH can be thought of as a parameterized analogue of the PCP theorem [2,3]; e.g., Lokshtanov et al. [30] show that assuming PIH and FPT $\neq$ W[1], Directed Odd Cycle Transversal does not admit a $(1-\varepsilon)$-factor FPT-approximation algorithm for some $\varepsilon > 0$. The proof of Theorem 5.1 involves FPT-inapproximability of Grid Tiling under PIH, which is reminiscent of Marx's work [33] and might be of some independent interest. Because we cannot achieve an exponential gap by simply reusing the parameterized reduction from Grid Tiling of the second hardness result (as inferred from Observation 5.11 below), we apply a gadget invented by Çivril and Magdon-Ismail [11] to construct an $\mathcal{O}(k^2n^2)$-dimensional vector for each integer pair of a Grid Tiling instance. We further show that the same kind of hardness result does *not* hold when parameterized by the rank r of an input matrix. Specifically, we develop an ε-additive approximation algorithm that runs in $\varepsilon^{-r^2} \cdot r^{\mathcal{O}(r^3)} \cdot n^{\mathcal{O}(1)}$ time for any $\varepsilon > 0$, provided that the diagonal entries are bounded (Observation 5.11).

Owing to space limitations, proofs marked with $*$ are omitted and can be found in a full version of this paper [38].

More Related Work. Determinant Maximization is not only applied in artificial intelligence and machine learning but also in computational geometry [21] and discrepancy theory; refer to Nikolov [36] and references therein. On the negative side, Ko, Lee, and

Queyranne [25] prove that Determinant Maximization is NP-hard, and Koutis [26] proves that it is further W[1]-hard. NP-hardness of approximating Determinant Maximization has been investigated in [11, 14, 26, 39]. On the algorithmic side, a greedy algorithm achieves an approximation factor of $1/k!$ [10]. Subsequently, Nikolov [36] gives an e^{-k}-factor approximation algorithm; partition constraints [37] and matroid constraints [32] are also studied. Several #P-hard computation problems over matrices including permanents [9, 12], hyperdeterminants [9], and partition functions of product determinantal point processes [40] are efficiently computable if the treewidth of the symmetrized graph or the matrix rank is bounded.

2 Preliminaries

Notations and Definitions. For two integers $m, n \in \mathbb{N}$ with $m \leqslant n$, let $[n] \triangleq \{1, 2, \ldots, n\}$ and $[m \mathinner{..} n] \triangleq \{m, m+1, \ldots, n-1, n\}$. For a finite set S and an integer k, we write $\binom{S}{k}$ for the family of all size-k subsets of S. For a statement P, $[\![P]\!]$ is 1 if P is true, and 0 otherwise. The base of logarithms is 2. The Euclidean norm is denoted $\|\cdot\|$; i.e., $\|\mathbf{v}\| \triangleq \sqrt{\sum_{i\in[d]}(v(i))^2}$ for a vector $\mathbf{v} \in \mathbb{R}^d$. We use $\langle\cdot,\cdot\rangle$ for the standard inner product; i.e., $\langle \mathbf{v}, \mathbf{w}\rangle \triangleq \sum_{i\in[d]} v(i) \cdot w(i)$ for two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$. For an $n \times n$ matrix $\mathbf{A}$ and an index set $S \subseteq [n]$, we use $\mathbf{A}_S$ to denote the principal submatrix of $\mathbf{A}$ whose rows and columns are indexed by S. For an $m \times n$ matrix $\mathbf{A}$, the *spectral norm* $\|\mathbf{A}\|_2$ is defined as the square root of the maximum eigenvalue of $\mathbf{A}^\top\mathbf{A}$ and the *max norm* is defined as $\|\mathbf{A}\|_{\max} \triangleq \max_{i,j} |A_{i,j}|$. It is well-known that $\|\mathbf{A}\|_{\max} \leqslant \|\mathbf{A}\|_2 \leqslant \sqrt{mn} \cdot \|\mathbf{A}\|_{\max}$. The *symmetrized graph* [9, 12] of an $n \times n$ matrix $\mathbf{A}$ is defined as an undirected graph G that has each integer of $[n]$ as a vertex and an edge $(i, j) \in \binom{[n]}{2}$ if $A_{i,j} \neq 0$ or $A_{j,i} \neq 0$; i.e., $G = ([n], \{(i, j) \colon A_{i,j} \neq 0\})$. For a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, its *determinant* is defined as follows:

$$\det(\mathbf{A}) \triangleq \sum_{\sigma\in\mathfrak{S}_n} \operatorname{sgn}(\sigma) \prod_{i\in[n]} A_{i,\sigma(i)},$$

where $\mathfrak{S}_n$ denotes the symmetric group on $[n]$, and $\operatorname{sgn}(\sigma)$ denotes the sign of a permutation σ. We define $\det(\mathbf{A}_\emptyset) \triangleq 1$. For a collection $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ of n vectors in $\mathbb{R}^d$, the *volume* of the parallelepiped spanned by $\mathbf{V}$ is defined as follows:

$$\operatorname{vol}(\mathbf{V}) \triangleq \|\mathbf{v}_1\| \cdot \prod_{2\leqslant i\leqslant n} \mathrm{d}(\mathbf{v}_i, \{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}\}). \tag{1}$$

Here, $\mathrm{d}(\mathbf{v}, \mathbf{P})$ denotes the distance of $\mathbf{v}$ to the subspace spanned by $\mathbf{P}$; i.e., $\mathrm{d}(\mathbf{v}, \mathbf{P}) \triangleq \|\mathbf{v} - \operatorname{proj}_{\mathbf{P}}(\mathbf{v})\|$, where $\operatorname{proj}_{\mathbf{P}}(\cdot)$ is an operator of orthogonal projection onto the subspace spanned by $\mathbf{P}$. We define $\operatorname{vol}(\emptyset) \triangleq 1$ for the sake of consistency to the determinant of an empty matrix (i.e., $\det([\,]) = 1 = \operatorname{vol}^2(\emptyset)$). If $\mathbf{A}$ is the Gram matrix defined as $A_{i,j} \triangleq \langle \mathbf{v}_i, \mathbf{v}_j\rangle$ for all $i, j \in [n]$, we have a simple relation between the principal minor and the volume of the parallelepiped that

$$\det(\mathbf{A}_S) = \operatorname{vol}^2(\{\mathbf{v}_i \colon i \in S\}) \tag{2}$$

for every $S \subseteq [n]$. We formally define the Determinant Maximization problem as follows.[1]

[1] Note that if we consider the decision version of Determinant Maximization, we are additionally given a target number τ and are required to decide if $\operatorname{maxdet}(\mathbf{A}, k) \geqslant \tau$.

▶ **Problem 2.1.** *Given a positive semi-definite matrix* $\mathbf{A}$ *in* $\mathbb{Q}^{n\times n}$ *and a positive integer* $k \in [n]$, Determinant Maximization *asks to find a set* $S \in \binom{[n]}{k}$ *such that the determinant* $\det(\mathbf{A}_S)$ *of a* $k \times k$ *principal submatrix is maximized. The optimal value is denoted* $\mathrm{maxdet}(\mathbf{A}, k) \triangleq \max_{S\in\binom{[n]}{k}} \det(\mathbf{A}_S)$.

Due to the equivalence between squared volume and determinant in Eq. (2), Determinant Maximization is equivalent to the following problem of volume maximization: Given a collection of n vectors in $\mathbb{Q}^d$ and a positive integer $k \in [n]$, we are required to find k vectors such that the volume of the parallelepiped spanned by them is maximized. We shall use the problem definition based on the determinant and the volume interchangeably.

Parameterized Complexity. Given a *parameterized problem* Π consisting of a pair $\langle \mathcal{I}, k\rangle$ of instance $\mathcal{I}$ and parameter $k \in \mathbb{N}$, we say that Π is *fixed-parameter tractable* (FPT) with respect to k if it is solvable in $f(k)|\mathcal{I}|^{\mathcal{O}(1)}$ time for some computable function f, and *slice-wise polynomial* (XP) if it is solvable in $|\mathcal{I}|^{f(k)}$ time; it holds that $\mathsf{FPT} \subsetneq \mathsf{XP}$ [16]. The value of parameter k may be independent of the instance size $|\mathcal{I}|$ and may be given by some computable function $k = k(\mathcal{I})$ on instance $\mathcal{I}$ (e.g., the rank of an input matrix). Our objective is to prove that a problem (i.e., Determinant Maximization) is unlikely to admit an FPT algorithm under plausible assumptions in parameterized complexity. The central notion for this purpose is a parameterized reduction, which is used to demonstrate that a problem of interest is hard for a particular class of parameterized problems that is believed to be a superclass of FPT. We say that a parameterized problem Π_1 is *parameterized reducible* to another parameterized problem Π_2 if (i) an instance $\mathcal{I}_1$ with parameter k_1 for Π_1 can be transformed into an instance $\mathcal{I}_2$ with parameter k_2 for Π_2 in FPT time and (ii) the value of k_2 only depends on the value of k_1. Note that a parameterized reduction may not be a polynomial-time reduction and vice versa. W[1] is a class of parameterized problems that are parameterized reducible to k-Clique, and it is known that $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{XP}$. This class is often regarded as a parameterized counterpart to NP of classical complexity; in particular, the conjecture $\mathsf{FPT} \neq \mathsf{W}[1]$ is a widely-believed assumption in parameterized complexity [16, 19]. Thus, the existence of a parameterized reduction from a W[1]-complete problem to a problem Π is a strong evidence that Π is not in FPT. In Determinant Maximization, a simple brute-force search algorithm that examines all $\binom{[n]}{k}$ subsets of size k runs in $n^{k+\mathcal{O}(1)}$ time; hence, this problem belongs to XP. On the other hand, it is proven to be W[1]-hard [26].

3 W[1]-hardness and NP-hardness on Arrowhead Matrices

We first prove the W[1]-hardness with respect to k and NP-hardness on arrowhead matrices. A square matrix $\mathbf{A}$ in $\mathbb{R}^{[0..n]\times[0..n]}$ is an *arrowhead matrix* if $A_{i,j} = 0$ for all $i, j \in [n]$ with $i \neq j$. In the language of graph theory, $\mathbf{A}$ is arrowhead if its symmetrized graph is a star $K_{1,n}$.

▶ **Theorem 3.1.** Determinant Maximization *on arrowhead matrices is* NP*-hard and* W[1]*-hard when parameterized by* k.

The proof of Theorem 3.1 requires a reduction from k-Sum, a natural parameterized version of the NP-complete Subset Sum problem, whose membership of W[1] and W[1]-hardness was proven by Abboud, Lewi, and Williams [1] and Downey and Fellows [15], respectively.

▶ **Problem 3.2** (k-Sum due to Abboud, Lewi, and Williams [1])**.** *Given* n *integers* $x_1, \ldots, x_n \in [0 \mathinner{..} n^{2k}]$, *a target integer* $t \in [0 \mathinner{..} n^{2k}]$, *and a positive integer* $k \in [n]$, *we are required to decide if there exists a size-*k *set* $S \in \binom{[n]}{k}$ *such that* $\sum_{i\in S} x_i = t$.

Here, we introduce a slightly-modified version of k-Sum such that the input numbers are *rational* and their sum is *normalized* to 1, without affecting its computational complexity.

▶ **Problem 3.3** (k-Sum modified from [1]). *Given n rational numbers $x_1, \dots, x_n$ in $(0,1) \cap \mathbb{Q}_+$, a target rational number t in $(0,1) \cap \mathbb{Q}_+$, and a positive integer $k \in [n]$ such that x_i's are integer multiples of some rational number at least $\frac{1}{n^{2k+1}}$ and $\sum_{i \in [n]} x_i = 1$, k-Sum asks to decide if there exists a set $S \in \binom{[n]}{k}$ such that $\sum_{i \in S} x_i = t$.*

Hereafter, for any set $S \subseteq [0 \,..\, n]$ including 0, we denote $S_{-0} \triangleq S \setminus \{0\}$.

3.1 Reduction from k-Sum and Proof of Theorem 3.1

In this subsection, we give a parameterized, polynomial-time reduction from k-Sum. We first use an explicit formula of the determinant of arrowhead matrices.

▶ **Lemma 3.4** (∗). *Let $\mathbf{A}$ be an arrowhead matrix in $\mathbb{R}^{[0..n] \times [0..n]}$ such that $A_{i,i} \neq 0$ for all $i \in [n]$. Then, for any set $S \subseteq [0 \,..\, n]$, it holds that*

$$\det(\mathbf{A}_S) = \begin{cases} \displaystyle\prod_{i \in S_{-0}} A_{i,i} \cdot \left(A_{0,0} - \sum_{i \in S_{-0}} \frac{A_{0,i} \cdot A_{i,0}}{A_{i,i}} \right) & \text{if } 0 \in S, \\ \displaystyle\prod_{i \in S} A_{i,i} & \text{if } 0 \notin S. \end{cases}$$

Lemma 3.4 shows us a way to express the *product* of $\prod_{i \in S_{-0}} x_i$ and $1 - C \cdot \sum_{i \in S_{-0}} x_i$ for some constant C, which is a key step in proving Theorem 3.1. Specifically, given n rational numbers $x_1, \dots, x_n$ and a target rational number t as a k-Sum instance, we construct $n+1$ $2n$-dimensional vectors $\mathbf{v}_0, \dots, \mathbf{v}_n$ in $\mathbb{R}^{2n}_+$, each entry of which is defined as follows:

$$v_0(j) = \begin{cases} \gamma \cdot \sqrt{x_j} & \text{if } j \leqslant n, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and } v_i(j) = \begin{cases} \sqrt{\alpha \cdot \mathrm{e}^{x_i}} & \text{if } j = i, \\ \sqrt{\beta \cdot \mathrm{e}^{x_i}} & \text{if } j = i + n, \\ 0 & \text{otherwise,} \end{cases} \text{ for all } i \in [n], \tag{3}$$

where α, β, and γ are parameters, whose values are positive and will be determined later. We calculate the principal minor of the Gram matrix defined by $\mathbf{v}_0, \dots, \mathbf{v}_n$ as follows.

▶ **Lemma 3.5** (∗). *Let $\mathbf{A}$ be the Gram matrix defined by $n+1$ vectors $\mathbf{v}_0, \dots, \mathbf{v}_n$ that are constructed from an instance of k-Sum by Eq. (3). Then, $\mathbf{A}$ is an arrowhead matrix, and for any set $S \subseteq [0 \,..\, n]$, it holds that*

$$\det(\mathbf{A}_S) = \begin{cases} (\alpha+\beta)^{|S|-1} \cdot \gamma^2 \cdot \exp\left(\displaystyle\sum_{i \in S_{-0}} x_i \right) \cdot \left(1 - \frac{\alpha}{\alpha+\beta} \sum_{i \in S_{-0}} x_i \right) & \text{if } 0 \in S, \\ (\alpha+\beta)^{|S|} \cdot \exp\left(\displaystyle\sum_{i \in S} x_i \right) & \text{if } 0 \notin S. \end{cases}$$

Moreover, if we regard the principal minor $\det(\mathbf{A}_S)$ in the case of $0 \in S$ as a function in $X \triangleq \sum_{i \in S_{-0}} x_i$, it is maximized when $X = \frac{\beta}{\alpha}$.

We now determine the values of α, β, and γ. Since Lemma 3.5 demonstrates that the principal minor for S including 0 is maximized when $\sum_{i \in S_{-0}} x_i = \frac{\beta}{\alpha}$, we fix $\alpha \triangleq 1$ and $\beta \triangleq t$. We define $\delta \triangleq \frac{1}{n^{2k+1}}$, denoting a lower bound on the *minimum possible absolute*

difference between any sum of x_i's; i.e., $|\sum_{i\in S} x_i - \sum_{i\in T} x_i| \geqslant \delta$ for any $S, T \subseteq [n]$ whenever $\sum_{i\in S} x_i \neq \sum_{i\in T} x_i$. For the correctness of the value of δ, refer to the definition of Problem 3.3. We finally fix the value of γ as $\gamma \triangleq 5$, so that

$$(1+t)^2 \cdot \mathrm{e}^{1-t} \cdot \frac{1}{\mathrm{e}^{-\delta}\cdot(1+\delta)} \leqslant 2^2 \cdot \mathrm{e} \cdot \frac{1}{\mathrm{e}^{-\frac{1}{2}}\cdot 1} < 25 = \gamma^2. \tag{4}$$

The above inequality ensures that $\det(\mathbf{A}_S)$ is "sufficiently" small whenever $0 \notin S$, as validated in the following lemma.

▶ **Lemma 3.6** (∗). *Let* $\mathbf{A}$ *be the Gram matrix defined by* $n+1$ *vectors constructed according to Eq.* (3)*, where* $\alpha = 1$*,* $\beta = t$*, and* $\gamma = 5$*. Define* $\mathsf{OPT} \triangleq (1+t)^{k-1}\cdot\gamma^2\cdot\mathrm{e}^t$*. Then, for any set* $S \in \binom{[0..n]}{k+1}$*,*

$$\det(\mathbf{A}_S) \text{ is } \begin{cases} \text{equal to } \mathsf{OPT} & \text{if } 0 \in S \text{ and } \sum_{i\in S_{-0}} x_i = t, \\ \text{at most } \mathrm{e}^{-\delta}(1+\delta)\cdot\mathsf{OPT} & \text{otherwise.} \end{cases}$$

In particular, $\operatorname{maxdet}(\mathbf{A}, k+1)$ *is* OPT *if* k-Sum *has a solution, and is at most* $\mathrm{e}^{-\delta}(1+\delta)\cdot\mathsf{OPT} < \mathsf{OPT}$ *otherwise.*

We complete our reduction by approximating the Gram matrix $\mathbf{A}$ of $n+1$ vectors defined in Eq. (3) by a *rational* matrix $\mathbf{B}$ whose maximum determinant maintains sufficient information to solve k-Sum.

▶ **Lemma 3.7** (∗). *Let* $\mathbf{B}$ *be the Gram matrix in* $\mathbb{Q}^{(n+1)\times(n+1)}$ *defined by* $n+1$ *vectors* $\mathbf{w}_0, \dots, \mathbf{w}_n$ *in* $\mathbb{Q}^{2n}$*, each entry of which is a* $(1\pm\varepsilon)$*-factor approximation to the corresponding entry of* $n+1$ *vectors* $\mathbf{v}_0, \dots, \mathbf{v}_n$ *defined by Eq.* (3)*, where* $\varepsilon = 2^{-\mathcal{O}(k\log(nk))}$*. Then,*

$$\operatorname{maxdet}(\mathbf{B}, k+1) \text{ is } \begin{cases} \text{at least } \left(\frac{2}{3} + \frac{1}{3}\mathrm{e}^{-\delta}(1+\delta)\right)\cdot\mathsf{OPT} & \text{if } k\text{-Sum has a solution,} \\ \text{at most } \left(\frac{1}{3} + \frac{2}{3}\mathrm{e}^{-\delta}(1+\delta)\right)\cdot\mathsf{OPT} & \text{otherwise.} \end{cases}$$

Moreover, we can calculate $\mathbf{B}$ *in polynomial time.*

The crux of its proof is to approximate $\mathbf{A}$ within a factor of $\varepsilon = 2^{-\mathcal{O}(k\log(nk))}$. To this end, we use the following lemma.

▶ **Lemma 3.8** (cf. [4, page 107]). *For two complex-valued* $n \times n$ *matrices* $\mathbf{A}$ *and* $\mathbf{B}$*, the absolute difference in the determinant of* $\mathbf{A}$ *and* $\mathbf{B}$ *is bounded from above by*

$$|\det(\mathbf{A}) - \det(\mathbf{B})| \leqslant n\cdot\max\{\|\mathbf{A}\|_2, \|\mathbf{B}\|_2\}^{n-1}\cdot\|\mathbf{A}-\mathbf{B}\|_2.$$

What remains to be done is to prove Theorem 3.1 using Lemma 3.7.

Proof of Theorem 3.1. Our parameterized reduction is as follows. Given n rational numbers $x_1, \dots, x_n \in (0,1)\cap\mathbb{Q}$, a target rational number $t \in (0,1)\cap\mathbb{Q}$, and a positive integer $k \in [n]$ as an instance of k-Sum, we construct $n+1$ rational vectors $\mathbf{w}_0, \dots, \mathbf{w}_n$ in $\mathbb{Q}_+^{2n}$, each of which is an entry-wise $(1\pm\varepsilon)$-factor approximation to $\mathbf{v}_0, \dots, \mathbf{v}_n$ defined by Eq. (3), where $\varepsilon = 2^{-\mathcal{O}(k\log(nk))}$. This construction requires polynomial time owing to Lemma 3.7. Thereafter, we compute the Gram matrix $\mathbf{B}$ in $\mathbb{Q}^{(n+1)\times(n+1)}$ defined by $\mathbf{w}_0, \dots, \mathbf{w}_n$. Consider Determinant Maximization defined by $(\mathbf{B}, k+1)$ with parameter $k+1$. According to Lemma 3.7, the maximum principal minor $\operatorname{maxdet}(\mathbf{B}, k+1)$ is at least $(\frac{2}{3}+\frac{1}{3}\mathrm{e}^{-\delta}(1+\delta))\cdot\mathsf{OPT}$ if and only if k-Sum has a solution. Moreover, if this is the case, the optimal solution S^* for Determinant Maximization satisfies that $\sum_{i\in S^*_{-0}} x_i = t$. The above discussion ensures the correctness of the parameterized reduction from k-Sum to Determinant Maximization, finishing the proof. ◀

3.2 Polynomial-time Algorithm for Tridiagonal Matrices

Here, we demonstrate that Determinant Maximization is polynomial-time solvable on tridiagonal matrices. Recall that a *tridiagonal matrix* is a square matrix $\mathbf{A}$ such that $A_{i,j} = 0$ whenever $|i - j| \geqslant 2$; i.e., its symmetrized graph is a path graph (and thus a linear forest). Our polynomial-time algorithm is based on dynamic programming and uses the observation that the removal of any pair of row and column from a tridiagonal matrix renders it block diagonal.

▶ **Observation 3.9** (∗). Determinant Maximization *on tridiagonal matrices can be solved in polynomial time.*

4 W[1]-hardness With Respect to Rank

We then prove the W[1]-hardness of Determinant Maximization when parameterized by the *rank* of an input matrix. In fact, we obtain the stronger hardness result on the problem of finding a set of pairwise orthogonal rational vectors, which is formally stated below.

▶ **Problem 4.1.** *Given n d-dimensional vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ in $\mathbb{Q}^d$ and a positive integer $k \in [n]$, we are required to decide if there exists a set of k vectors that is pairwise orthogonal, i.e., a set $S \in \binom{[n]}{k}$ such that $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = 0$ for all $i \neq j \in S$.*

▶ **Theorem 4.2.** *Problem 4.1 is* W[1]*-hard when parameterized by the dimension d of the input vectors. Moreover, the same hardness result holds even if every vector has the same Euclidean norm.*

The following is immediate from Theorem 4.2.

▶ **Corollary 4.3** (∗). Determinant Maximization *is* W[1]*-hard when parameterized by the rank of an input matrix.*

▶ Remark 4.4. We briefly describe a failed attempt to proving Theorem 4.2 using a straightforward reduction from k-Clique. Let $G = (V, E)$ be a graph on n vertices and k a parameter denoting the solution size. Consider constructing a $f(k)$-dimensional vector $\mathbf{v}_i$ for each vertex $i \in V$ such that $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = 0$ if and only if $(i, j) \in E$, which ensures that k-Clique has a solution if and only if Problem 4.1 has a solution. When G consists of a clique of size $n - 1$ and an isolated vertex o, this seems impossible: On one hand, every vector should be a nonzero vector because $\mathbf{v}_o$ and $\mathbf{v}_i$ for $i \neq o$ are nonorthogonal; on the other hand, for $n - 1$ vectors in $\{\mathbf{v}_i : i \neq o\}$ to be pairwise orthogonal, they must be (at least) $(n - 1)$-dimensional.

The key tool to bypass this difficulty is Grid Tiling introduced in the next subsection.

4.1 Grid Tiling and Pythagorean Triples

We first define Grid Tiling due to Marx [33].

▶ **Problem 4.5** (Grid Tiling due to Marx [33]). *For two integers n and k, given a collection $\mathcal{S}$ of k^2 nonempty sets $S_{i,j} \subseteq [n]^2$ called* cells *for each $i, j \in [k]$,* Grid Tiling *asks to find an assignment $\sigma : [k]^2 \to [n]^2$ with $\sigma(i, j) \in S_{i,j}$ such that*

1. *vertical neighbors agree in the first coordinate; i.e., if $\sigma(i, j) = (x, y)$ and $\sigma((i + 1) \bmod k, j) = (x', y')$, then $x = x'$, and*
2. *horizontal neighbors agree in the second coordinate; i.e., if $\sigma(i, j) = (x, y)$ and $\sigma(j, (i + 1) \bmod k) = (x', y')$, then $y = y'$,*

where we define $(k + 1) \bmod k \triangleq 1$, and hereafter omit the symbol mod *for modulo operator.*

Grid Tiling parameterized by k is proven to be W[1]-hard by Marx [33, 35]. We say that two cells (i_1, j_1) and (i_2, j_2) are *adjacent* if the Manhattan distance between them is 1. Let $\mathcal{I}$ be the set of all pairs of two adjacent cells; i.e.,

$$\mathcal{I} \triangleq \Big\{(i_1, j_1, i_2, j_2) \in [n]^4 \colon |i_1 - i_2| + |j_1 - j_2| = 1\Big\}. \tag{5}$$

Note that $|\mathcal{I}| = 2k^2$. Grid Tiling has the two useful properties that (i) the constraint to be satisfied is the equality on the first and second coordinates, which is pretty simple, and (ii) there are only k^2 cells and each cell is adjacent to (at most) *four* cells. By contrast, as there are $\mathcal{O}(n^2)$ candidates for the assignment of integer pairs, we need to represent the *consistency* between adjacent cells using only $f(k)$-dimensional vectors. For this purpose, we exploit rational points on the unit circle; i.e., Pythagorean triples. A *Pythagorean triple* is a triple of three positive integers (a, b, c) such that $a^2 + b^2 = c^2$; e.g., $(a, b, c) = (3, 4, 5)$. It is further said to be *primitive* if (a, b, c) are coprime; i.e., $\gcd(a, b) = \gcd(b, c) = \gcd(c, a) = 1$. We assume for a while that we have n primitive Pythagorean triples, denoted $(a_1, b_1, c_1), \dots, (a_n, b_n, c_n)$.

4.2 Reduction from Grid Tiling and Proof of Theorem 4.2

We are now ready to describe a parameterized reduction from Grid Tiling to Problem 4.1. Given an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of Grid Tiling, we define a rational vector for each $(x, y) \in S_{i,j}$, whose dimension is bounded by some function in k. Each vector consists of $|\mathcal{I}| = 2k^2$ blocks (indexed by an element of $\mathcal{I}$), each of which is two dimensional and is either a rational point on the unit circle or the origin O. Hence, each vector is of dimension $2|\mathcal{I}| = 4k^2$. Let $\mathbf{v}_{x,y}^{(i,j)}$ denote the vector for an element $(x, y) \in S_{i,j}$ of cell $(i, j) \in [k]^2$, let $\mathbf{v}_{x,y}^{(i,j)}(i_1, j_1, i_2, j_2)$ denote the block of $\mathbf{v}_{x,y}^{(i,j)}$ corresponding to each pair of adjacent cells $(i_1, j_1, i_2, j_2) \in \mathcal{I}$. Each bock is defined as follows:

$$\mathbf{v}_{x,y}^{(i,j)}(e) \triangleq \begin{cases} \left[-b_x/c_x, a_x/c_x\right] & \text{if } e = (i-1, j, i, j), \\ \left[a_x/c_x, b_x/c_x\right] & \text{if } e = (i, j, i+1, j), \\ \left[-b_y/c_y, a_y/c_y\right] & \text{if } e = (i, j-1, i, j), \\ \left[a_y/c_y, b_y/c_y\right] & \text{if } e = (i, j, i, j+1), \\ [0, 0] & \text{otherwise.} \end{cases} \tag{6}$$

Because each vector contains exactly four points on the unit circle, its squared norm is equal to 4. We denote by $\mathbf{V}^{(i,j)}$ the set of vectors corresponding to the elements of $S_{i,j}$; i.e., $\mathbf{V}^{(i,j)} \triangleq \{\mathbf{v}_{x,y}^{(i,j)} \colon (x, y) \in S_{i,j}\}$. We now define an instance $(\mathbf{V}, K)$ of Problem 4.1 as $\mathbf{V} \triangleq \bigcup_{i,j\in[k]} \mathbf{V}^{(i,j)}$ and $K \triangleq k^2$. Note that $\mathbf{V}$ consists of $N \triangleq \sum_{i,j\in[k]} |S_{i,j}|$ vectors. We prove that the existence of a set of pairwise orthogonal k^2 vectors yields the answer of Grid Tiling. The key property of the above construction is that $[-b_x/c_x, a_x/c_x]$ and $[a_{x'}/c_{x'}, b_{x'}/c_{x'}]$ are orthogonal if and only if $x = x'$.

▶ **Lemma 4.6** (∗). *Let $\mathbf{V}$ be the set of vectors constructed from an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of Grid Tiling according to Eq. (6). Then, Grid Tiling has a solution if and only if Problem 4.1 has a solution.*

Proof of Theorem 4.2. Our parameterized reduction is as follows. Given an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of Grid Tiling, we first generate n primitive Pythagorean triples (a_1, b_1, c_1), $\dots, (a_n, b_n, c_n)$. This can be done in $\mathcal{O}(n^2)$ time, e.g., using Fibonacci's method [18]. We then construct a set $\mathbf{V}$ of N $4k^2$-dimensional rational vectors from $\mathcal{S}$ according to Eq. (6)

in polynomial time, where $N \triangleq \sum_{i,j\in[k]} |S_{i,j}|$. According to Lemma 4.6, $\mathcal{S}$ has a solution of Grid Tiling if and only if there exists a set of k^2 pairwise orthogonal vectors in $\mathbf{V}$. Since Grid Tiling is W[1]-hard with respect to k, Problem 4.1 is also W[1]-hard when parameterized by dimension $d(= 4k^2)$. Note that every vector is of squared norm 4, completing the proof. ◀

4.3 Problem 4.1 on Nonnegative Vectors is FPT

We note that Problem 4.1 is FPT with respect to the dimension if the input vectors are *nonnegative*. Briefly speaking, Problem 4.1 on nonnegative vectors is equivalent to Set Packing parameterized by the size of the universe, which is easily shown to be FPT.

▶ **Observation 4.7** (∗). *Problem 4.1 is FPT with respect to the dimension if every input vector is entry-wise nonnegative.*

5 W[1]-hardness of Approximation

Our final result is FPT-inapproximability of Determinant Maximization as stated below.

▶ **Theorem 5.1.** *Under the Parameterized Inapproximability Hypothesis, it is* W[1]*-hard to approximate* Determinant Maximization *within a factor of* $2^{-c\sqrt{k}}$ *for some universal constant* $c > 0$ *when parameterized by the number* k *of vectors to be selected. Moreover, the same hardness result holds even if the diagonal entries of an input matrix are restricted to* 1*.*

Since the above result relies on the Parameterized Inapproximability Hypothesis, Section 5.1 begins with its formal definition.

5.1 Inapproximability of Grid Tiling under Parameterized Inapproximability Hypothesis

We first introduce Binary Constraint Satisfaction Problem, for which the Parameterized Inapproximability Hypothesis asserts FPT-inapproximability. For two integers n and k, we are given a set $V \triangleq [k]$ of k variables, an alphabet $\Sigma \triangleq [n]$ of size n, and a set of constraints $\mathcal{C} = (C_{i,j})_{i,j\in V}$ such that $C_{i,j} \subseteq \Sigma^2$.[2] Each variable $i \in V$ may take a value from Σ. Each constraint $C_{i,j}$ specifies the pairs of values that variables i and j can take simultaneously, and it is said to be *satisfied* by an assignment $\psi : V \to \Sigma$ of values to the variables if $(\psi(i), \psi(j)) \in C_{i,j}$.

▶ **Problem 5.2.** *Given a set* V *of* k *variables, an alphabet set* Σ *of size* n*, and a set of constraints* $\mathcal{C} = (C_{i,j})_{i,j\in V}$*,* Binary Constraint Satisfaction Problem *(*BCSP*) asks to find an assignment* $\psi : V \to \Sigma$ *that satisfies the maximum fraction of constraints.*

It is well known that BCSP parameterized by the number k of variables is W[1]-complete from a standard parameterized reduction from k-Clique. Lokshtanov et al. [30] posed a conjecture asserting that a constant-factor gap version of BCSP is also W[1]-hard.

▶ **Hypothesis 5.3** (Parameterized Inapproximability Hypothesis (PIH) [30])**.** *There exists some universal constant* $\varepsilon \in (0, 1)$ *such that it is* W[1]*-hard to distinguish between* BCSP *instances that are promised to either be satisfiable, or have a property that every assignment violates at least* ε*-fraction of the constraints.*

[2] Though each constraint is actually indexed by an unordered pair of variables $\{i, j\}$, we use the present notation $C_{i,j}$ for sake of clarity and assume that $C_{i,j} = C_{j,i}$ without loss of generality.

Here, we prove that an *optimization version* of Grid Tiling is FPT-inapproximable assuming PIH. Given an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of Grid Tiling and an assignment $\sigma : [k]^2 \to [n]^2$, $\sigma(i,j)$ and $\sigma(i',j')$ for a pair of adjacent cells $(i,j,i',j') \in \mathcal{I}$ are said to be *consistent* if they agree on the first coordinate when $j = j'$ or on the second coordinate when $i = i'$, and *inconsistent* otherwise. The *consistency* of σ, denoted $\mathrm{cons}(\sigma)$, is defined as the number of pairs of adjacent cells that are consistent; namely,

$$\mathrm{cons}(\sigma) \triangleq \sum_{(i_1,j_1,i_2,j_2)\in\mathcal{I}} \left[\!\!\left[\sigma(i_1,j_1) \text{ and } \sigma(i_2,j_2) \text{ are consistent}\right]\!\!\right].$$

The *inconsistency* of σ is defined as the number of inconsistent pairs of adjacent cells. The optimization version of Grid Tiling asks to find an assignment σ such that $\mathrm{cons}(\sigma)$ is maximized.[3] Note that the maximum possible consistency is $|\mathcal{I}| = 2k^2$. We will use $\mathrm{opt}(\mathcal{S})$ to denote the optimal consistency among all possible assignments. We now demonstrate that Grid Tiling is FPT-inapproximable in an *additive sense* under PIH, whose proof is reminiscent of [33].

▶ **Lemma 5.4** (∗). *Under PIH, there exists some universal constant $\delta \in (0,1)$ such that it is* W[1]*-hard to distinguish* Grid Tiling *instances between the following cases:*

- *Completeness: the optimal consistency is $2k^2$.*
- *Soundness: the optimal consistency is at most $2k^2 - \delta k$.*

It should be noted that we may not be able to significantly improve the additive term $\mathcal{O}(k)$ owing to a polynomial-time εk^2-additive approximation algorithm for any constant $\varepsilon > 0$:

▶ **Observation 5.5** (∗). *Given an instance of* Grid Tiling *and an error tolerance parameter $\varepsilon > 0$, we can find an assignment whose consistency is at least $\mathrm{opt}(\mathcal{S}) - \varepsilon k^2$ in $\varepsilon^2 k^2 n^{\mathcal{O}(1/\varepsilon^2)}$ time.*

Our technical result is a gap-preserving parameterized reduction from Grid Tiling to Determinant Maximization, whose proof is presented in the subsequent subsection.

▶ **Lemma 5.6.** *There is a polynomial-time, parameterized reduction from an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of* Grid Tiling *to an instance $(\mathbf{A}, k^2)$ of* Determinant Maximization *such that all diagonal entries of $\mathbf{A}$ are 1 and the following conditions are satisfied:*

- *Completeness: If $\mathrm{opt}(\mathcal{S}) = 2k^2$, then $\mathrm{maxdet}(\mathbf{A}, k^2) = 1$.*
- *Soundness: If $\mathrm{opt}(\mathcal{S}) \leqslant 2k^2 - \delta k$ for some $\delta > 0$, then $\mathrm{maxdet}(\mathbf{A}, k^2) \leqslant 0.999^{\delta k}$.*

Using Lemma 5.6, we can prove Theorem 5.1.

Proof of Theorem 5.1. Our gap-preserving parameterized reduction is as follows. Given an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of Grid Tiling, we construct an instance $(\mathbf{A} \in \mathbb{Q}^{N\times N}, K \triangleq k^2)$ of Determinant Maximization in polynomial time according to Lemma 5.6, where $N \triangleq \sum_{i,j\in[k]} |S_{i,j}|$. The diagonal entries of $\mathbf{A}$ are 1 by definition. Since K is a function only in k, this is a parameterized reduction. According to Lemmas 5.4 and 5.6, it is W[1]-hard to determine whether $\mathrm{maxdet}(\mathbf{V}, K) = 1$ or $\mathrm{maxdet}(\mathbf{V}, K) \leqslant 0.999^{\delta k}$ under PIH, where $\delta \in (0,1)$ is a constant appearing Lemma 5.4. In particular, Determinant Maximization is W[1]-hard to approximate within a factor better than $0.999^{\delta k} = 2^{-c\sqrt{K}}$ when parameterized by K, where $c \in (0,1)$ is some universal constant. This completes the proof. ◀

[3] Our definition is different from Marx [33] in that the latter seeks a partial assignment such that the number of defined cells is maximized while the former requires maximizing the number of consistent adjacent pairs.

5.2 Gap-preserving Reduction from Grid Tiling and Proof of Lemma 5.6

To prove Lemma 5.6, we describe a gap-preserving parameterized reduction from Grid Tiling to Determinant Maximization. Before going into its details, we introduce a convenient gadget due to Çivril and Magdon-Ismail [11].

▶ **Lemma 5.7** (Çivril and Magdon-Ismail [11, Lemma 13]). *For any positive even integer ℓ, we can construct a set of 2^ℓ rational vectors $\mathbf{B}^{(\ell)} = \{\mathbf{b}_1, \dots, \mathbf{b}_{2^\ell}\}$ of dimension $2^{\ell+1}$ in $\mathcal{O}(4^\ell)$ time such that the following conditions are satisfied:*

- *Each entry of vectors is either 0 or $2^{-\frac{\ell}{2}}$; $\|\mathbf{b}_i\| = 1$ for all $i \in [2^\ell]$.*
- *$\langle \mathbf{b}_i, \mathbf{b}_j \rangle = \frac{1}{2}$ for all $i, j \in [2^\ell]$ with $i \neq j$.*
- *$\langle \mathbf{b}_i, \overline{\mathbf{b}_j} \rangle = \frac{1}{2}$ for all $i, j \in [2^\ell]$ with $i \neq j$, where $\overline{\mathbf{b}_j} \triangleq 2^{-\frac{\ell}{2}} \cdot \mathbf{1} - \mathbf{b}_j$.*

By definition of $\mathbf{B}^{(\ell)}$, we further have the following:

$$\langle \mathbf{b}_i, \overline{\mathbf{b}_i} \rangle = 2^{-\frac{\ell}{2}} \langle \mathbf{1}, \mathbf{b}_i \rangle - \langle \mathbf{b}_i, \mathbf{b}_i \rangle = 0,$$
$$\langle \overline{\mathbf{b}_i}, \overline{\mathbf{b}_j} \rangle = \langle 2^{-\frac{\ell}{2}} \mathbf{1} - \mathbf{b}_i, 2^{-\frac{\ell}{2}} \mathbf{1} - \mathbf{b}_j \rangle = 2^{-\ell} \langle \mathbf{1}, \mathbf{1} \rangle - 2^{-\frac{\ell}{2}} \langle \mathbf{1}, \mathbf{b}_i + \mathbf{b}_j \rangle + \langle \mathbf{b}_i, \mathbf{b}_j \rangle = \langle \mathbf{b}_i, \mathbf{b}_j \rangle.$$

Our reduction strategy is very similar to that of Theorem 4.2. Given an instance $\mathcal{S} = (S_{i,j})_{i,j \in [k]}$ of Grid Tiling, we construct a rational vector $\mathbf{v}_{x,y}^{(i,j)}$ for each element $(x, y) \in S_{i,j}$ of cell $(i, j) \in [k]^2$. Each vector consists of $|\mathcal{I}| = 2k^2$ blocks indexed by $\mathcal{I}$, each of which is either a vector in the set $\mathbf{B}^{(2\lceil \log n \rceil)}$ or the zero vector $\mathbf{0}$. Hence, the dimension of the vectors is $2k^2 \cdot 2^{2\lceil \log n \rceil + 1} = \mathcal{O}(k^2 n^2)$. Let $\mathbf{v}_{x,y}^{(i,j)}(i_1, j_1, i_2, j_2)$ denote the block of $\mathbf{v}_{x,y}^{(i,j)}$ corresponding to a pair of adjacent cells $(i_1, j_1, i_2, j_2) \in \mathcal{I}$. Each block is subsequently defined as follows:

$$\mathbf{v}_{x,y}^{(i,j)}(e) \triangleq \begin{cases} \overline{\mathbf{b}_x} & \text{if } e = (i-1, j, i, j), \\ \mathbf{b}_x & \text{if } e = (i, j, i+1, j), \\ \overline{\mathbf{b}_y} & \text{if } e = (i, j-1, i, j), \\ \mathbf{b}_y & \text{if } e = (i, j, i, j+1), \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Hereafter, two vectors $\mathbf{v}_{x,y}^{(i,j)}$ and $\mathbf{v}_{x',y'}^{(i',j')}$ are said to be *adjacent* if (i, j) and (i', j') are adjacent, and two adjacent vectors are said to be *consistent* if (x, y) and (x', y') are consistent (i.e., $x = x'$ whenever $j = j'$ and $y = y'$ whenever $i = i'$) and *inconsistent* otherwise. Since each vector contains exactly four vectors chosen from $\mathbf{B}^{(2\lceil \log n \rceil)}$, its squared norm is equal to 4. In addition, $\mathbf{v}_{x,y}^{(i,j)}$ and $\mathbf{v}_{x',y'}^{(i',j')}$ are orthogonal whenever (i, j) and (i', j') are not identical or adjacent. Observe further that if two cells are adjacent, the inner product of two vectors in $\mathbf{V}$ is calculated as follows:

$$\left\langle \mathbf{v}_{x,y}^{(i,j)}, \mathbf{v}_{x',y'}^{(i+1,j)} \right\rangle = \langle \mathbf{b}_x, \overline{\mathbf{b}_{x'}} \rangle = \begin{cases} 0 & \text{if they are consistent } x = x', \\ \frac{1}{2} & \text{otherwise (i.e., } x \neq x'\text{)}, \end{cases} \tag{7}$$

$$\left\langle \mathbf{v}_{x,y}^{(i,j)}, \mathbf{v}_{x',y'}^{(i,j+1)} \right\rangle = \langle \mathbf{b}_y, \overline{\mathbf{b}_{y'}} \rangle = \begin{cases} 0 & \text{if they are consistent } y = y' \\ \frac{1}{2} & \text{otherwise (i.e., } y \neq y'\text{)}. \end{cases} \tag{8}$$

On the other hand, the inner product of two vectors in the same cell is as follows:

$$\left\langle \mathbf{v}_{x,y}^{(i,j)}, \mathbf{v}_{x',y'}^{(i,j)} \right\rangle = 2 \cdot \langle \mathbf{b}_x, \mathbf{b}_{x'} \rangle + 2 \cdot \langle \mathbf{b}_y, \mathbf{b}_{y'} \rangle = \begin{cases} 4 & \text{if } x = x' \text{ and } y = y', \\ 3 & \text{if } x = x' \text{ xor } y = y', \\ 2 & \text{if } x \neq x' \text{ and } y \neq y'. \end{cases} \tag{9}$$

We denote by $\mathbf{V}^{(i,j)}$ the set of vectors corresponding to the elements of $S_{i,j}$; i.e., $\mathbf{V}^{(i,j)} \triangleq \{\mathbf{v}_{x,y}^{(i,j)} : (x,y) \in S_{i,j}\}$ for each $i,j \in [k]$. We now define an instance $(\mathbf{V}, K)$ of Determinant Maximization as $\mathbf{V} \triangleq \bigcup_{i,j\in[k]} \mathbf{V}^{(i,j)}$ and $K \triangleq k^2$. Note that $\mathbf{V}$ contains $N \triangleq \sum_{i,j\in[k]} |S_{i,j}|$ vectors.

We now proceed to the proof of (the soundness argument of) Lemma 5.6. Let $\mathbf{S}$ be a set of k^2 vectors from $\mathbf{V}$. Define $\mathbf{S}^{(i,j)} \triangleq \mathbf{V}^{(i,j)} \cap \mathbf{S} = \{\mathbf{v}_{x,y}^{(i,j)} \in \mathbf{S} : (x,y) \in S_{i,j}\}$ for each $i,j \in [k]^2$. Denote by $\mathrm{cov}(\mathbf{S})$ the number of cells $(i,j) \in [k]^2$ such that $\mathbf{S}$ includes $\mathbf{v}_{x,y}^{(i,j)}$ for some (x,y); i.e., $\mathrm{cov}(\mathbf{S}) \triangleq \{(i,j) \in [k]^2 : \mathbf{S}^{(i,j)} \neq \emptyset\}$, and we also define $\mathrm{dup}(\mathbf{S}) \triangleq \{(i,j) \in [k]^2 : \mathbf{S}^{(i,j)} = \emptyset\}$. It follows from the definition that $\mathrm{cov}(\mathbf{S}) + \mathrm{dup}(\mathbf{S}) = k^2$ and $\mathrm{dup}(\mathbf{S})$ counts the total number of "duplicate" vectors in the same cell. We first present an upper bound on the volume of $\mathbf{S}$ in terms of $\mathrm{dup}(\mathbf{S})$, implying that we cannot select many duplicate vectors from the same cell.

▶ **Lemma 5.8** (∗). *If* $\mathrm{dup}(\mathbf{S}) \leqslant \frac{k^2}{2}$, *then it holds that*

$$\mathrm{vol}^2(\mathbf{S}) \leqslant 4^{k^2} \cdot \left(\frac{3}{4}\right)^{\mathrm{dup}(\mathbf{S})}.$$

We then present another upper bound on the volume of $\mathbf{S}$ in terms of the *inconsistency* of a partial solution of Grid Tiling constructed from the selected vectors. For a set $\mathbf{S}$ of k^2 vectors from $\mathbf{V}$, a *partial* assignment $\sigma_{\mathbf{S}} : [k]^2 \to [n]^2 \cup \{\bigstar\}$ for Grid Tiling is defined as

$$\sigma_{\mathbf{S}}(i,j) \triangleq \begin{cases} \text{any } (x,y) \text{ such that } \mathbf{v}_{x,y}^{(i,j)} \in \mathbf{S}^{(i,j)} & \text{if such } (x,y) \text{ exists,} \\ \bigstar & \text{otherwise (i.e., } \mathbf{S}^{(i,j)} = \emptyset\text{),} \end{cases}$$

where the symbol "$\bigstar$" means *undefined* and the choice of (x,y) is arbitrary. The inconsistency of a partial assignment $\sigma_{\mathbf{S}}$ is defined as

$$\sum_{(i_1,j_1,i_2,j_2)\in\mathcal{I}} \left[\!\!\left[\sigma(i_1,j_1) \neq \bigstar;\ \sigma(i_2,j_2) \neq \bigstar;\ \sigma(i_1,j_1) \text{ and } \sigma(i_2,j_2) \text{ are inconsistent}\right]\!\!\right].$$

Note that the sum of the consistency and inconsistency of $\sigma_{\mathbf{S}}$ is no longer necessarily $2k^2$. Using $\sigma_{\mathbf{S}}$, we define a partition $(\mathbf{P}, \mathbf{Q})$ of $\mathbf{S}$ as $\mathbf{P} \triangleq \{\mathbf{v}_{x,y}^{(i,j)} \in \mathbf{S} : i,j \in [k], \sigma_{\mathbf{S}}(i,j) = (x,y)\}$ and $\mathbf{Q} \triangleq \mathbf{S} \setminus \mathbf{P}$. We further prepare an arbitrary *ordering* $\prec$ over $[k]^2$; e.g., $(i,j) \prec (i',j')$ if $i < i'$, or $i = i'$ and $j < j'$. We abuse the notation by writing $\mathbf{v}_{x,y}^{(i,j)} \prec \mathbf{v}_{x',y'}^{(i',j')}$ for any two vectors of $\mathbf{V}$ whenever $(i,j) \prec (i',j')$. Define now $\mathbf{P}_{\prec\mathbf{v}} \triangleq \{\mathbf{u} \in \mathbf{P} : \mathbf{u} \prec \mathbf{v}\}$. The following lemma states that the squared volume of k^2 vectors exponentially decays in the minimum possible inconsistency among all assignments of $\mathcal{S}$.

▶ **Lemma 5.9** (∗). *Suppose* $\mathrm{opt}(\mathcal{S}) \leqslant 2k^2 - \delta k$ *for some* $\delta > 0$ *and* $\mathrm{cov}(\mathbf{S}) \geqslant k^2 - \gamma k$ *for some* $\gamma > 0$. *If* $\delta k - 4\gamma k$ *is positive, then it holds that*

$$\mathrm{vol}^2(\mathbf{S}) \leqslant 4^{k^2} \cdot \left(\frac{63}{64}\right)^{\frac{\delta-4\gamma}{4}k}.$$

The proof of Lemma 5.9 involves the following claim.

▷ Claim 5.10 (∗). Suppose the same conditions as in Lemma 5.9 are satisfied. Then, the inconsistency of $\sigma_{\mathbf{S}}$ is at least $\delta k - 4\gamma k$. Moreover, the number of vectors $\mathbf{v}$ in $\mathbf{P}$ such that $\mathbf{v}$ is inconsistent with some adjacent vector of $\mathbf{P}_{\prec\mathbf{v}}$ is at least $\frac{\delta k - 4\gamma k}{4}$.

Using Lemmas 5.8 and 5.9, we can easily conclude Lemma 5.6 as follows.

Proof of Lemma 5.6. Observe that the reduction described in Section 5.2 is a parameterized reduction as it requires polynomial time and an instance $\mathcal{S} = (S_{i,j})_{i,j\in[k]}$ of Grid Tiling is transformed into an instance $(\mathbf{V}, k^2)$ of Determinant Maximization. In addition, the construction of $\mathbf{B}^{(2\lceil \log n\rceil)}$ completes in time $\mathcal{O}(4^{2\lceil \log n\rceil}) = \mathcal{O}(n^4)$ by Lemma 5.7.

We now prove the correctness of the reduction. Let us begin with the completeness argument. Suppose $\mathrm{opt}(\mathcal{S}) = 2k^2$; i.e., there is an assignment σ of consistency $2k^2$. Then, k^2 vectors in the set $\mathbf{S} \triangleq \{\mathbf{v}^{(i,j)}_{\sigma(i,j)} \colon i,j \in [k]\}$ are orthogonal to each other, implying that $\mathrm{vol}^2(\mathbf{S}) = 4^{k^2}$. On the other hand, because every vector of $\mathbf{V}$ is of squared norm 4, the maximum possible squared volume among k^2 vectors in $\mathbf{V}$ is 4^{k^2}; namely, $\mathrm{maxdet}(\mathbf{V}, k^2) = 4^{k^2}$.

We then prove the soundness argument. Suppose $\mathrm{opt}(\mathcal{S}) \leqslant 2k^2 - \delta k$ for some constant $\delta > 0$. Then, for any set $\mathcal{S}$ of k^2 vectors from $\mathbf{V}$ such that $\mathrm{dup}(\mathbf{S}) > \frac{\log 0.999^{-1}}{\log(\frac{3}{4})^{-1}} \cdot \delta k$, we have that by Lemma 5.8, $\mathrm{vol}^2(\mathbf{S}) < 4^{k^2} \cdot 0.999^{\delta k}$. It is thus sufficient to consider the case that

$$\mathrm{dup}(\mathbf{S}) \leqslant \frac{\log 0.999^{-1}}{\log(\frac{3}{4})^{-1}} \cdot \delta k \approx 0.0035 \cdot \delta.$$

In particular, it suffices to assume that $\mathrm{dup}(\mathbf{S}) \leqslant \gamma k$ for some $\gamma \in (0, \frac{\delta}{4})$. Simple calculation using Lemmas 5.8 and 5.9 derives that

$$\begin{aligned}\mathrm{vol}^2(\mathbf{S}) &\leqslant \min\left\{4^{k^2} \cdot \left(\frac{3}{4}\right)^{\gamma k}, 4^{k^2} \cdot \left(\frac{63}{64}\right)^{\frac{\delta-4\gamma}{4}k}\right\} \leqslant 4^{k^2} \cdot \min\left\{\left(\frac{63}{64}\right)^{\gamma k}, \left(\frac{63}{64}\right)^{\frac{\delta-4\gamma}{4}k}\right\} \\ &\leqslant 4^{k^2} \cdot \left(\frac{63}{64}\right)^{\underbrace{\min\left\{\gamma, \frac{\delta-4\gamma}{4}\right\}}_{\heartsuit} \cdot k} \leqslant 4^{k^2} \cdot \left(\frac{63}{64}\right)^{\frac{\delta}{8}k} \leqslant 4^{k^2} \cdot 0.999^{\delta k},\end{aligned}$$

where the second-to-last inequality is due to the fact that $\heartsuit$ is maximized when $\gamma = \frac{\delta-4\gamma}{4}$; i.e., $\gamma = \frac{\delta}{8} > 0$.

Because the diagonal entries of the Gram matrix $\mathbf{A}$ defined by the vectors of $\mathbf{V}$ are 4, we can construct another instance of Determinant Maximization as $(\widetilde{\mathbf{A}}, k^2)$, where $\widetilde{\mathbf{A}} \triangleq \frac{1}{4}\mathbf{A}$. Observe finally that the diagonal entries of $\widetilde{\mathbf{A}}$ are 1 and $\det(\widetilde{\mathbf{A}}_S) = 4^{-|S|} \cdot \det(\mathbf{A}_S)$ for any S, which completes the proof. ◀

5.3 ε-Additive FPT-Approximation Parameterized by Rank

Here, we develop an ε-additive FPT-approximation algorithm parameterized by the *rank* of an input matrix $\mathbf{A}$, provided that $\mathbf{A}$ is the Gram matrix of vectors of *infinity norm at most* 1. Our algorithm complements Lemma 5.6 in a sense that we can solve the promise problem in FPT time with respect to $\mathrm{rank}(\mathbf{A})$. The proof uses the standard rounding technique.

▶ **Observation 5.11** (∗). *Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be n d-dimensional vectors in $\mathbb{Q}^d$ such that $\|\mathbf{v}_i\|_\infty \leqslant 1$ for all $i \in [n]$, $\mathbf{A}$ the Gram matrix defined by the vectors, $k \in [d]$ a positive integer, and $\varepsilon > 0$ an error tolerance parameter. Then, we can compute an approximate solution $S \in \binom{[n]}{k}$ to* Determinant Maximization *in $\varepsilon^{-d^2} \cdot d^{\mathcal{O}(d^3)} \cdot n^{\mathcal{O}(1)}$ time such that $\det(\mathbf{A}_S) \geqslant \mathrm{maxdet}(\mathbf{A}, k) - \varepsilon$.*

6 Open Problems

We investigated the W[1]-hardness of Determinant Maximization in the three restricted cases, improving upon the result due to Koutis [26]. Our parameterized hardness results leave a few natural open problems: For what kinds of sparse matrices is Determinant Maximization FPT? Is there a $(1-\varepsilon)$-factor (rather than "additive") FPT-approximation algorithm with respect to the matrix rank? Quantitative lower bounds can be also proved; e.g., due to the lower bound of k-Sum [41], Determinant Maximization on tridiagonal matrices cannot be solved in $n^{o(k)}$ time, unless *Exponential Time Hypothesis* [23, 24] fails.

References

1 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *ESA*, pages 1–12, 2014.
2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.
4 Rajendra Bhatia. *Perturbation Bounds for Matrix Eigenvalues*. SIAM, 2007.
5 Erdem Bıyık, Kenneth Wang, Nima Anari, and Dorsa Sadigh. Batch active learning using determinantal point processes. *CoRR*, abs/1906.07975, 2019. `arXiv:1906.07975`.
6 Alexei Borodin and Eric M. Rains. Eynard-Mehta theorem, Schur process, and their Pfaffian analogs. *J. Stat. Phys.*, 121(3–4):291–317, 2005.
7 L. Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth K. Vishnoi. Fair and diverse DPP-based data summarization. In *ICML*, pages 715–724, 2018.
8 Wei-Lun Chao, Boqing Gong, Kristen Grauman, and Fei Sha. Large-margin determinantal point processes. In *UAI*, pages 191–200, 2015.
9 Diego Cifuentes and Pablo A. Parrilo. An efficient tree decomposition method for permanents and mixed discriminants. *Linear Algebra Appl.*, 493:45–81, 2016.
10 Ali Çivril and Malik Magdon-Ismail. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theor. Comput. Sci.*, 410(47-49):4801–4811, 2009.
11 Ali Çivril and Malik Magdon-Ismail. Exponential inapproximability of selecting a maximum volume sub-matrix. *Algorithmica*, 65(1):159–176, 2013.
12 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.*, 108(1-2):23–52, 2001.
13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
14 Marco Di Summa, Friedrich Eisenbrand, Yuri Faenza, and Carsten Moldenhauer. On largest volume simplices and sub-determinants. In *SODA*, pages 315–323, 2014.
15 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theor. Comput. Sci.*, 141(1-2):109–131, 1995.
16 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 2012.
17 Andreas Emil Feldmann, C. S. Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.
18 Leonardo Pisano Fibonacci. The book of squares, 1225.
19 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.
20 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. An EATCS Series: Texts in Theoretical Computer Science. Springer, 2010.

21 Peter Gritzmann, Victor Klee, and David G. Larman. Largest j-simplices in n-polytopes. *Discrete Comput. Geom.*, 13:477–515, 1995.

22 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, pages 291–300, 2004.

23 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

24 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

25 Chun-Wa Ko, Jon Lee, and Maurice Queyranne. An exact algorithm for maximum entropy sampling. *Oper. Res.*, 43(4):684–691, 1995.

26 Ioannis Koutis. Parameterized complexity and improved inapproximability for computing the largest j-simplex in a V-polytope. *Inf. Process. Lett.*, 100(1):8–13, 2006.

27 Alex Kulesza and Ben Taskar. k-DPPs: Fixed-size determinantal point processes. In *ICML*, pages 1193–1200, 2011.

28 Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Found. Trends Mach. Learn.*, 5(2–3):123–286, 2012.

29 Donghoon Lee, Geonho Cha, Ming-Hsuan Yang, and Songhwai Oh. Individualness and determinantal point processes for pedestrian detection. In *ECCV*, pages 330–346, 2016.

30 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurab, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *SODA*, pages 2181–2200, 2020.

31 Odile Macchi. The coincidence approach to stochastic point processes. *Adv. Appl. Probab.*, 7(1):83–122, 1975.

32 Vivek Madan, Aleksandar Nikolov, Mohit Singh, and Uthaipon Tantipongpipat. Maximizing determinants under matroid constraints. In *FOCS*, pages 565–576, 2020.

33 Dániel Marx. On the optimality of planar and geometric approximation schemes. In *FOCS*, pages 338–348, 2007.

34 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

35 Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In *ICALP*, pages 677–688, 2012.

36 Aleksandar Nikolov. Randomized rounding for the largest simplex problem. In *STOC*, pages 861–870, 2015.

37 Aleksandar Nikolov and Mohit Singh. Maximizing determinants under partition constraints. In *STOC*, pages 192–201, 2016.

38 Naoto Ohsaka. On the parameterized intractability of determinant maximization. *CoRR*, abs/2209.12519, 2022. `arXiv:2209.12519`.

39 Naoto Ohsaka. Some inapproximability results of MAP inference and exponentiated determinantal point processes. *J. Artif. Intell. Res.*, 73:709–735, 2022.

40 Naoto Ohsaka and Tatsuya Matsuoka. On the (in)tractability of computing normalizing constants for the product of determinantal point processes. In *ICML*, pages 7414–7423, 2020.

41 Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *SODA*, pages 1065–1075, 2010.

42 Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. Practical diversified recommendations on YouTube with determinantal point processes. In *CIKM*, pages 2165–2173, 2018.

43 Jin-ge Yao, Feifan Fan, Wayne Xin Zhao, Xiaojun Wan, Edward Y. Chang, and Jianguo Xiao. Tweet timeline generation with determinantal point processes. In *AAAI*, pages 3080–3086, 2016.

44 Martin J. Zhang and Zhijian Ou. Block-wise MAP inference for determinantal point processes with application to change-point detection. In *SSP*, pages 1–5, 2016.

One-Face Shortest Disjoint Paths with a Deviation Terminal

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Tatsuya Terao
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

For an undirected graph G and distinct vertices $s_1, t_1, \ldots, s_k, t_k$ called terminals, the *shortest k-disjoint paths problem* asks for k pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots, k$ and the sum of their lengths is minimized. This problem is a natural optimization version of the well-known k-disjoint paths problem, and its polynomial solvability is widely open. One of the best results on the shortest k-disjoint paths problem is due to Datta et al. [9], who present a polynomial-time algorithm for the case when G is planar and all the terminals are on one face. In this paper, we extend this result by giving a polynomial-time randomized algorithm for the case when all the terminals except one are on some face of G. In our algorithm, we combine the arguments of Datta et al. with some results on the shortest disjoint $(A + B)$-paths problem shown by Hirai and Namba [15]. To this end, we present a non-trivial bijection between k disjoint paths and disjoint $(A + B)$-paths, which is a key technical contribution of this paper.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Combinatorial algorithms

Keywords and phrases shortest disjoint paths, polynomial time algorithm, planar graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.47

Funding Supported by JSPS KAKENHI Grant Numbers 18H05291 and 20K11692.

1 Introduction

1.1 Shortest Disjoint Paths Problem

The *k-disjoint paths problem* is a well-studied and important problem in algorithmic graph theory and combinatorial optimization. In the problem, we are given an undirected graph $G = (V, E)$ and $2k$ distinct vertices $s_1, t_1, \ldots, s_k, t_k$, called *terminals*, and the objective is to find k pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots, k$, if they exist. This problem has attracted attention since 1980s because of its applications to practical problems such as network routing [24, 34] and VLSI-design [13, 19].

The main focus in this topic is the polynomial solvability of the problem. When k is a part of the input, the k-disjoint paths problem is shown to be NP-hard by Karp [16], and it remains NP-hard even if the input graph is restricted to be planar [22]. When $k = 2$, elementary polynomial-time algorithms are presented in [32, 33, 36], whereas the directed variant is NP-hard [12]. For the case when the graph is undirected and k is a fixed constant, Robertson and Seymour's graph minor theory gives a polynomial-time algorithm [29], which is one of the biggest achievements in this area.

An interesting special case of the disjoint paths problem is when the input graph is planar or embedded on a fixed surface. For example, when G is planar and all the terminals are on one face or two faces, a structural characterization and a polynomial algorithm are given in [27]. The k-disjoint paths problem on a fixed surface is solved in [28]. In the graph minor series, these special cases play important roles to obtain the algorithm for general

graphs [29]. A faster algorithm for the planar case is presented in [1]. The directed variant of the problem can be solved in polynomial time if the input digraph is planar and k is a fixed constant [8, 30].

The *shortest k-disjoint paths problem* is a natural optimization version of the k-disjoint paths problem. In the problem, we are given an undirected graph $G = (V, E)$ and terminals $s_1, t_1, \ldots, s_k, t_k$, and the objective is to find k pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots, k$ and the sum of their lengths is minimized. Here, the length of a path is defined as the number of edges in it. When k is a part of the input, since the search problem is NP-hard, so is the optimization problem. Although the problem setting is natural and easy to understand, surprisingly, the polynomial solvability of the shortest k-disjoint paths problem is widely open when k is a fixed constant. For the shortest 2-disjoint paths problem, Björklund and Husfeldt [5] present a randomized polynomial-time algorithm based on an algebraic approach, while a deterministic polynomial-time algorithm is still unknown. They also give a deterministic polynomial-time algorithm for counting optimal solutions of the shortest 2-disjoint paths problem if the input graph is cubic and planar [4].

Several positive results are obtained if G is planar and the configuration of the terminals satisfies certain conditions. Colin de Verdière and Schrijver [7] devise an $O(kn \log n)$ time algorithm for the shortest k-disjoint paths problem when all sources $s_1, \ldots, s_k$ are on one face and all sinks $t_1, \ldots, t_k$ are on another face, where $n = |V|$. Kobayashi and Sommer [18] give a polynomial-time algorithm for the case when $k = 2$ and the terminals are on two faces in an arbitrary way. The problem is not easy even when all the terminals are on one face, which we call One-Face Shortest k-disjoint paths problem. Polynomial-time algorithms are presented for the cases when $k = 3$ [18] and $k = 4$ [11]. Borradaile et al. [6] give an $O(kn^5)$ time algorithm for the case when $s_1, t_1, s_2, t_2, \ldots, s_k, t_k$ are on the boundary of some face counter-clockwise in this order. Datta et al. [9] generalize these results by presenting a polynomial-time algorithm for One-Face Shortest k-disjoint paths problem when k is a fixed constant. Their idea is to compute a polynomial associated with k disjoint paths by using determinants of several polynomial matrices, and a similar argument will be used in this paper.

1.2 Our Contribution

As described in the previous subsection, for the shortest k-disjoint paths problem, polynomial-time algorithms are devised only for very restricted cases. Among them, one of the strongest results is a polynomial-time algorithm for One-Face Shortest k-disjoint paths problem due to Datta et al. [9]. In this paper, we extend this result by dealing with the case when all the terminals except one are on some face of G, which we call One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal.

One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal

Input: A planar graph $G = (V, E)$ and distinct vertices $s_1, t_1, \ldots, s_k, t_k$ called terminals such that $2k - 1$ of them are on the boundary of some face of G.

Output: Pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i and t_i for $i = 1, \ldots, k$ and the sum of their lengths is minimized (if they exist).

The main contribution of this paper is to present a randomized polynomial-time algorithm for this problem.

▶ **Theorem 1.** *For a fixed positive integer k,* One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal *can be solved in randomized polynomial time.*

Although our problem looks similar to One-Face Shortest k-disjoint paths problem, there is a big gap between these two problems in the following sense. In One-Face Shortest k-disjoint paths problem, if $s_1, s_2, \ldots, s_k, t_k, \ldots, t_2$, and t_1 are on the boundary of a face in this order, then the problem can be solved easily (e.g., by a minimum cost flow algorithm or by a determinant computation). This special case acts as a "base case" in the algorithm of Datta et al. [9]. In contrast, in One-Face Shortest k-disjoint paths problem with a Deviation Terminal, such an easy "base case" does not exist. This difference makes the problem quite difficult and interesting, and reinforces the importance of Theorem 1.

We overcome the above difficulty by giving a non-trivial bijection between k disjoint paths and disjoint $(A+B)$-paths introduced by Hirai and Namba [15] (see Section 2.2 for disjoint $(A+B)$-paths), which is a key ingredient in our algorithm. By combining several polynomials associated with disjoint $(A+B)$-paths, we compute a polynomial associated with the desired k disjoint paths in a similar way to [9], which enables us to compute an optimal solution.

1.3 Related Work

The *k-disjoint shortest paths problem* is another variant of the k-disjoint paths problem that is actively studied recently. In the problem, an instance consists of a graph and terminals $s_1, t_1, \ldots, s_k, t_k$ in the same way as the k-disjoint paths problem. The objective is to find pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i is a *shortest* path between s_i and t_i for $i = 1, \ldots, k$. It is obvious that if each P_i is a shortest path, then the total length is minimized. Therefore, algorithms for the shortest k-disjoint paths problem can solve the k-disjoint shortest paths problem. Eilam-Tzoreff [10] introduces the k-disjoint shortest paths problem and devises a polynomial-time algorithm for the case of $k = 2$. For the case when k is a fixed constant, polynomial-time algorithms are recently given by Lochet [21] and Bentert et al. [2]. When each edge has a non-negative length, Gottschau et al. [14] and Kobayashi and Sako [17] independently give polynomial-time algorithms for the 2-disjoint shortest paths problem. The polynomial solvability of the k-disjoint shortest paths problem with non-negative edge-length is still open for fixed k. For the directed variant with positive edge-length, Bérczi and Kobayashi [3] present a polynomial-time algorithm when $k = 2$.

A special case when (almost) all the terminals are on the boundary of some face has attracted attention also in the context of the edge-disjoint paths problem and the multicommodity flow problem. The most famous example will be Okamura-Seymour Theorem [26]. Suppose that all the terminals are on the boundary of some face of a planar graph and assume that the Euler condition holds. Then, Okamura-Seymour Theorem states that the existence of edge-disjoint paths connecting the terminal pairs can be characterized by the cut condition. This theorem is generalized by Okamura [25] to the case when the terminals are on two faces. A node-capacitated variant of Okamura-Seymour Theorem is studied by Lee et al. [20]. See [31, Chapter 74] for more related results.

1.4 Organization

The remaining of this paper is organized as follows. In Section 2, we give notation and describe some results on the shortest disjoint $(A+B)$-paths problem by Hirai and Namba [15]. In Sections 3.1 and 3.2, we show a bijection between k disjoint paths and disjoint $(A+B)$-paths without giving a proof of a key lemma (Lemma 7). By using this bijection, in Section 3.3, we present a randomized algorithm for One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal and prove Theorem 1. The proof of Lemma 7 is given in Section 4.

2 Preliminaries

2.1 Basic Notation

For a positive integer p, we denote $[p] = \{1, \ldots, p\}$. For a finite set X, let $\#X$ and $|X|$ denote the cardinality of X, which is also called the *size* of X. In this paper, all graphs and paths are undirected unless stated otherwise. The *length* of a path in a graph is defined as the number of edges in it. For a collection $\mathcal{P}$ of paths, let $w(\mathcal{P})$ denote the total length of the paths in $\mathcal{P}$.

Let k be a fixed positive integer. Suppose we are given an instance of One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal, which consists of a planar graph $G = (V, E)$ and terminals $s_1, t_1, \ldots, s_k, t_k$. Let T denote the set of all the terminals. Suppose that all the terminals except one terminal, say s_1, are on the boundary of some face F of G. Without loss of generality, we may assume that G is 2-connected, since otherwise we can easily reduce to the 2-connected case. Then, the boundary of F forms a cycle.

For a finite set X of size $2p$, a partition of X into p disjoint sets of size two is called a *pairing of* X. That is, a pairing M of X is of the form $M = \{\{x_1, y_1\}, \ldots, \{x_p, y_p\}\}$, where $X = \{x_1, y_1, \ldots, x_p, y_p\}$. Although each element in M is an unordered pair, by following the convention, each pair $\{x_i, y_i\}$ is denoted by (x_i, y_i) in this paper. Thus, (x_i, y_i) is identified with (y_i, x_i). A pairing of T is simply called a *pairing* if T is clear from the context. The pairing $M^* = \{(s_i, t_i) \mid i \in [k]\}$ of T is referred to as the *input pairing*.

In our argument, the ordering of the terminals along the boundary of F is more important than the input pairing. Hence, we rename the terminals so that $T = \{u^*, u_0, u_1, \ldots u_{2k-2}\}$, $u^* = s_1$, $u_0 = t_1$, and $u_0, u_1, \ldots, u_{2k-2}$ lie on the boundary of F counter-clockwise in this order (Figure 1). Let $I = \{0, 1, \ldots, 2k-2\}$ be the set of the indices of the vertices incident to F.

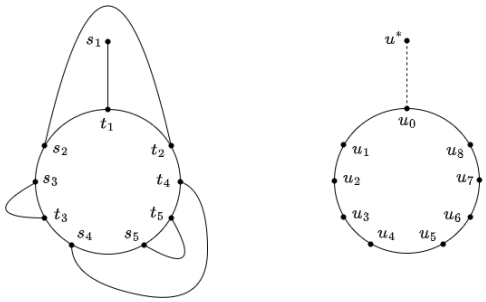


Figure 1 Renaming the terminals.

2.2 The Shortest Disjoint $(A + B)$-Paths Problem

Hirai and Namba [15] introduce the *shortest disjoint* $(A+B)$*-paths problem* as a generalization of the shortest 2-disjoint paths problem. In the shortest disjoint $(A + B)$-paths problem, we are given a graph $G = (V, E)$ and two disjoint terminal sets $A, B \subseteq V$ of even size, and the task is to find $|A|/2 + |B|/2$ pairwise vertex-disjoint paths with endpoints both in A or both in B such that the sum of their lengths is minimized. Note that each feasible solution is called *disjoint* $(A + B)$*-paths*. For this problem, Hirai and Namba [15] design a randomized algorithm running in $|V|^{O(|A|+|B|)}$ time, which is polynomial if $|A| + |B|$ is fixed. For the case when the input graph is cubic and planar, Björklund and Husfeldt [4] give a deterministic polynomial-time algorithm.

We now describe the outline of the algorithm in [15], because some of their results will be used in our argument later. In their algorithm, they construct a univariate polynomial matrix from the instance, and compute its *hafnian*. Although computing the hafnian of a matrix is hard in general, they establish a technique to compute the hafnian modulo 2^τ for a fixed positive integer τ based on a similar argument to [5]. Note that the hafnian of a $2n \times 2n$ symmetric matrix $S = (s_{ij})$ is defined as

$$\operatorname{haf} S = \sum_{M \in \mathcal{M}} \prod_{(i,j) \in M} s_{ij},$$

where $\mathcal{M}$ is the set of all pairings of $\{1, 2, 3, \ldots, 2n\}$. The key theorems in [15] are formally stated as follows.

▶ **Theorem 2** (Hirai and Namba [15, Lemma 2.4]). *For any instance of the shortest disjoint $(A + B)$-paths problem, in polynomial time, we can construct a matrix S such that each element is a polynomial in x with integer coefficients and*

$$\operatorname{haf} S = \sum_{\mathcal{P}} \pm 2^{\frac{|A|}{2}+\frac{|B|}{2}} x^{w(\mathcal{P})}(1 + x f_{\mathcal{P}}(x)),$$

where $\mathcal{P}$ ranges over all disjoint $(A + B)$-paths, $f_{\mathcal{P}}(x)$ is some polynomial with integer coefficients, and the sign is determined by A, B, and $\mathcal{P}$.

Note that $f_{\mathcal{P}}(x)$ depends only on $\mathcal{P}$, and it does not depend on how the end vertices of $\mathcal{P}$ are partitioned into A and B. Recall that $w(\mathcal{P})$ is the number of edges in the paths in $\mathcal{P}$.

▶ **Theorem 3** (Hirai and Namba [15, Theorem 2.1]). *Let τ be a fixed positive integer. For a given univariate matrix with integer coefficients, we can compute its hafnian modulo 2^τ in polynomial time.*

Note that "hafnian modulo 2^τ" means that the integer coefficients of the hafnian are modulo 2^τ. In their algorithm, after perturbing the length of edges to obtain an instance with a unique optimal solution, we construct a matrix S as in Theorem 2. Then, we compute the hafnian of S modulo $2^{\frac{|A|}{2}+\frac{|B|}{2}+1}$ by Theorem 3. We can see that the lowest degree term of the hafnian corresponds to the shortest disjoint $(A + B)$-paths under the assumption that the instance has a unique optimal solution. Note that the randomization is required only in the perturbation step, and the algorithms in Theorems 2 and 3 are deterministic.

3 One-Face Shortest k-Disjoint Paths with a Deviation Terminal

In this section, we present our algorithm for One-Face Shortest k-disjoint paths problem with a Deviation Terminal. In our algorithm, we consider several partitions (A, B) of the terminals T and compute polynomials associated with disjoint $(A + B)$-paths by using Theorems 2 and 3. By using these polynomials, we compute a polynomial whose lowest degree term corresponds to the shortest k disjoint paths. A key ingredient in our argument is to make a correspondence between a partition (A, B) and a pairing of T. More precisely, we show that there is a bijection between a *good* partition and a *feasible* pairing including (u^*, u_0), which will be defined in Section 3.1.

3.1 Good Partition

Let M be a pairing of T. We say that M is *infeasible* if there exist distinct pairs $(u_i, u_j), (u_{i'}, u_{j'}) \in M$ such that $i < i' < j < j'$, that is, $u_i, u_{i'}, u_j$, and $u_{j'}$ are on the boundary of F counter-clockwise in this order. Otherwise, M is called *feasible*. Observe that,

for a pairing M of T, if G contains k disjoint paths connecting the pairs in M, then M is feasible. For a partition (A, B) of T, a pairing M of T is called (A, B)-*compatible* if, for any $(s, t) \in M$, both s and t belong to the same set, either A or B.

In our algorithm, for several partitions (A, B) of T, we compute polynomials associated with disjoint $(A + B)$-paths by using Theorems 2 and 3. We focus on disjoint $(A + B)$-paths that contain a u^*-u_0 path, and so we consider partitions (A, B) of T such that every (A, B)-compatible pairing contains (u^*, u_0). Such a partition is called *good*, which is formally defined as follows.

▶ **Definition 4.** *A partition (A, B) of T is said to be* good *if*

- $u^*, u_0 \in A$,
- *there exists a feasible (A, B)-compatible pairing M with $(u^*, u_0) \in M$, and*
- *there exists no feasible (A, B)-compatible pairing M with $(u^*, u_i) \in M$ for $i \in [2k - 2]$.*

3.2 Bijection Between Good Partitions and Feasible Pairings

In this subsection, we give a bijection between good partitions of T and feasible pairings including (u^*, u_0). For this purpose, we show that both good partitions and feasible pairings are related to *ballot sequences*. A sequence $(f(1), \ldots, f(2k))$ of integers is called a k-*ballot sequence* if $f(i) \in \{+1, -1\}$ for $i \in [2k]$, $\sum_{i=1}^{\ell} f(i) \geq 0$ for $\ell \in [2k - 1]$, and $\sum_{i=1}^{2k} f(i) = 0$. It is well-known that the number of k-ballot sequences is equal to the k-th Catalan number $c_k = \frac{1}{k+1}\binom{2k}{k}$ (see [35]).

First, we give a bijection between $(k - 1)$-ballot sequences and feasible pairings including (u^*, u_0). For a $(k - 1)$-ballot sequence $(f(1), f(2), \ldots, f(2k - 2))$, we construct the corresponding pairing $g_1(f(1), f(2), \ldots, f(2k - 2))$ as follows.

▶ **Definition 5.** *Let $(f(1), f(2), \ldots, f(2k - 2))$ be a $(k - 1)$-ballot sequence. We initialize M as $M = \{(u^*, u_0)\}$. For every $i \in [2k - 2]$ with $f(i) = +1$, let $i' \in [2k - 2]$ be the minimum index such that $\sum_{j=1}^{i-1} f(j) = \sum_{j=1}^{i'} f(j)$ and $i < i'$, and add $(u_i, u_{i'})$ to M. Then, define $g_1(f(1), f(2), \ldots, f(2k - 2)) = M$.*

This construction is the same as a well-known bijection between ballot sequences and *legal sequences of parentheses* (see [35]). Since a legal sequence of parentheses can be identified with a feasible pairing including (u^*, u_0), we obtain the following lemma.

▶ **Lemma 6** (see [35])**.** *Mapping g_1 in Definition 5 is a bijection from $(k-1)$-ballot sequences to feasible pairings including (u^*, u_0).*

We next construct a bijection from good partitions of T to $(k - 1)$-ballot sequences. For a partition (A, B) of T, define

$$f_{(A,B)}(i) = \begin{cases} -1 & \text{if } u_i \in A \text{ and } i \text{ is odd,} \\ +1 & \text{if } u_i \in A \text{ and } i \text{ is even,} \\ +1 & \text{if } u_i \in B \text{ and } i \text{ is odd,} \\ -1 & \text{if } u_i \in B \text{ and } i \text{ is even} \end{cases}$$

for $i \in [2k - 2]$. We show the following lemma, whose proof is given in Section 4.

▶ **Lemma 7.** *Let (A, B) be a partition of T with $u^*, u_0 \in A$. Then, (A, B) is good if and only if $(f_{(A,B)}(1), \ldots, f_{(A,B)}(2k - 2))$ is a $(k - 1)$-ballot sequence.*

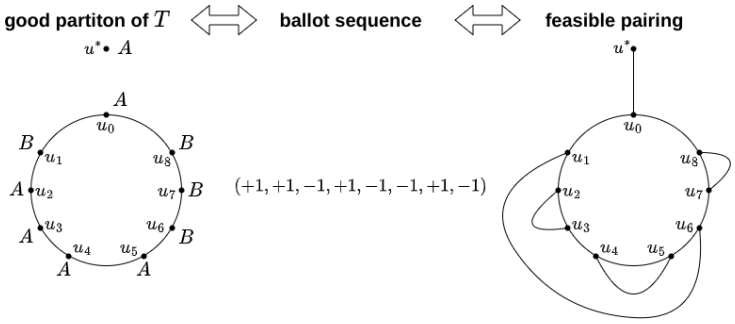


Figure 2 Correspondence among good partitions, ballot sequences, and feasible pairings.

By this lemma, the construction of $f_{(A,B)}$ defines a bijection g_2 from good partitions of T to $(k-1)$-ballot sequences. Therefore, by composing g_1 and g_2, we obtain a bijection g from good partitions to feasible pairings including (u^*, u_0); see Figure 2.

Since the number of ballot sequences is equal to the Catalan number, Lemma 7 implies the following result, which is of independent interest.

▶ **Corollary 8.** *The number of good partitions of $2k$ terminals T is equal to the Catalan number $c_{k-1} = \frac{1}{k}\binom{2k-2}{k-1}$.*

We next show a property of bijection g, which plays an important role in our algorithm. For a pairing M of T with $(u^*, u_0) \in M$, let $d(M) = \sum_{(u_i,u_j)\in M\setminus\{(u^*,u_0)\}} |i-j|$.

▶ **Lemma 9.** *For any good partition (A,B) of T, $g(A,B)$ is a feasible (A,B)-compatible pairing. Furthermore, among all feasible (A,B)-compatible pairings M, $g(A,B)$ is a unique minimizer of $d(M)$.*

Proof. By Lemmas 6 and 7, $g(A,B)$ is a feasible pairing with $(u^*,u_0) \in g(A,B)$. We first show that $g(A,B)$ is (A,B)-compatible. Suppose that $(u_i, u_{i'}) \in g(A,B)$ with $i < i'$. By the construction of bijection g_1 in Definition 5, we see that $i \in [2k-3]$ satisfies $f_{(A,B)}(i) = +1$, and $i' \in [2k-2]$ is the minimum index such that $\sum_{j=1}^{i-1} f_{(A,B)}(j) = \sum_{j=1}^{i'} f_{(A,B)}(j)$ and $i < i'$. Then, $f_{(A,B)}(i') = -1$ by the minimality of i', and $i - 1 \equiv i' \pmod 2$. Since $f_{(A,B)}(i) \neq f_{(A,B)}(i')$ and $i \not\equiv i' \pmod 2$, the definition of $f_{(A,B)}$ shows that both u_i and $u_{i'}$ belong to the same set, either A or B. Therefore, $g(A,B)$ is (A,B)-compatible.

To show the latter half of the lemma, let M be a feasible (A,B)-compatible pairing. Since (A,B) is a good partition of T, we obtain $(u^*,u_0) \in M$. If $(u_i,u_{i'}) \in M$ for $i, i' \in [2k-2]$, then i and i' have different parities by the feasibility of M. Since both u_i and $u_{i'}$ belong to the same set, either A or B, the definition of $f_{(A,B)}$ shows that

$$f_{(A,B)}(i) + f_{(A,B)}(i') = 0. \tag{1}$$

Define $I^+ = \{i \in [2k-2] \mid f_{(A,B)}(i) = +1\}$ and $I^- = \{i \in [2k-2] \mid f_{(A,B)}(i) = -1\}$. Then, (1) means that each pair in $M \setminus \{(u^*,u_0)\}$ consists of u_i with $i \in I^+$ and $u_{i'}$ with $i' \in I^-$. Furthermore, for any $(u_i,u_{i'}) \in g(A,B)$ with $i < i'$, the definition of g_1 shows that $i \in I^+$ and $i' \in I^-$. Therefore, we obtain

$$d(M) = \sum_{(u_i,u_{i'})\in M} |i'-i| \geq \sum_{\substack{(u_i,u_{i'})\in M \\ i\in I^+,\ i'\in I^-}} (i'-i) = \sum_{i'\in I^-} i' - \sum_{i\in I^+} i = d(g(A,B)). \tag{2}$$

This shows that $g(A,B)$ is a minimizer of $d(M)$.

To show the uniqueness of the minimizer, suppose that $M \neq g(A,B)$. It suffices to show that the inequality in (2) is strict, that is, there exists $(u_i, u_{i'}) \in M$ such that $i \in I^+$, $i' \in I^-$, and $i > i'$. For $(u_i, u_{i'}) \in M$, since the vertices between u_i and $u_{i'}$ are partitioned into pairs in M and (1) holds for any pair in $M \setminus \{(u^*, u_0)\}$, we obtain

$$\sum_{j=1}^{i-1} f_{(A,B)}(j) = \sum_{j=1}^{i'} f_{(A,B)}(j). \tag{3}$$

Since $M \neq g(A,B)$, there exist $i_0 \in I^+$ and $i'_0 \in I^-$ with $(u_{i_0}, u_{i'_0}) \in M \setminus g(A,B)$. If $i_0 > i'_0$, then we immediately conclude that the inequality in (2) is strict. In what follows, suppose that $i_0 < i'_0$. Let $i'_1 \in I^-$ be the minimum index such that $\sum_{j=1}^{i_0-1} f_{(A,B)}(j) = \sum_{j=1}^{i'_1} f_{(A,B)}(j)$ and $i_0 < i'_1$, that is, $(i_0, i'_1) \in g(A,B)$. By (3) and by the minimality of i'_1, we obtain $i'_1 < i'_0$. We now consider the pair $(u_{i_1}, u_{i'_1}) \in M$ containing $u_{i'_1}$, where $i_1 \in I^+$. Since M is feasible and $i_0 < i'_1 < i'_0$, we obtain $i_0 < i_1$. Furthermore, since $\sum_{j=1}^{i_1-1} f_{(A,B)}(j) = \sum_{j=1}^{i'_1} f_{(A,B)}(j)$ by (3), the minimality of i'_1 shows that i_1 is not contained in the interval between i_0 and i'_1. Therefore, we obtain $i'_1 < i_1$, and hence the inequality in (2) is strict. This shows that $g(A,B)$ is a unique minimizer of $d(M)$. ◀

3.3 Algorithm

We are now ready to describe our algorithm, which is based on an idea similar to [9]. In this subsection, since all computations of polynomials are done modulo 2^{k+1}, we regard polynomials with integer coefficients as elements in $\mathbb{Z}_{2^{k+1}}[x]$. For a set $\mathcal{P}$ of paths, define $f_{\mathcal{P}}(x)$ as in Theorem 2. For a feasible pairing M of T, define a polynomial $\Phi(M)$ by

$$\Phi(M) = \sum_{\mathcal{P}} 2^k x^{w(\mathcal{P})}(1 + x f_{\mathcal{P}}(x)),$$

where $\mathcal{P}$ ranges over all sets of k disjoint paths connecting the pairs in M.

Let $\mathcal{M}$ be the set of all feasible pairings including (u^*, u_0). For a good partition (A,B) of T, let $\mathcal{M}_{(A,B)}$ be the set of all feasible (A,B)-compatible pairings. Since (A,B) is a good partition, we see that $\mathcal{M}_{(A,B)} \subseteq \mathcal{M}$. With these notations, Theorem 2 shows that we can obtain a polynomial matrix $S_{(A,B)}$ such that

$$\operatorname{haf} S_{(A,B)} = \sum_{M \in \mathcal{M}_{(A,B)}} \Phi(M),$$

where we note that the computation is done modulo 2^{k+1}. Since $g(A,B)$ is a unique minimizer of $d(M)$ in $\mathcal{M}_{(A,B)}$ by Lemma 9, this shows that

$$\Phi(g(A,B)) = \operatorname{haf} S_{(A,B)} - \sum_{\substack{M \in \mathcal{M}_{(A,B)} \\ d(M) > d(g(A,B))}} \Phi(M) \tag{4}$$

for a good partition (A,B) of T. By using this equation, for every $M \in \mathcal{M}$, we can compute $\Phi(M)$ in the decreasing order of $d(M)$ as follows. Note that the number of pairing is bounded by a fixed constant as k is fixed, and hence $\mathcal{M}_{(A,B)}$ can be enumerated in a brute-force way.

First, suppose that $M \in \mathcal{M}$ is a maximizer of $d(M)$. Since g is a bijection, there exists a good partition (A,B) of T with $g(A,B) = M$. Then, (4) implies $\Phi(g(A,B)) = \operatorname{haf} S_{(A,B)}$, and hence this value can be computed by Theorem 3. Recall again that the computation is done modulo 2^{k+1}.

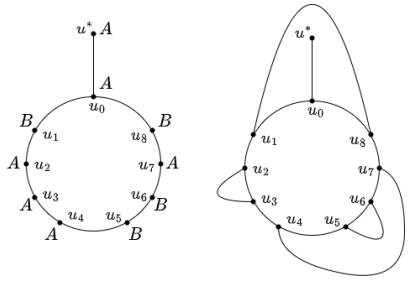


Figure 3 Partition (A_1, B_1) and M_1.

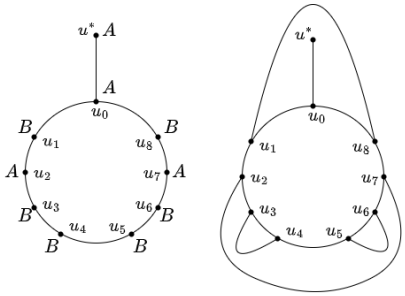


Figure 4 Partition (A_2, B_2) and M_2.

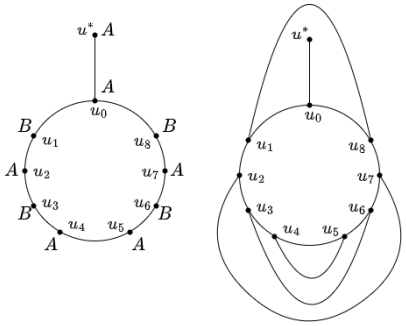


Figure 5 Partition (A_3, B_3) and M_3.

Next, let $M \in \mathcal{M}$ be a feasible pairing and suppose that $\Phi(M')$ is computed for every $M' \in \mathcal{M}$ with $d(M') > d(M)$. Let (A, B) be the good parition of T with $g(A, B) = M$. Since haf $S_{(A,B)}$ can be computed by Theorem 3, we obtain $\Phi(M)$ by using (4).

By applying this procedure repeatedly, we obtain $\Phi(M)$ for every $M \in \mathcal{M}$. In particular, we obtain $\Phi(M^*)$, where M^* is the input pairing. Since $|\mathcal{M}| = c_{k-1}$ is a constant for fixed k and each step runs in polynomial time by Theorems 2 and 3, this algorithm runs in polynomial time.

▶ **Example 10.** Let $T = \{u^*, u_0, u_1, \ldots, u_8\}$, and suppose that

$$M_1 = \{(u^*, u_0), (u_1, u_8), (u_2, u_3), (u_4, u_7), (u_5, u_6)\},$$
$$M_2 = \{(u^*, u_0), (u_1, u_8), (u_2, u_7), (u_3, u_4), (u_5, u_6)\},$$
$$M_3 = \{(u^*, u_0), (u_1, u_8), (u_2, u_7), (u_3, u_6), (u_4, u_5)\}.$$

For $i = 1, 2, 3$, the partition (A, B) corresponding to M_i, which we denote (A_i, B_i), is as shown in Figures 3–5, respectively. Since

$$\text{haf } S_{(A_1,B_1)} = \Phi(M_1) + \Phi(M_2),$$
$$\text{haf } S_{(A_2,B_2)} = \Phi(M_2) + \Phi(M_3),$$
$$\text{haf } S_{(A_3,B_3)} = \Phi(M_3),$$

we can compute $\Phi(M_3)$, $\Phi(M_2)$, and $\Phi(M_1)$ in this order.

By using this procedure, we can obtain a polynomial-time algorithm if the given instance has at most one unique optimal solution.

▶ **Lemma 11.** One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal *can be solved in polynomial time under the assumption that a given instance has a unique optimal solution or has no feasible solution.*

Proof. If the given instance has a unique optimal solution, then the lowest degree term of $\Phi(M^*)$ is $2^k x^{\mathsf{opt}}$, where opt is the optimal value. If the instance has no feasible solution, then $\Phi(M^*) = 0$, i.e., the optimal value is $\mathsf{opt} = +\infty$. Since $\Phi(M^*)$ can be computed in polynomial time by the above argument, we obtain opt.

We now describe how to obtain an optimal solution $\mathcal{P}^*$ when $\mathsf{opt} < +\infty$. For $e \in E$, we remove e and compute the optimal value opt_e in the obtained instance by using the same algorithm as above. Then, e is contained in $\mathcal{P}^*$ if and only if $\mathsf{opt}_e \neq \mathsf{opt}$. This shows that we can determine whether e is contained in $\mathcal{P}^*$ or not in polynomial time. By applying this procedure for every $e \in E$, we obtain $\mathcal{P}^*$. ◀

When we do not assume the uniqueness of the optimal solutions, we perturb the length of edges so that the instance has a unique solution. The following lemma is derived from the *Isolation Lemma* [23], and the same argument is used in [5, 15].

▶ **Lemma 12** (See [5, 15, 23].)**.** *Let $\mathcal{F}$ be a non-empty family of subsets of E with $|E| = m$ such that $|F| \leq n$ for every $F \in \mathcal{F}$. If we assign for each $e \in E$, $w(e) \in \{2mn, 2mn+1, \ldots, 2mn+2m-1\}$ uniformly at random, then with probability greater than $1/2$, there exists a unique set $F^* \in \mathcal{F}$ with the minimum weight $\sum_{e \in F^*} w(e)$. Furthermore, $|F^*| = \min_{F \in \mathcal{F}} |F|$.*

Proof of Theorem 1. Let $G = (V, E)$ be the input graph, where $|V| = n$ and $|E| = m$. For each edge $e \in E$, choose $w(e) \in \{2mn, 2mn+1, \ldots, 2mn+2m-1\}$ uniformly at random, and replace e with a path of length $w(e)$. By applying Lemma 12 in which $\mathcal{F}$ consists of all edge sets of feasible solutions of the problem, we see that the obtained instance has a unique optimal solution with probability greater than $1/2$ (unless the original instance is infeasible). Therefore, by applying Lemma 11 to the obtained instance, we can solve the problem in polynomial time. ◀

4 Proof of Lemma 7

In this section, we give a proof of Lemma 7. To this end, we characterize when a feasible (A, B)-compatible pairing including (u^*, u_0) exists.

▶ **Lemma 13.** *Let (A, B) be a partition of T with $u^*, u_0 \in A$. There is a feasible (A, B)-compatible pairing M with $(u^*, u_0) \in M$ if and only if*

$$\frac{\#A - 2}{2} = \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\} = \#\{j \in [2k-2] \mid j \text{ is even}, u_j \in A\}. \quad (5)$$

Proof. We observe that if one equality in (5) holds, then the other equality also holds, because $\#A - 2 = \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\} + \#\{j \in [2k-2] \mid j \text{ is even}, u_j \in A\}$.

We first show the necessity ("only if" part). Suppose that there is a feasible (A, B)-compatible pairing M with $(u^*, u_0) \in M$. For any pair $(u_i, u_{i'}) \in M \setminus \{(u^*, u_0)\}$, the parities of i and i' are different as M is feasible. Thus, we obtain $\#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\} = \#\{j \in [2k-2] \mid j \text{ is even}, u_j \in A\}$, and hence (5) holds.

We next show the sufficiency ("if" part) by induction on $|T|$. Suppose that (5) holds. A desired pairing obviously exists when $k = 1$, and so suppose that $k \geq 2$. Then, there exists an index $j \in [2k-3]$ such that both u_j and u_{j+1} belong to the same set, either A or B, since otherwise terminals in A and B appear alternately along the boundary of F, which

contradicts (5). Since removing u_j and u_{j+1} does not affect the parities of the indices of the other terminals, (5) holds after removing u_j and u_{j+1}. By the induction hypothesis, we obtain a feasible pairing M' of $T \setminus \{u_j, u_{j+1}\}$. Then, $M = M' \cup \{(u_j, u_{j+1})\}$ is a desired pairing, which shows the sufficiency. ◀

Even when $(u^*, u_0) \notin M$, we can obtain a similar characterization by shifting the indices in Lemma 13 as follows. Recall that $I = \{0, 1, \ldots, 2k-2\} = [2k-2] \cup \{0\}$.

▶ **Lemma 14.** *Let $i \in I$ and let (A, B) be a partition of T with $u^*, u_i \in A$. There is a feasible (A, B)-compatible pairing M with $(u^*, u_i) \in M$ if and only if*

$$\frac{\#A - 2}{2} = \#\{j \in I \mid j < i,\ j \text{ is even},\ u_j \in A\} + \#\{j \in I \mid j > i,\ j \text{ is odd},\ u_j \in A\}. \quad (6)$$

Proof. Let $v_0 = u_i$ and define

$$v_j = \begin{cases} u_{i+j} & \text{if } j \leq 2k-2-i, \\ u_{i+j-2k+1} & \text{if } j \geq 2k-1-i \end{cases}$$

for $j \in [2k-2]$. That is, we relabel the terminals in $T \setminus \{u^*\}$ so that $v_0, v_1, v_2, \ldots, v_{2k-2}$ appear counter-clockwise in this order and $v_0 = u_i$. Since the right-hand side of (6) is equal to either $\#\{j \in [2k-2] \mid j \text{ is odd}, v_j \in A\}$ or $\#\{j \in [2k-2] \mid j \text{ is even}, v_j \in A\}$, we obtain the lemma by Lemma 13. ◀

We are now ready to prove Lemma 7, which we restate here.

▶ **Lemma 7.** *Let (A, B) be a partition of T with $u^*, u_0 \in A$. Then, (A, B) is good if and only if $(f_{(A,B)}(1), \ldots, f_{(A,B)}(2k-2))$ is a $(k-1)$-ballot sequence.*

Proof. We first show the necessity ("only if" part). Suppose that (A, B) is a good partition of T. Since there exists a feasible (A, B)-compatible pairing M with $(u^*, u_0) \in M$, by Lemma 13, we obtain (5). By the definition of $f_{(A,B)}$, we obtain

$$\begin{aligned} &\sum_{j=1}^{2k-2} f_{(A,B)}(j) \\ &= \#\{j \in [2k-2] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\} \\ &\quad - \#\{j \in [2k-2] \mid j \text{ is even}, u_j \in B\} + \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in B\} \quad (7) \\ &= 2(\#\{j \in [2k-2] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\}) \\ &= 0, \end{aligned}$$

where the last equality follows from (5).

To derive a contradiction, assume that $(f_{(A,B)}(1), \ldots, f_{(A,B)}(2k-2))$ is not a $(k-1)$-ballot sequence. Then, there exists $i \in [2k-2]$ such that $\sum_{j=1}^{i} f_{(A,B)}(j) < 0$. Among such i, we choose the minimum one. By the minimality of i, we obtain $\sum_{j=1}^{i-1} f_{(A,B)}(j) = 0$ and $f_{(A,B)}(i) = -1$. Since $i - 1 \equiv \sum_{j=1}^{i-1} f_{(A,B)}(j) = 0 \pmod 2$, we see that i is odd, and hence $u_i \in A$. By a similar calculation as (7), we obtain

$$0 = \sum_{j=1}^{i-1} f_{(A,B)}(j) = 2(\#\{j \in [i-1] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [i-1] \mid j \text{ is odd}, u_j \in A\}),$$

which shows that $\#\{j \in [i-1] \mid j \text{ is even}, u_j \in A\} = \#\{j \in [i-1] \mid j \text{ is odd}, u_j \in A\}$. This together with (5) shows that

$$\begin{aligned}\frac{\#A-2}{2} &= \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\} \\ &= \#\{j \in [i-1] \mid j \text{ is odd}, u_j \in A\} + 1 + \#\{j \in I \mid j > i, j \text{ is odd}, u_j \in A\} \\ &= \#\{j \in [i-1] \mid j \text{ is even}, u_j \in A\} + 1 + \#\{j \in I \mid j > i, j \text{ is odd}, u_j \in A\} \\ &= \#\{j \in I \mid j < i, j \text{ is even}, u_j \in A\} + \#\{j \in I \mid j > i, j \text{ is odd}, u_j \in A\},\end{aligned}$$

where we note that $u_0, u_i \in A$ and $\{j \in I \mid j < i\} = [i-1] \cup \{0\}$. By Lemma 14, this shows that there exists a feasible (A, B)-compatible pairing M with $(u^*, u_i) \in M$, which contradicts that (A, B) is a good partition.

We next show the sufficiency ("if" part). Suppose that $(f_{(A,B)}(1), \ldots, f_{(A,B)}(2k-2))$ is a $(k-1)$-ballot sequence. Since $\sum_{j=1}^{2k-2} f_{(A,B)}(j) = 0$, by the same calculation as (7), we obtain $\#\{j \in [2k-2] \mid j \text{ is even}, u_j \in A\} = \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\}$, and hence (5) holds. Then, there exists a feasible (A, B)-compatible pairing M with $(u^*, u_0) \in M$ by Lemma 13.

To derive a contradiction, assume that (A, B) is not a good partition of T. Then, there exists a feasible (A, B)-compatible pairing M with $(u^*, u_i) \in M$ for some $i \in [2k-2]$. By Lemma 14, this means that

$$\frac{\#A-2}{2} = \#\{j \in I \mid j < i, j \text{ is even}, u_j \in A\} + \#\{j \in I \mid j > i, j \text{ is odd}, u_j \in A\}. \quad (8)$$

Since

$$\begin{aligned}\frac{\#A-2}{2} &= \#\{j \in [2k-2] \mid j \text{ is odd}, u_j \in A\} \\ &= \#\{j \in I \mid j \le i, j \text{ is odd}, u_j \in A\} + \#\{j \in I \mid j > i, j \text{ is odd}, u_j \in A\}\end{aligned}$$

by (5), this together with (8) shows that

$$\#\{j \in I \mid j < i, j \text{ is even}, u_j \in A\} = \#\{j \in I \mid j \le i, j \text{ is odd}, u_j \in A\}. \quad (9)$$

If i is odd, then

$$\begin{aligned}&\sum_{j=1}^{i} f_{(A,B)}(j) \\ &= \#\{j \in [i] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [i] \mid j \text{ is odd}, u_j \in A\} \\ &\quad - \#\{j \in [i] \mid j \text{ is even}, u_j \in B\} + \#\{j \in [i] \mid j \text{ is odd}, u_j \in B\} \\ &= \#\{j \in [i] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [i] \mid j \text{ is odd}, u_j \in A\} \\ &\quad - \Big(\frac{i-1}{2} - \#\{j \in [i] \mid j \text{ is even}, u_j \in A\}\Big) + \Big(\frac{i+1}{2} - \#\{j \in [i] \mid j \text{ is odd}, u_j \in A\}\Big) \\ &= 2(\#\{j \in [i] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [i] \mid j \text{ is odd}, u_j \in A\}) + 1 \\ &= 2(\#\{j \in I \mid j < i, j \text{ is even}, u_j \in A\} - 1 - \#\{j \in I \mid j \le i, j \text{ is odd}, u_j \in A\}) + 1 \\ &= -1,\end{aligned}$$

where the fourth equality follows from $u_0 \in A$ and the last equality is by (9). This contradicts that $f_{(A,B)}(j)$ is a ballot sequence.

Similarly, if i is even, then

$$\begin{aligned}
&\sum_{j=1}^{i-1} f_{(A,B)}(j) \\
&= \#\{j \in [i-1] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [i-1] \mid j \text{ is odd}, u_j \in A\} \\
&\quad - \#\{j \in [i-1] \mid j \text{ is even}, u_j \in B\} + \#\{j \in [i-1] \mid j \text{ is odd}, u_j \in B\} \\
&= 2(\#\{j \in [i-1] \mid j \text{ is even}, u_j \in A\} - \#\{j \in [i-1] \mid j \text{ is odd}, u_j \in A\}) + 1 \\
&= 2(\#\{j \in I \mid j < i, j \text{ is even}, u_j \in A\} - 1 - \#\{j \in I \mid j \leq i, j \text{ is odd}, u_j \in A\}) + 1 \\
&= -1,
\end{aligned}$$

which is a contradiction. ◀

5 Conclusion

We introduced One-Face Shortest k-Disjoint Paths Problem with a Deviation Terminal and gave a first randomized polynomial-time algorithm. In our algorithm, we combined the arguments by Datta et al. [9] and Hirai and Namba [15] with new insights on combinatorial properties of the problem.

It is natural to ask whether our results can be extended to the case when all the terminals except two or more are on the same face, which is still open. Another interesting unsolved case is when the terminals are on two faces in an arbitrary way.

Note that our algorithm can be derandomized if the input planar graph is cubic, because we can adopt a deterministic algorithm of Björklund and Husfeldt [4] for disjoint $(A+B)$-paths problem instead of the randomized one of Hirai and Namba [15]. It is open whether there exists a deterministic algorithm for non-cubic graphs.

References

1. Isolde Adler, Stavros G Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Irrelevant vertices for the planar disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 122:815–843, 2017.
2. Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a geometric lens to find k disjoint shortest paths. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 26:1–26:14, 2021.
3. Kristóf Bérczi and Yusuke Kobayashi. The directed disjoint shortest paths problem. In *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 122, pages 13:1–13:13, 2017.
4. Andreas Björklund and Thore Husfeldt. Counting shortest two disjoint paths in cubic planar graphs with an NC algorithm. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, pages 1–19, 2018.
5. Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, 2019.
6. Glencora Borradaile, Amir Nayyeri, and Farzad Zafarani. Towards single face shortest vertex-disjoint paths in undirected planar graphs. In *23rd Annual European Symposium on Algorithms (ESA 2015)*, pages 227–238. Springer, 2015.
7. Éric Colin de Verdière and Alexander Schrijver. Shortest vertex-disjoint two-face paths in planar graphs. *ACM Transactions on Algorithms*, 7(2):1–12, 2011.
8. Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 197–206, 2013.

9 Samir Datta, Siddharth Iyer, Raghav Kulkarni, and Anish Mukherjee. Shortest k-disjoint paths via determinants. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122, pages 19:1–19:21, 2018.

10 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998.

11 Jeff Erickson and Yipu Wang. Four shortest vertex-disjoint paths in planar graphs. preprint on webpage, October 2017. URL: `http://jeffe.cs.illinois.edu/pubs/pdf/disjoint.pdf`.

12 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

13 András Frank. Packing paths, circuits, and cuts — a survey. In *Paths, flows, and VLSI-layout*, pages 47–100. Springer, 1990.

14 Marinus Gottschau, Marcus Kaiser, and Clara Waldmann. The undirected two disjoint shortest paths problem. *Operations Research Letters*, 47(1):70–75, 2019.

15 Hiroshi Hirai and Hiroyuki Namba. Shortest $(A + B)$-path packing via hafnian. *Algorithmica*, 80(8):2478–2491, 2018.

16 Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

17 Yusuke Kobayashi and Ryo Sako. Two disjoint shortest paths problem with non-negative edge length. *Operations Research Letters*, 47(1):66–69, 2019.

18 Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234–245, 2010.

19 Mark R Kramer. The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits. *Advances in Computing Research*, 2:129–146, 1984.

20 James R. Lee, Manor Mendel, and Mohammad Moharrami. A node-capacitated Okamura-Seymour theorem. *Mathematical Programming*, 153(2):381–415, 2015.

21 Willian Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 169–178, 2021.

22 James F Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5(3):31–36, 1975.

23 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th Annual ACM symposium on Theory of Computing (STOC 1987)*, pages 345–354, 1987.

24 Richard G Ogier, Vladislav Rutenburg, and Nachum Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *IEEE Transactions on Information Theory*, 39(2):443–455, 1993.

25 Haruko Okamura. Multicommodity flows in graphs. *Discrete Applied Mathematics*, 6(1):55–62, 1983.

26 Haruko Okamura and Paul D Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31(1):75–81, 1981.

27 Neil Robertson and Paul D Seymour. Graph minors. VI. Disjoint paths across a disc. *Journal of Combinatorial Theory, Series B*, 41(1):115–138, 1986.

28 Neil Robertson and Paul D Seymour. Graph minors. VII. Disjoint paths on a surface. *Journal of Combinatorial Theory, Series B*, 45(2):212–254, 1988.

29 Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

30 Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994.

31 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

32 Paul D. Seymour. Disjoint paths in graphs. *Discrete Mathematics*, 29:293–309, 1980.

33 Yossi Shiloach. A polynomial solution to the undirected two paths problem. *Journal of the ACM*, 27(3):445–456, 1980.

34 Anand Srinivas and Eytan Modiano. Finding minimum energy disjoint paths in wireless ad-hoc networks. *Wireless Networks*, 11(4):401–417, 2005.

35 Richard P Stanley. *Catalan Numbers*. Cambridge University Press, 2015.

36 Carsten Thomassen. 2-linked graphs. *European Journal of Combinatorics*, 1:371–378, 1980.

Optimizing Quantum Circuit Parameters via SDP

Eunou Lee ✉
Sunkyunkwan University, Seoul, South Korea

Abstract

In recent years, parameterized quantum circuits have become a major tool to design quantum algorithms for optimization problems. The challenge in fully taking advantage of a given family of parameterized circuits lies in finding a good set of parameters in a non-convex landscape that can grow exponentially to the number of parameters.

We introduce a new framework for optimizing parameterized quantum circuits: round SDP solutions to circuit parameters. Within this framework, we propose an algorithm that produces approximate solutions for a quantum optimization problem called Quantum Max Cut. The rounding algorithm runs in polynomial time to the number of parameters regardless of the underlying interaction graph.

The resulting 0.562-approximation algorithm for generic instances of Quantum Max Cut improves on the previously known best algorithms by Anshu, Gosset, and Morenz with a ratio 0.531 and by Parekh and Thompson with a ratio 0.533.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Quantum algorithm, Optimization, Rounding algorithm, Quantum Circuit, Approximation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.48

Related Version *Full Version*: `https://arxiv.org/pdf/2209.00789.pdf`

Acknowledgements This work was supported by the Postdoctoral Research Program of Sungkyunkwan University (2021).

1 Introduction

Since Farhi, Goldstone, and Gutmann proposed the Quantum Approximate Optimization Algorithm (QAOA) [7], parameterized quantum circuits have become a major tool to design quantum algorithms for optimization problems. For example, the growing area of Quantum Variational Algorithms (VQA), where parameterized quantum circuits are optimized in a variational manner, is considered a leading candidate to deliver quantum advantage over classical computation using the current quantum machines [6].

A parameterized quantum circuit is a quantum circuit whose components are determined by given circuit parameters $\vec{\theta}$ and output a state $|\phi(\vec{\theta})\rangle$ in the end. Given a parameterized circuit, the challenge in fully taking advantage of it lies in finding a good set of parameters in a non-convex landscape that can grow exponentially to the number of parameters. VQAs attempt to find locally optimal parameters, without guarantees on how good the output states are for the given optimization problem. Other parameterized quantum circuit algorithms provide rigorous guarantees on the quality of their output states, often in the form of approximation ratios. To make the search for good parameters more tractable, existing rigorous algorithms optimize the parameters over a small portion of the entire parameter space, by assuming the parameters to be equal on each circuit layer or on the entire circuit [3, 8, 9, 4, 5, 10, 12, 1, 16, 18, 19, 2, 13]. As another limitation, the existing rigorous algorithms only work on specific instances such as those with regular interaction graphs or with bounded light-cone sizes.

In this paper, we introduce a novel way to optimize circuit parameters rigorously and efficiently without reducing the parameter space or limiting the algorithm to special case instances. We do so by applying the ideas of the seminal Goemans-Williamson algorithm [11];

for each edge of the interaction graph, we round an SDP solution vector attached to it to a circuit parameter on the edge. In this way, polynomially many circuit parameters can be efficiently and rigorously optimized. The parameterized quantum circuit we optimize in this study was initially proposed by Anshu, Gosset, and Morenz [1] to approximate degree-bounded regular instances of Quantum Max Cut. We obtain a 0.562-approximation algorithm for generic instances of Quantum Max Cut, improving on the previously known best approximation algorithms for the problem, which give approximation ratios of less than 0.54 [1, 18].

Quantum Max Cut is a QMA-hard quantum optimization problem that has recently been studied in the context of approximation [10, 1, 18, 19, 14, 20]. The problem asks one to find the largest eigenvalue $\lambda_{max}(H)$ of a 2-local Hamiltonian defined by the local term $H_{ij} = (I - X_iX_j - Y_iY_j - Z_iZ_j)/4 = (|01\rangle - |10\rangle)_{ij}(\langle 01| - \langle 10|)_{ij}/2$. Note that the local term is a projector onto the singlet state, giving it the name Quantum Max Cut. More formally, we give the following definition.

▶ **Definition 1** (Pauli matrices). *We use the standard definition of the Pauli matrices:*

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},\ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

An indexed Pauli matrix works on a specific qubit in a multi-qubit system. Formally, $P_i := I \otimes \cdots \otimes I \otimes P \otimes I \otimes \ldots I$, where the matrix $P \in \{X, Y, Z\}$ is in the i-th place.

▶ **Definition 2** (Quantum Max Cut). *Given a weighted graph $G = (V, E, W)$, find the maximum eigenvalue of the following Hamiltonian:*

$$H = \sum_{(i,j)\in E} w_{ij} H_{ij}, \tag{H}$$

where $H_{ij} = \frac{1}{4}(I - X_iX_j - Y_iY_j - Z_iZ_j)$, and $w_{ij} \geq 0$ for all $(i, j) \in E$. We denote the maximum eigenvalue as $OPT := \lambda_{max}(H)$.

Because Quantum Max Cut is QMA-hard [21], the exact optimum is not expected to be found efficiently. As a natural reaction to this hardness, multiplicative approximation algorithms have been studied for the problem. An α-approximation algorithm returns a state $|\phi\rangle$ with its objective value $\langle\phi|H|\phi\rangle$ guaranteed to be high relative to the optimum, satisfying $\langle\phi|H|\phi\rangle \geq \alpha\lambda_{max}(H)$ for a constant $\alpha \in [0, 1]$.

Given that the Goemans-Williamson algorithm [11] approximates Max Cut optimally up to the unique games conjecture [15] and that the evaluation of 2-local Hamiltonians and the cut of graphs share the similarity of being quadratic, it is natural to adopt the Goemans-Williamson algorithm in some form to approximately optimize Quantum Max Cut. This path was indeed followed by all existing approximation algorithms for Quantum Max Cut [10, 1, 18, 19, 14, 20] and for the Max 2-local Hamiltonian with positive-semidefinite (PSD) terms [12, 19, 20], which contains Quantum Max Cut as a sub-problem.

To apply the ideas from the Goemans-Williamson algorithm [11] to Quantum Max Cut, two things are required: an SDP that represents the optimization problem, and an algorithm to round the SDP solution to a quantum state.

In the case of the Quantum Max Cut and Max 2-local Hamiltonian, a natural set of SDPs called the quantum Lasserre hierarchy [17] has been used by existing algorithms. Roughly speaking, a level-k SDP in the hierarchy forces its feasible solutions to be consistent with quantum states up to k-qubit correlation. Because the Lasserre SDPs of a higher level are

closer to the physical reality, they may yield better results for a given optimization problem. Some algorithms [10, 12, 1, 19] employ level-1 SDPs while others employ [18, 20] level-2 SDPs. The recent Unique Games-hardness result [13] of 0.956-approximation for Quantum Max Cut was also proved using a level-1 Lasserre SDP. The algorithms using the level-2 SDPs [18, 20] demonstrate that approximation ratios are improved compared to the level-1 SDP algorithms. Especially, [18] showed that a monogamy of entanglement relation can be derived from the SDP solutions, which we also use in this paper. The SDP we use in this paper is of level-2, inspired by and resembling the SDPs in [18, 20].

We now give a formal definition of the quantum Lasserre hierarchy.

▶ **Definition 3** ($\mathcal{P}_n(k)$)**.** *Given a natural number k, we define the set $\mathcal{P}_n(k)$ as the set of Pauli matrices with a Hamming weight less than or equal to k on n qubits:*

$$\mathcal{P}_n(k) = \{\prod_{i\in A} P_i | A \subset [n], |A| \leq k, P_i \in \{X_i, Y_i, Z_i\}\}.$$

▶ **Definition 4** (Canonical Lasserre)**.** *Given a weighted graph $G = (V, E, W)$, we define the level-k Lasserre $L_k(G)$ for Quantum Max Cut on the graph G as the following SDP:*

$$\textit{Maximize} \sum_{(i,j)\in E} \frac{1}{4} w_{ij}(M(I,I) - M(X_iX_j, I) - M(Z_iZ_j, I) - M(Z_iZ_j, I)) \tag{L_k}$$

subject to:

$$\begin{aligned}
&M(\Phi, \Phi) = 1, && \forall\ \Phi \in \mathcal{P}_n(k)\\
&M(\Phi, \Psi) = 0, && \forall\ \Phi, \Psi \in \mathcal{P}_n(k)\ s.t.\ \Phi\Psi + \Psi\Phi = 0\\
&M(\Phi_1, \Psi_1) = M(\Phi_2, \Psi_2), && \forall\ \Phi_1, \Psi_1, \Phi_2, \Psi_2 \in \mathcal{P}_n(k)\ s.t.\ \Phi_1\Psi_1 = \Phi_2\Psi_2\\
&M(\Phi_1, \Psi_1) = -M(\Phi_2, \Psi_2), && \forall\ \Phi_1, \Psi_1, \Phi_2, \Psi_2 \in \mathcal{P}_n(k)\ s.t.\ \Phi_1\Psi_1 = -\Phi_2\Psi_2\\
&M \succeq 0,\\
&M \in \mathbb{R}^{|\mathcal{P}_n(k)|\times|\mathcal{P}_n(k)|}.
\end{aligned}$$

▶ **Lemma 5.** *The level-k Lasserre $L_k(G)$ is a relaxation of Quantum Max Cut for any $k \in [n]$. Meaning, for any $k \in [n]$ and an arbitrary n-qubit state $|\phi\rangle$, there exists a matrix M that satisfies the constraints of $L_k(G)$ and its objective value for $L_k(G)$ is at least $\langle\phi|H|\phi\rangle$.*

Proof. See Appendix A.1. ◀

The other component of the Goemans-Williamson scheme is a rounding algorithm that maps SDP solutions to valid quantum states. The existing rounding algorithms for Quantum Max Cut and Max 2-local Hamiltonian round to product states of 1- or 2-qubit states, with the exception of [1]. Anshu, Gosset, and Morenz [1] devised a shallow parameterized quantum circuit to generate high energy states when given instances with low-degree regular interaction graphs. Our rounding algorithm also makes use of the circuit proposed in [1], but we improve it by allowing parameters on each edge to be different from each other and to be applied to generic instances. Because there are much more degrees of freedom in our circuits than in [1], it is more challenging to optimize the circuits.

The circuit is comprised of mutually commuting gates $\exp[i\theta_{ij}P(i)P(j)]$ on each edge $(i, j) \in E$, where $P(i)$ is a Pauli matrix on the qubit i, and θ_{ij} is a real parameter. The gates together implement a unitary $U(\vec{\theta}) = \Pi_{(i,j)\in E} \exp[i\theta_{ij}P(i)P(j)]$. Our algorithm first

randomly rounds the level-1 SDP solution vectors to a random bit string $|z\rangle$. Then the bit string is evolved by the unitary $U(\vec{\theta})$, with the circuit parameters $\vec{\theta}$ determined by the level-2 SDP solution vectors.

To illustrate how the circuit works, suppose an initial state $|z\rangle_{i,j} = |01\rangle$ is assigned on the qubits i, j. Our algorithm assigns $P(i) = Y_i, P(j) = X_j$ based on the values assigned to z_i and z_j. The gate on (i, j) will evolve the bit string to the state

$$\exp[i\theta_{ij} Y_i X_j]|01\rangle = \cos\theta_{ij}|01\rangle - \sin\theta_{ij}|10\rangle.$$

The state yields the maximum energy 1 on the edge if $\theta_{ij} = \pi/2$, and energy 1/2 if $\theta = \pi/2$. Thus, we assign a higher value to θ_{ij} if the SDP vector on (i, j) suggests that the state should be more entangled on the edge. If any of the vertices i, j is connected to edges than other (i, j), the energy on (i, j) will be affected by the parameters on the neighboring edges as well. This quantum interference poses a great challenge to the design of the algorithm, and we circumvent it by utilizing the monogamy of entanglement on each vertex.

The algorithm we propose approximates Quantum Max Cut with the ratio 0.562, outperforming existing algorithms. More important are the novel techniques we utilize to optimize quantum circuits by consulting SDP vectors. It is noteworthy that our algorithm outperforms the parameterized circuit algorithm in [1] even though ours works on more general instances. This perhaps illustrates the power of circuit optimization based on SDP rounding. We believe, however, that our algorithm is not optimal in terms of approximation ratio. According to Brandão and Harrow [4] and Parekh and Thompson [20], an optimal algorithm should be able to generate arbitrary product states, but our algorithm cannot. More specifically to this problem, the Hamiltonian is symmetric under a unitary $U \otimes U \otimes \cdots \otimes U$ for any 1 qubit unitary U, but the algorithm is not symmetric under the same transformation.

In conclusion, the algorithm lies in a narrow intersection of two important classes of algorithms: parameterized quantum circuit algorithms and SDP rounding algorithms. The connection between the two areas provides two different ways to view our algorithm: it is the first quantum SDP rounding algorithm that works on generic Quantum Max Cut instances, and it is the first algorithm that optimize circuit parameters via SDP rounding.

Many natural questions arise. Can we optimize QAOA circuits using SDP? Can SDPs help optimize circuits for practical problems such as the quantum chemistry problems? Can we get a better rounding circuit by optimizing the generalized version [2] of the circuit we used? Will level-3 SDPs be helpful for 2-local Hamiltonian problems? Is the approximation ratio upper bound given in [13] optimal? Can we optimize non-commuting circuits using SDPs?

2 Our SDP

In this section, we define a customized level-2 Lasserre SDP for the sake of better notation. An equivalent way to describe SDP (L_k) is to use a set of vectors as variables of an optimization program instead of a PSD matrix itself. This is because given a PSD matrix, one can find a set of vectors of which the Gram matrix is the given PSD matrix (e.g. via the Cholesky decomposition), and conversely given a set of vectors, their Gram matrix is always PSD.

We choose to represent the SDP in vectors because the geometric meaning is more accessible in this way. Furthermore, for indexing reasons, we only leave the necessary constraints of the previous SDP (L_k), regarding the $I, X_i, Y_i, Z_i, X_iX_j, Y_iY_j, Z_iZ_j$ terms.

▶ **Definition 6.** *Given a weighted graph $G = (V, E, W)$, we define SDP (S) for Quantum Max Cut on the graph G as followes:*

$$\textit{Maximize} \sum_{(i,j)\in E} \frac{1}{4} w_{ij}(v_0 - v_{ij,1} - v_{ij,2} - v_{ij,3}) \cdot v_0, \tag{S}$$

subject to:

$$\|v_0\| = 1, \tag{1}$$
$$\|v_{i,a}\| = 1, \tag{2}$$
$$v_{i,a} \cdot v_{i,b} = 0, \tag{3}$$
$$\|v_{ij,a}\| = 1, \tag{4}$$
$$v_{i,a} \cdot v_{j,a} = v_{ij,a} \cdot v_0, \tag{5}$$
$$v_{ij,a} \cdot v_{jk,a} = v_{ik,a} \cdot v_0, \tag{6}$$
$$v_{ij,a} \cdot v_{jk,b} = 0, \tag{7}$$
$$v_{ij,a} \cdot v_{ij,b} = -v_{ij,c} \cdot v_0, \tag{8}$$
$$v_0,\ v_{i,a},\ v_{ij,a} \in \mathbb{R}^N,$$

$\forall$ *distinct* $i,j,k \in V$, $\quad \forall$ *distinct* $a,b,c \in \{1,2,3\}$,
for a sufficiently big $N \in O(|V|^2)$.

We denote the optimal value of the above SDP as OPT_{SDP}.

▶ **Lemma 7.** *The SDP (S) is a relaxation of the level-2 Lasserre* (L_2). *Thus, given an arbitrary matrix* M *that satisfies the constraints of the SDP* (L_2), *there exists a tuple of vectors* $(v_0, (v_{i,a})_{i,a}, (v_{ij,a})_{i,j,a})$ *that satisfies the constraints of (S), and its objective value for (S) is at least the objective value of* M *for* (L_2).

Proof. Let us consider an SDP with the same objective function as SDP (S), and a subset of the constraints restricted to $T := \{I, X_i, Y_i, Z_i, X_iX_j, Y_iY_j, Z_iZ_j | i,j \in V, i \neq j\} \subset \mathcal{P}_n(2)$. Because we now have fewer constraints, we get a relaxed SDP with respect to L_2. The constraints we get by restricting the indices are

$M(I,I) = 1$,
$M(P_i,P_i) = 1$,
$M(P_i,Q_i) = 0$,
$M(P_iP_j, P_iP_j) = 1$,
$M(P_i,P_j) = M(P_iP_j, I)$,
$M(P_iP_j, P_jP_k) = M(P_iP_k, I)$,
$M(P_iP_j, Q_jQ_k) = 0$,
$M(P_iP_j, Q_iQ_j) = -M(R_iR_j, I)$,
$M \in \mathbb{R}^{|\mathcal{P}_n(2)| \times |\mathcal{P}_n(2)|}$,
$M \succeq 0$,
$\forall$ distinct $i,j,k \in V$, $\forall$ distinct $P,Q,R \in \{X,Y,Z\}$.

Since M is a PSD matrix, we can obtain a Cholesky decomposition $(w_s)_{s\in T}$ such that $w_s \cdot w_t = M(s,t)$, where $w_s, w_t \in \mathbb{R}^N$ for all $s,t \in T$, with some natural number N. Now re-index the vectors so that

$$v_0 = w_I,$$
$$v_{i,1} = w_{X_i},\ v_{i,2} = w_{Y_i},\ w_{i,3} = w_{Z_i},$$
$$v_{ij,1} = w_{X_iX_j},\ v_{ij,2} = w_{Y_iY_j},\ v_{ij,3} = w_{Z_iZ_j}$$

for all distinct $i, j \in V$. Writing the above SDP constraints and the objective function in terms of the vectors $v_0, v_{i,a}, v_{ij,a}$ gives the desired SDP (S). ◀

Feasible solutions of the SDP (S) have useful properties that we will use later.

▶ **Lemma 8** (Useful properties of SDP solutions). *Let $(v_0, (v_{i,a})_{i,a}, (v_{ij,a})_{i,j,a})$ be a feasible solution to the SDP (S). Define a vector*

$$v_{ij} := v_{ij,1} + v_{ij,2} + v_{ij,3} \tag{9}$$

on each edge (i, j). Then the following relations hold for all $i, j \in V$:

$$\|v_{ij}\|^2 = 3 - 2v_{ij} \cdot v_0. \tag{10}$$
$$\|v_0 + v_{ij}\|^2 = 4, \tag{11}$$
$$v_{ij} \cdot v_{jk} = v_{ik} \cdot v_0. \tag{12}$$

In particular, $v_0 + v_{ij}$ is on a sphere.

Proof. We derive the equations from the constraints of the SDP (S). For the equation (10), expand the left-hand side to get

$$\begin{aligned}\|v_{ij}\|^2 = \|v_{ij,1} + v_{ij,2} + v_{ij,3}\|^2 &= \sum_{a\in[3]} \|v_{ij,a}\|^2 + 2(v_{ij,1} \cdot v_{ij,2} + v_{ij,2} \cdot v_{ij,3} + v_{ij,3} \cdot v_{ij,1}) \\ &= 3 - 2(v_{ij,1} + v_{ij,2} + v_{ij,3}) \cdot v_0 \\ &= 3 - 2v_{ij} \cdot v_0,\end{aligned}$$

where the second line is from the SDP constraints (4) and (8). We get the equation (11) using the equation (10):

$$\|v_0 + v_{ij}\|^2 = v_0 \cdot v_0 + 2v_0 \cdot v_{ij} + v_{ij} \cdot v_{ij} = 1 + 2v_{ij} \cdot v_0 + 3 - 2v_{ij} \cdot v_0 = 4.$$

To get the equation (12),

$$v_{ij} \cdot v_{jk} = \sum_{a\in[3]} v_{ij,a} \cdot \sum_{b\in[3]} v_{jk,b} = \sum_{a\in[3]} v_{ij,a} \cdot v_{jk,a} = \sum_{a\in[3]} v_{ik,a} \cdot v_0 = v_{ik} \cdot v_0.$$

Here we used the SDP constraints (6) and (7). ◀

The following lemma is crucial for the design of the algorithm. It says that the total SDP value on d edges with weight 1 that share a common vertex is at most $(d+1)/2$. For context, a uniformly random state has energy $d/4$, and a best bit string would yield energy $d/2$. So the lemma says that the quantum advantage in energy per edge decreases as the degree increases. This lemma is from [18] (and a quantum state version of it from [1]) and we prove it in a more elementary way without the Schur complement argument used in the original proof.

▶ **Lemma 9** (Monogamy of entanglement, [18]). *Given a feasible solution* $(v_0, (v_{i,a})_{i,a}, (v_{ij,a})_{i,j,a})$ *to SDP (S), a vertex* $i \in V$*, and a set* T *of adjacent vertices to the node* i*, the total contributions to the value of SDP (S) from the star graph formed by* i *and* T *is upper bounded by* $(d+1)/2$*, where* d *is the number of vertices in* T*. In other words,*

$$\frac{1}{4}\sum_{j\in T}(v_0 - v_{ij})\cdot v_0 \le \frac{d+1}{2}, \tag{13}$$

where $(i,j) \in E \forall j \in T$ *and as before,* $v_{ij} = v_{ij,1} + v_{ij,2} + v_{ij,3}$.

Proof. Without loss of generality, we can assume that $T = [d] = \{1, 2, \dots, d\}$. Then, the lemma is equivalent to the inequality

$$\sum_{j\in[d]} v_{ij}\cdot v_0 \ge -d - 2.$$

The Gram matrix from the vectors $\{v_0, v_{i1}, v_{i2}, \dots, v_{id}\}$ is PSD, so we get

$$0 \preceq \begin{pmatrix} 1 & v_0\cdot v_{i1} & \cdots & v_0\cdot v_{id} \\ v_0\cdot v_{i1} & \|v_{i1}\|^2 & \cdots & v_{i1}\cdot v_{id} \\ \vdots & \vdots & \ddots & \vdots \\ v_0\cdot v_{id} & v_{i1}\cdot v_{id} & \dots & \|v_{id}\|^2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & v_0\cdot v_{i1} & \cdots & v_0\cdot v_{id} \\ v_0\cdot v_{i1} & 3 - 2v_0\cdot v_{i1} & \cdots & v_0\cdot v_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ v_0\cdot v_{id} & v_0\cdot v_{1d} & \dots & 3 - 2v_0\cdot v_{id} \end{pmatrix},$$

where the second line is from the equations (11) and (12).

Since an average of PSD matrices is a PSD matrix, we average the above matrix over the permutation of the indices $\{1, 2, \dots, d\}$ to get

$$0 \preceq \begin{pmatrix} 1 & s & \cdots & s \\ s & 3-2s & \cdots & t \\ \vdots & \vdots & \ddots & \vdots \\ s & t & \dots & 3-2s \end{pmatrix},$$

where

$$s := \mathop{\mathbb{E}}_{k\in[d]}[v_0\cdot v_{ik}], \quad t := \mathop{\mathbb{E}}_{\substack{k,l\in[d]\\ k\neq l}}[v_0\cdot v_{kl}].$$

We want to lower bound the variable s to prove the lemma. To this end, we multiply by a vector $(x, 1, \dots, 1)$ from the both sides to the above matrix, where $x \in \mathbb{R}$. This gives

$$0 \le x^2 + 2dsx + (3-2s)d + td(d-1).$$

This inequality should hold regardless of the value of x. Therefore,

$$0 \ge (ds)^2 - d(3 - 2s + t(d-1)) = d^2s^2 + 2ds - d(3 + td - t).$$

The right-hand side expression is a concave parabola in s, and the constant term is a non-increasing function of t. This means that increasing t can only widen the range that s could be in. From the property of SDP solutions (10), we know that $-3 \leq s, t \leq 1$, so we set $t = 1$ to lower bound s. By finding the roots of the right-hand side expression, we get $-(d+2)/d \leq s \leq 1$. Therefore, we get $\sum_{j \in T} v_0 \cdot v_{ij} = ds \geq -d - 2$, proving the lemma. ◀

3 Approximation algorithm

In this section, we present a 0.562-approximation algorithm for the quantum analogue of Max Cut defined in Definition 1, and we prove its approximation ratio. In this paper, the value of the arccosine function is always in $[0, \pi]$.

Algorithm 1 0.562-approximation algorithm.

1: Given a weighted graph $G = (V, E, W)$,
2: Solve the SDP (S), and get vectors $(v_0, (v_{i,a})_{i,a}, (v_{ij,a})_{ij,a})$ that optimize the SDP.
3: Generate a random bit string $|z\rangle \in \{0,1\}^n$ using the Goemans-Williamson rounding as follows:
4: Pick a uniformly random $a \in \{1,2,3\}$ and a uniformly random vector $r \in S^{N-1}$.
5: Assign $z_i = (\mathrm{sgn}(v_{i,a} \cdot r) + 1)/2$, for all $i \in V$.
6: Generate a parameterized state $|\phi(\vec{\theta})\rangle$ from the bit string $|z\rangle$ as follows:
7: Compute a vector $v_{ij} := v_{ij,1} + v_{ij,2} + v_{ij,3}$ on each $(i,j) \in E$.
8: Compute a circuit parameter $\theta_{ij} = f(\gamma_{ij})$ on each edge $(i,j) \in E$,
9: where $\gamma_{ij} := -\frac{(v_0+v_{ij})\cdot v_0}{\|v_0+v_{ij}\|\|v_0\|}$, $f(\gamma) := \cos^{-1} \exp[-\alpha_0 \max(\gamma, 0)]/2$, $\alpha_0 := 0.041$.
10: Compose a circuit $U(\vec{\theta}) = \prod_{(i,j)\in E} \exp(i\theta_{ij} P(i)P(j))$, where $P(i) = X_i$ if $z_i = 1$, and $P(i) = Y_i$ if $z_i = 0$.
11: Evolve the bit string $|z\rangle$ on the circuit and get $|\phi(\vec{\theta})\rangle = U(\vec{\theta})|z\rangle$.
12: Output the state $|\phi(\vec{\theta})\rangle$.

The algorithm design is within the Goemans-Williamson framework [11]. We first solve the SDP (S). Our rounding algorithm then maps the SDP solution to a quantum state in two stages. In the first stage, a bit string $|z\rangle \in \{0,1\}^n$ is computed such that on each edge $(i,j) \in E$, the bits z_i and z_j are likely to be assigned different values if the level-1 vectors on the vertices are far from each other. When two different bit values are assigned to the two vertices of an edge, we say the edge is "cut". The probability that an edge (i,j) is cut by $|z\rangle$ is lower bounded in Lemma 10.

In the second stage, the bit string is entangled by a parameterized quantum circuit. The circuit consists of mutually commuting 2-qubit gates $\exp[i\theta_{ij} P(i)P(j)]$ on each edge $(i,j) \in E$, where θ_{ij} is a real parameter, and $P(i) = Y_i$ if $z_i = 0$, and $P(i) = X_i$ if $z_i = 1$. For example, say $|z_i z_j\rangle = |01\rangle$ was assigned to $(i,j) \in E$ in the first stage. The gate would evolve the state to

$$\exp[i\theta_{ij} Y_i X_j]|01\rangle = \cos\theta_{ij}|01\rangle - \sin\theta_{ij}|10\rangle.$$

The energy on the edge is an increasing function of θ_{ij} in $[0, \pi/4]$. Thus, the rounding algorithm will assign a higher value to θ_{ij} if the SDP value on the edge $(v_0 - v_{ij}) \cdot v_0$ is higher.

The analysis becomes more complicated when there are other gates connected to (i,j) to interfere. The monogamy of entanglement relation stated in Lemma 9 implies that the assignment state cannot be too strongly entangled on all edges, meaning the neighboring parameters should be considered as well for optimization on (i,j). We optimize the circuit parameter against the monogamy of entanglement via the function f.

▶ **Lemma 10.** *Suppose an SDP solution $(v_0, (v_{i,a})_{i,a}, (v_{ij,a})_{ij,a})$ and a bit string $|z\rangle$ are as in the statement of Algorithm 1, and a vector v_{ij} is defined by the SDP solution as $v_{ij} = v_{ij,1} + v_{ij,2} + v_{ij,3}$. Let a real number $\gamma_{ij} := -\frac{(v_0+v_{ij})\cdot v_0}{\|v_0+v_{ij}\|\|v_0\|}$ be the normalized inner product of the vectors $v_0 + v_{ij}$ and $-v_0$. Then the probability that an edge $(i,j) \in E$ is cut by the string $|z\rangle$ is lower bounded as*

$$\Pr[z_i \neq z_j] \geq \frac{\alpha_{GW}}{3}(2+\gamma_{ij}), \tag{14}$$

where the coefficient α_{GW} is the Goemans-Williamson constant [11]

$$\alpha_{GW} := \min_{t\in[-1,1]} \frac{1}{\pi}\frac{\arccos t}{1/2 - t/2} = 0.8785\cdots.$$

Proof. See Appendix A.2. ◀

The following lemma deals with the performance of the parameterized circuit $U(\vec{\theta})$ constructed in the algorithm. The parameterized family of circuits was proposed in [1], with the assumption that the parameters are the same across the edges. We modify the analysis of [1] to include the case that we need, the case where different parameters can be assigned to different edges.

▶ **Lemma 11** (Energy contribution from (i,j))**.** *Let a quantum state $|\phi(\vec{\theta})\rangle$ and a bit string $|z\rangle$ be as in the statement of Algorithm 1, and circuit parameters $\theta_{ij} \geq 0$ for all $(i,j) \in E$. Then we can lower bound the expected energy contribution from the edge (i,j) as*

$$\langle\phi(\vec{\theta})|4H_{ij}|\phi(\vec{\theta})\rangle \;\geq\; \begin{cases} 1+\sin(2\theta_{ij})(A_{ij}+B_{ij}) + A_{ij}B_{ij} & \text{if } z_i \neq z_j, \\ 0 & \text{if } z_i = z_j, \end{cases} \tag{15}$$

where

$$A_{ij} := \prod_{k\in N(i)\setminus\{j\}} \cos(2\theta_{ik}), \qquad B_{ij} := \prod_{k\in N(j)\setminus\{i\}} \cos(2\theta_{kj}).$$

and $N(i) := \{j \in V | (i,j) \in E\}$ for all $i \in V$.

Proof. See Appendix A.3. ◀

▶ **Theorem 12** (Main)**.** *Algorithm 1 is a 0.562-approximation algorithm for Quantum Max Cut defined in the Definition 1, with the output state $|\phi(\vec{\theta})\rangle$ having energy $\langle\phi(\vec{\theta})|H|\phi(\vec{\theta})\rangle \geq 0.562\,OPT$.*

Proof. We will show that the expected energy of the random state $|\phi(\vec{\theta})\rangle$ on an arbitrary edge $(i,j) \in E$ is at least the SDP contribution from that edge, $w_{ij}(v_0 - v_{ij})\cdot v_0/4$, times 0.562. Summing over all the edges, this will give us $\mathbb{E}[\langle\phi(\vec{\theta})|H|\phi(\vec{\theta})\rangle] = \sum_{(i,j)\in E}\mathbb{E}[\langle\phi(\vec{\theta})|w_{ij}H_{ij}|\phi(\vec{\theta})\rangle] \geq 0.562\sum_{(i,j)\in E} w_{ij}(v_0 - v_{ij})\cdot v_0/4 = 0.562\mathrm{OPT}_{\mathrm{SDP}} \geq 0.562\mathrm{OPT}$, proving the theorem.

Now, let us focus on a single edge $(i,j) \in E$. The expected energy of the state $|\phi(\vec{\theta})\rangle$ on the edge (i,j) is $\mathbb{E}[\langle\phi(\vec{\theta})|w_{ij}H_{ij}|\phi(\vec{\theta})\rangle]$, whereas the SDP value contribution is $w_{ij}(v_0 - v_{ij})\cdot v_0/4$. Therefore we want to lower bound their ratio

$$\alpha_{ij} := \frac{\mathbb{E}\left[\langle\phi(\vec{\theta})|4H_{ij}|\phi(\vec{\theta})\rangle\right]}{(v_0 - v_{ij}) \cdot v_0}. \tag{16}$$

The edge (i, j) is randomly cut by the bit string $|z\rangle$ in the algorithm statement, and we can lower bound the energy by an expression that depends on whether the edge (i, j) is cut or not. By applying Lemmas 10 and 11, we get

$$\mathbb{E}[\langle\phi(\vec{\theta})|4H_{ij}|\phi(\vec{\theta})\rangle] \geq \Pr[z_i \neq z_j]\big[1 + \sin(2\theta_{ij})(A_{ij} + B_{ij}) + A_{ij}B_{ij}\big] + \Pr[z_i = z_j] \cdot 0 \tag{17}$$

$$\geq \frac{\alpha_{GW}}{3}(2 + \gamma_{ij})\big[1 + \sin(2\theta_{ij})(A_{ij} + B_{ij}) + A_{ij}B_{ij}\big], \tag{18}$$

where the variables $A_{ij}, B_{ij}, \gamma_{ij}$ are as defined in the Lemmas;

$$A_{ij} := \prod_{k \in N(i)\setminus\{j\}} \cos(2\theta_{ik}), \quad B_{ij} := \prod_{k \in N(j)\setminus\{i\}} \cos(2\theta_{kj}), \quad \gamma_{ij} := -\frac{(v_0 + v_{ij}) \cdot v_0}{\|v_0 + v_{ij}\|\|v_0\|}.$$

We have

$$\gamma_{ij} = -\frac{(v_0 + v_{ij}) \cdot v_0}{\|v_0 + v_{ij}\|\|v_0\|} = -\frac{1 + v_{ij} \cdot v_0}{2} \tag{19}$$

by the definition of γ_{ij}, so we get $(v_0 - v_{ij}) \cdot v_0 = 2(1 + \gamma_{ij})$. Thus, we can lower bound the right-hand side of (16) as

$$\alpha_{ij} \geq \frac{\alpha_{GW}}{6}\big[1 + \sin(2\theta_{ij})(A_{ij} + B_{ij}) + A_{ij}B_{ij}\big]\frac{2 + \gamma_{ij}}{1 + \gamma_{ij}}. \tag{20}$$

Case 1: $-1 \leq \gamma_{ij} \leq 0$. The circuit parameter on the edge becomes $\theta_{ij} = \cos^{-1}\exp 0/2 = 0$. Therefore, we get

$$\alpha_{ij} \geq \frac{\alpha_{GW}}{6}\big[1 + A_{ij}B_{ij}\big]\frac{2 + \gamma_{ij}}{1 + \gamma_{ij}}.$$

Note that the value of γ_{ij} does not affect the $A_{ij}B_{ij}$ term, so the right-hand side is a decreasing function in γ_{ij} in the region $\gamma_{ij} \leq 0$. This means that it suffices to look at the case where $\gamma_{ij} = 0$ which is a sub-case of Case 2.

Case 2: $0 \leq \gamma_{ij} \leq 1$. We have $\theta_{ij} = f(\gamma_{ij}) = \cos^{-1}\exp[-\alpha_0 \max(\gamma_{ij}, 0)]/2 \geq 0$. We want to lower bound the terms A_{ij}, B_{ij} to lower bound the expression on the right-hand side of (20). By the definition of the function f, we have

$$\begin{aligned}
A_{ij} &= \prod_{k \in N(i)\setminus\{j\}} \cos(2\theta_{ik}) \\
&= \exp \sum_{k \in N(i)\setminus\{j\}} \log\cos(2\theta_{ik}) \\
&= \exp \sum_{k \in N(i)\setminus\{j\}} \log\cos(2f(\gamma_{ij})) \\
&= \exp \sum_{k \in N(i)\setminus\{j\}} \log\cos\cos^{-1}\exp[-\alpha_0 \max(\gamma_{ik}, 0)] \\
&= \exp \sum_{k \in N(i)\setminus\{j\}} -\alpha_0 \max(\gamma_{ik}, 0) \\
&= \exp\left[-\alpha_0 \sum_{k \in N(i)^+\setminus\{j\}} \gamma_{ik}\right],
\end{aligned}$$

where the set $N(i)^+$ is defined to be $N(i)^+ := \{j|(i,j) \in E,\ \gamma_{ij} > 0\}$. We will use the monogamy inequality (13) to lower bound the above expression. From the equation (19), we have

$$\sum_{k\in N(i)^+} \gamma_{ik} = -\sum_{k\in N(i)^+} \frac{1 + v_{ij}\cdot v_0}{2} = \sum_{k\in N(i)^+} \frac{-2 + (v_0 - v_{ij})\cdot v_0}{2} \tag{21}$$

$$\leq -|N(i)^+| + |N(i)^+| + 1 \tag{22}$$

$$= 1.$$

Therefore we have

$$\begin{aligned} A_{ij} &= \exp\left[-\alpha_0 \sum_{k\in N(i)^+\setminus\{j\}} \gamma_{ik}\right] \\ &= \exp\left[-\alpha_0 \left[\sum_{k\in N(i)^+} \gamma_{ik} - \gamma_{ij}\right]\right] \\ &\geq \exp[-\alpha_0(1-\gamma_{ij})]. \end{aligned} \tag{23}$$

Note that the product term A_{ij} is now lower bounded by a function of a single variable γ_{ij}. We can apply the same analysis to get an inequality

$$B_{ij} \geq \exp[-\alpha_0(1-\gamma_{ij})]. \tag{24}$$

By plugging in the relations (23) and (24) to the RHS of (20), we get

$$\begin{aligned} \alpha_{ij} &\geq \frac{\alpha_{GW}}{6}\left[1 + \sin(2\theta_{ij})(A_{ij} + B_{ij}) + A_{ij}B_{ij}\right]\frac{2+\gamma_{ij}}{1+\gamma_{ij}}. \\ &\geq \frac{\alpha_{GW}}{6}\left[1 + 2\sin(\cos^{-1} e^{-\alpha_0\gamma_{ij}})e^{-\alpha_0(1-\gamma_{ij})} + e^{-2\alpha_0(1-\gamma_{ij})}\right]\frac{2+\gamma_{ij}}{1+\gamma_{ij}} \\ &\geq \frac{\alpha_{GW}}{6}\left[1 + 2\sqrt{1-e^{-2\alpha_0\gamma_{ij}}}e^{-\alpha_0(1-\gamma_{ij})} + e^{-2\alpha_0(1-\gamma_{ij})}\right]\frac{2+\gamma_{ij}}{1+\gamma_{ij}} \\ &\geq \min_{\gamma\in[0,1]} \frac{\alpha_{GW}}{6}\left[1 + 2\sqrt{1-e^{-2\alpha_0\gamma}}e^{-\alpha_0(1-\gamma)} + e^{-2\alpha_0(1-\gamma)}\right]\frac{2+\gamma}{1+\gamma} \\ &= 0.562\cdots. \end{aligned}$$

This proves the theorem. The coefficient α_0 is optimized so that the final ratio is maximized. ◀

We make a few comments on the choice of the function f because it is the most non-trivial part in the algorithm and the idea behind the function might find place in future designs of parameterized quantum circuits.

On the one hand, we have the monogamy of entanglement inequality which upper bounds the total SDP value on an arbitrary star graph. More specifically, it upper bounds the sum of the input parameters $\sum \gamma_{ik}$. On the other hand, we want to lower bound a product of cosines of the output parameters $\prod \cos\theta_{ij}$ on the star graph.

Therefore, we want to somehow translate the linear condition into a multiplicative condition of cosines. The most natural approach is to select a function f such that the addition operator on the constraint side works *homomorphically* on the energy lower bound side. In other words, if we have $\cos(f(x))\cos(f(y)) = \cos(f(x+y))$, then we can lower bound the right hand side by the monogamy inequality. To this end, we set $f(x) = \cos^{-1} e^{-ax}$ for a non-negative constant a.

To define the function in the negative domain, an important observation is that having a negative θ_{ij} is never better than having $\theta_{ij} = 0$ in terms of the energy on (i, j) or on its neighboring edges. This observation leaves setting $\theta_{ij} = 0$ in the negative domain as the only choice to keep f non-decreasing.

References

1 Anurag Anshu, David Gosset, and Karen Morenz. Beyond Product State Approximations for a Quantum Analogue of Max Cut. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2020.7.

2 Anurag Anshu, David Gosset, Karen J. Morenz Korol, and Mehdi Soleimanifar. Improved approximation algorithms for bounded-degree local hamiltonians. *Phys. Rev. Lett.*, 127:250502, December 2021. doi:10.1103/PhysRevLett.127.250502.

3 Nikhil Bansal, Sergey Bravyi, and Barbara M. Terhal. Classical approximation schemes for the ground-state energy of quantum and classical Ising spin Hamiltonians on planar graphs. Quant. Inf. Comp. Vol. 9, No.8, p. 0701 (2009), 2007. arXiv:0705.1115v4. arXiv:0705.1115.

4 Fernando G. S. L. Brandão and Aram W. Harrow. Product-state approximations to quantum states. *Communications in Mathematical Physics*, 342(1):47–80, February 2016. doi:10.1007/s00220-016-2575-1.

5 Sergey Bravyi, David Gosset, Robert König, and Kristan Temme. Approximation algorithms for quantum many-body problems. *Journal of Mathematical Physics*, 60(3):032203, 2019. doi:10.1063/1.5085428.

6 Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

7 Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014. doi:10.48550/arXiv.1411.4028.

8 Sevag Gharibian and Julia Kempe. Approximation algorithms for QMA-complete problems. SIAM Journal on Computing 41(4): 1028-1050, 2012, 2011. arXiv:1101.3884v1. doi:10.1137/110842272.

9 Sevag Gharibian and Julia Kempe. *Hardness of approximation for quantum problems*, pages 387–398. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-31594-7_33.

10 Sevag Gharibian and Ojas Parekh. Almost Optimal Classical Approximation Algorithms for a Quantum Generalization of Max-Cut. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.31.

11 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. doi:10.1145/227683.227684.

12 Sean Hallgren, Eunou Lee, and Ojas Parekh. An approximation algorithm for the max-2-local hamiltonian problem. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 59:1–59:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.59.

13 Yeongwoo Hwang, Joe Neeman, Ojas Parekh, Kevin Thompson, and John Wright. Unique games hardness of quantum max-cut, and a vector-valued borell's inequality, 2021. doi:10.48550/arXiv.2111.01254.

14 John Kallaugher and Ojas Parekh. The quantum and classical streaming complexity of quantum and classical max-cut. *CoRR*, abs/2206.00213, 2022. doi:10.48550/arXiv.2206.00213.

15 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

16 Karen J Morenz Korol, Kenny Choo, and Antonio Mezzacapo. Quantum approximation algorithms for many-body and electronic structure problems. *arXiv preprint*, 2021. arXiv:2111.08090.

17 Miguel Navascues, Stefano Pironio, and Antonio Acín. Convergent hierarchy of semidefinite programs characterizing the set of quantum correlations. *New Journal of Physics*, 10, July 2008. doi:10.1088/1367-2630/10/7/073013.

18 Ojas Parekh and Kevin Thompson. Application of the Level-2 Quantum Lasserre Hierarchy in Quantum Approximation Algorithms. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 102:1–102:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.102.

19 Ojas Parekh and Kevin Thompson. Beating random assignment for approximating quantum 2-local hamiltonian problems. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 74:1–74:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.74.

20 Ojas Parekh and Kevin Thompson. An optimal product-state approximation for 2-local quantum hamiltonians with positive terms. *CoRR*, abs/2206.08342, 2022. doi:10.48550/arXiv.2206.08342.

21 Stephen Piddock and Ashley Montanaro. The complexity of antiferromagnetic interactions and 2d lattices. *Quantum Inf. Comput.*, 17(7&8):636–672, 2017. doi:10.26421/QIC17.7-8-6.

A Omitted proofs

A.1 Proof of Lemma 5

Proof. Given a quantum state $|\phi\rangle$, define a matrix M by matrix elements

$$M(\Phi, \Psi) := \mathrm{Re}[\langle\phi|\Psi\Phi|\phi\rangle] = \langle\phi|(\Phi\Psi + \Psi\Phi)|\phi\rangle/2$$

for all $\Phi, \Psi \in \mathcal{P}_n(k)$.

We will show that M satisfies the constraints of the SDP (L_k), and that the objective value of M is equal to the energy of $|\phi\rangle$ to prove the lemma. The first constraint is satisfied because of a Pauli identity $\Phi^2 = I$ for all Pauli Φ, because we get $M(\Phi, \Phi) = \langle\phi|(I + I)|\phi\rangle/2 = 1$. The second constraint comes directly from the definition of M.

For the other conditions, note that all the Pauli matrices are Hermitian. Therefore if $\Phi_1\Psi_1 = \Phi_2\Psi_2$, we have $\Psi_1\Phi_1 = (\Phi_1\Psi_1)^\dagger = (\Phi_2\Psi_2)^\dagger = \Psi_2\Phi_2$. So the third constraint is satisfied, and the fourth for similar reasons.

To see that M is PSD, observe that the matrix M is real and symmetric because the Pauli matrices are Hermitian. Now, consider that for an arbitrary real vector $v \in \mathbb{R}^{|\mathcal{P}_n(k)|}$, we have

$$
\begin{aligned}
v^T M v &= \sum_{\Phi,\Psi} v_\Phi M(\Phi,\Psi) v_\Psi \\
&= \sum_{\Phi,\Psi} v_\Phi \langle\phi|(\Phi\Psi + \Psi\Phi)|\phi\rangle v_\Psi/2 \\
&= \langle\phi| \sum_{\Phi,\Psi} (v_\Phi \Phi\Psi v_\Psi + v_\Psi \Psi\Phi v_\Phi)|\phi\rangle/2 \\
&:= \langle\phi|\Phi'\Phi'|\phi\rangle = \|\Phi'|\phi\rangle\|^2 \geq 0,
\end{aligned}
$$

where $\Phi' := \sum_\Phi v_\Phi \Phi$.

Finally to see that the objective value of M is equal to the energy of $|\phi\rangle$,

$$
\begin{aligned}
\langle\phi|H|\phi\rangle &= \sum_{(i,j)\in E} \langle\phi| \frac{1}{4} w_{ij}(I - X_iX_j - Y_iY_j - Z_iZ_j)|\phi\rangle \\
&= \sum_{(i,j)\in E} \frac{1}{4} w_{ij}(1 - \langle\phi|X_iX_j|\phi\rangle - \langle\phi|Y_iY_j|\phi\rangle - \langle\phi|Z_iZ_j|\phi\rangle) \\
&= \sum_{(i,j)\in E} \frac{1}{4} w_{ij}(M(I,I) - M(X_iX_j,I) - M(Z_iZ_j,I) - M(Z_iZ_j,I)).
\end{aligned}
$$

Therefore the SDP (L_k) is a relaxation of Quantum Max Cut. ◀

A.2 Proof of Lemma 10

Proof. We prove the lemma from the SDP constraints and the properties of the SDP solutions that are stated in Lemma 8. We have

$$
\begin{aligned}
\Pr[z_i \neq z_j] &= \frac{1}{3} \sum_{a\in\{1,2,3\}} \Pr_{r\in S^{N-1}}[\mathrm{sgn}(v_{i,a}\cdot r) = -\mathrm{sgn}(v_{j,a}\cdot r)] && (25)\\
&= \frac{1}{3} \sum_{a\in\{1,2,3\}} \frac{\arccos(v_{i,a}\cdot v_{j,a})}{\pi} && (26)\\
&= \frac{1}{3} \sum_{a\in\{1,2,3\}} \frac{\arccos(v_{i,a}\cdot v_{j,a})/\pi}{1/2 - v_{i,a}\cdot v_{j,a}/2} \left[\frac{1}{2} - \frac{v_{i,a}\cdot v_{j,a}}{2}\right] \\
&\geq \frac{1}{3} \min_{t\in[-1,1]} \frac{\arccos(t)/\pi}{1/2 - t/2} \sum_{a\in\{1,2,3\}} \left[\frac{1}{2} - \frac{v_{i,a}\cdot v_{j,a}}{2}\right] \\
&= \frac{\alpha_{GW}}{3} \left[\frac{3}{2} - \frac{\sum_{a\in\{1,2,3\}} v_{ij,a}\cdot v_0}{2}\right] \\
&= \frac{\alpha_{GW}}{3} \left[2 - \frac{(v_0 + v_{ij})\cdot v_0}{2}\right] \\
&= \frac{\alpha_{GW}}{3} \left[2 - \frac{(v_0 + v_{ij})\cdot v_0}{\|v_0 + v_{ij}\|\|v_0\|}\right] && (27)\\
&= \frac{\alpha_{GW}}{3}(2 + \gamma_{ij}).
\end{aligned}
$$

We get the equation (25) from the construction of the algorithm (the line 4 of Algorithm 1), the equation (26) from averaging over the random vector r over the sphere S^{N-1} as in the original Goemans-Williamson paper [11], and finally the equation (27) from the equation (11) of Lemma 8. ◀

A.3 Proof of Lemma 11

Proof. We largely take the analysis from [1] except that we allow different parameters on different edges. For the case of $z_i \neq z_j$, we will show something stronger; namely,

$$\langle\phi(\vec{\theta})|4H_{ij}|\phi(\vec{\theta})\rangle = 1 + \sin(2\theta_{ij})\left[\prod_{k\in N(i)\setminus\{j\}}\cos(2\theta_{ik}) + \prod_{k\in N(j)\setminus\{i\}}\cos(2\theta_{kj})\right] + \sum_{\substack{\Delta'\subset\Delta_{ij}\\ |\Delta'|:\text{ even}}}\prod_{k\in\Delta'}\frac{\sin(2\theta_{ik})\sin(2\theta_{kj})}{\cos(2\theta_{ik})\cos(2\theta_{kj})}\prod_{k\in N(i)\setminus\{j\}}\cos(2\theta_{ik})\prod_{k\in N(j)\setminus\{i\}}\cos(2\theta_{kj}),$$

where $\Delta_{ij} := N(i)\cap N(j)$. The inequality in the lemma is obtained by running the first summation just on $\Delta' = \phi$.

By symmetry, we can assume $z_i = 0, z_j = 1$. Then by construction, we have $P(i)P(j) = Y_iX_j$. From the decomposition $4H_{ij} = 1 - X_iX_j - Y_iY_j - Z_iZ_j$, we calculate the energy contribution $\langle\phi(\vec{\theta})|4H_{ij}|\phi(\vec{\theta})\rangle$ term by term. For the X_iX_j term, we have

$$\begin{aligned}
&\langle\phi(\vec{\theta})|X_iX_j|\phi(\vec{\theta})\rangle\\
&= \langle z|\prod_{(k,l)\in E}\exp[-i\theta_{kl}P(k)P(l)]X_iX_j\prod_{(k,l)\in E}\exp[i\theta_{kl}P(k)P(l)]|z\rangle\\
&= \langle z|\prod_{k\in N(i)}\exp[-i2\theta_{ik}P(i)P(k)]X_iX_j|z\rangle\\
&= \langle z|\prod_{k\in N(i)}[\cos(2\theta_{ik}) - i\sin(2\theta_{ik})Y_iP(k)]\,X_iX_j|z\rangle\\
&= -\langle z|i\sin(2\theta_{ij})Y_iX_j\prod_{k\in N(i)\setminus\{j\}}\cos(2\theta_{ik})X_iX_j|z\rangle\\
&= -\sin(2\theta_{ij})\prod_{k\in N(i)\setminus\{j\}}\cos(2\theta_{ik}).
\end{aligned}$$

Here, we used the standard commutator relations of Pauli matrices. From the line 4 and 5, we used the fact that the Pauli matrices on each qubit in $N(i)\setminus\{j\}$ should multiply to I or Z to yield a non-zero contribution.

Similar calculations yield

$$\langle\phi(\vec{\theta})|Y_iY_j|\phi(\vec{\theta})\rangle = -\sin(2\theta_{ij})\prod_{k\in N(j)\setminus\{i\}}\cos(2\theta_{kj}).$$

For the Z_iZ_j term,

$$
\begin{aligned}
&\langle\phi(\vec{\theta})|Z_iZ_j|\phi(\vec{\theta})\rangle \\
&= \langle z| \prod_{(k,l)\in E} \exp[-i\theta_{kl}P(k)P(l)] Z_iZ_j \prod_{(k,l)\in E} \exp[i\theta_{kl}P(k)P(l)]|z\rangle \\
&= \langle z| \prod_{k\in N(i)\setminus\{j\}} \exp[-i2\theta_{ik}P(i)P(k)] \prod_{k\in N(j)\setminus\{i\}} \exp[-i2\theta_{kj}P(k)P(j)] Z_iZ_j|z\rangle \qquad (28) \\
&= \langle z| \prod_{k\in N(i)\setminus\{j\}} \big[\cos(2\theta_{ik}) - i\sin(2\theta_{ik})P(i)P(k)\big] \\
&\qquad\qquad \prod_{k\in N(j)\setminus\{i\}} \big[\cos(2\theta_{kj}) - i\sin(2\theta_{kj})P(k)P(j)\big] Z_iZ_j|z\rangle \\
&= \langle z| \prod_{k\in\Delta_{ij}} \big[\cos(2\theta_{ik}) - i\sin(2\theta_{ik})P(i)P(k)\big]\big[\cos(2\theta_{kj}) - i\sin(2\theta_{kj})P(k)P(j)\big] \\
&\qquad\qquad \prod_{k\in N(i)\setminus N(j)\setminus\{j\}} \cos(2\theta_{ik}) \prod_{k\in N(j)\setminus N(i)\setminus\{i\}} \cos(2\theta_{kj}) Z_iZ_j|z\rangle \qquad (29) \\
&= \langle z| \prod_{k\in\Delta_{ij}} \big[\cos(2\theta_{ik})\cos(2\theta_{kj}) - \sin(2\theta_{ik})\sin(2\theta_{kj})P(i)P(j)\big] \\
&\qquad\qquad \prod_{k\in N(i)\setminus N(j)\setminus\{j\}} \cos(2\theta_{ik}) \prod_{k\in N(j)\setminus N(i)\setminus\{i\}} \cos(2\theta_{kj}) Z_iZ_j|z\rangle \\
&= \langle z| \prod_{k\in N(i)\cap N(j)} \left[1 - \frac{\sin(2\theta_{ik})\sin(2\theta_{kj})}{\cos(2\theta_{ik})\cos(2\theta_{kj})} Y_iX_j\right] \prod_{k\in N(i)\setminus\{j\}} cos(2\theta_{ik}) \prod_{k\in N(j)\setminus\{i\}} cos(2\theta_{kj}) Z_iZ_j|z\rangle \\
&= -\sum_{\substack{\Delta'\subset\Delta_{ij} \\ |\Delta'|:\text{ even}}} \prod_{k\in\Delta'} \frac{\sin(2\theta_{ik})\sin(2\theta_{kj})}{\cos(2\theta_{ik})\cos(2\theta_{kj})} \prod_{k\in N(i)\setminus\{j\}} \cos(2\theta_{ik}) \prod_{k\in N(j)\setminus\{i\}} \cos(2\theta_{kj}) \qquad (30)
\end{aligned}
$$

To get the equation (28), we used the fact that only the Pauli matrices with non-trivial commutator with Z_iZ_j survive. The equation (29) and (30) are from the fact that an even number of $P(i)$ should be applied on each qubit i to give a non-zero contribution, because the state $|z\rangle$ is an eigenstate of the matrix Z_iZ_j.

In the case of $z_i = z_j$, the inequality (15) is trivial, because the Hamiltonian H_{ij} is a PSD matrix. In fact, we can express the energy as a function of the parameters and the connectivity on the graph as in the earlier case, but we choose not to show since the calculation is similar and is not necessary for the main theorem. ◀

Package Delivery Using Drones with Restricted Movement Areas

Thomas Erlebach
Department of Computer Science, Durham University, UK

Kelin Luo
Institute of Computer Science, Universität Bonn, Germany

Frits C.R. Spieksma
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Abstract

For the problem of delivering a package from a source node to a destination node in a graph using a set of drones, we study the setting where the movements of each drone are restricted to a certain subgraph of the given graph. We consider the objectives of minimizing the delivery time (problem DDT) and of minimizing the total energy consumption (problem DDC). For general graphs, we show a strong inapproximability result and a matching approximation algorithm for DDT as well as *NP*-hardness and a 2-approximation algorithm for DDC. For the special case of a path, we show that DDT is *NP*-hard if the drones have different speeds. For trees, we give optimal algorithms under the assumption that all drones have the same speed or the same energy consumption rate. The results for trees extend to arbitrary graphs if the subgraph of each drone is isometric.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Mobile agents, approximation algorithm, inapproximability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.49

Related Version *Full Version*: https://arxiv.org/abs/2209.12314

Funding *Thomas Erlebach*: Supported by EPSRC grants EP/S033483/2 and EP/T01461X/1.
Kelin Luo: Supported by the National Natural Science Foundation of China, Grant No. 72071157.
Frits C.R. Spieksma: Supported by NWO Gravitation Project NETWORKS, Grant Number 024.002.003.

1 Introduction

Problem settings where multiple drones collaborate to deliver a package from a source location to a target location have received significant attention recently. One motivation for the study of such problems comes from companies considering the possibility of delivering parcels to consumers via drones, e.g., Amazon Prime Air [1]. In previous work in this area [8, 9, 3, 4, 2, 6, 7], the drones are typically modeled as agents that move along the edges of a graph, and the package has to be transported from a source node to a target node in that graph. Optimization objectives that have been considered include minimizing the delivery time, minimizing the energy consumption by the agents, or a combination of the two. A common assumption has been that every agent can travel freely throughout the whole graph [3, 4, 6], possibly with a restriction of each agent's travel distance due to energy constraints [8, 9, 2, 7]. In this paper, we study for the first time a variation of the problem in which each agent is only allowed to travel in a certain subgraph of the given graph.

We remark that the previously considered problem in which each agent has an energy budget that constrains its total distance traveled [8, 9, 2, 7] is fundamentally different from the problem considered here in which each agent can only travel in a certain subgraph: In

our problem, an agent can still travel an arbitrary distance by moving back and forth many times within its subgraph. Furthermore, the subgraph in which an agent can travel cannot necessarily be defined via a budget constraint. This means that neither hardness results nor algorithmic results translate directly between the two problems.

As motivation for considering agents with movement restrictions, we note that in a realistic setting, the usage of drones may be regulated by licenses that forbid some drones from flying in certain areas. The license of a drone operator may only allow that operator to cover a certain area. Furthermore, densely populated areas may have restrictions on which drones are allowed to operate there. Package delivery with multiple collaborating agents might also involve different types of agents (boats, cars, flying drones, etc.), where it is natural to consider the case that each agent can traverse only a certain part of the graph: For example, a boat can only traverse edges that represent waterways.

In our setting, we are given an undirected graph $G = (V, E)$ with a source node s and a target node y of the package as well as a set of k agents. The subgraph in which an agent a is allowed to operate is denoted by $G_a = (V_a, E_a)$. Each agent can pick up the package at its source location or from another drone, and it can deliver the package to the target location or hand it to another drone. We consider the objective of minimizing the delivery time, i.e., the time when the package reaches y, as well as the objective of minimizing the total energy consumption of the drones. We only consider the problem for a single package.

Related work. Collaborative delivery of a package from a source node s to a target node y with the goal of minimizing the delivery time was considered by Bärtschi et al. [4]. For k agents in a graph with n nodes and m edges, they showed that the problem can be solved in $O(k^2m + kn^2 + \text{APSP})$ time for a single package, where APSP is the time for computing all-pairs shortest paths in a graph with n nodes and m edges. Carvalho et al. [6] improved the time complexity for the problem to $O(kn \log n + km)$ and showed that the problem is *NP*-hard if two packages must be delivered.

For minimizing the energy consumption for the delivery of a package, Bärtschi et al. [3] gave a polynomial algorithm for one package and showed that the problem is *NP*-hard for several packages.

The combined optimization of delivery time $\mathcal{T}$ and energy consumption $\mathcal{E}$ for the collaborative delivery of a single package has also been considered. Lexicographically minimizing $(\mathcal{E}, \mathcal{T})$ can be done in polynomial time [5], but lexicographically minimizing $(\mathcal{T}, \mathcal{E})$ or minimizing any linear combination of $\mathcal{T}$ and $\mathcal{E}$ is *NP*-hard [4].

Delivering a package using k energy-constrained agents has been shown to be strongly *NP*-hard in general graphs [8] and weakly *NP*-hard on a path [9]. Bärtschi et al. [2] showed that the variant where each agent must return to its initial location can be solved in polynomial time for tree networks. The variant in which the package must travel via a fixed path in a general graph has been studied by Chalopin et al. [7].

Our results. In Section 2, we introduce definitions and give some auxiliary results. In Section 3, we show that movement restrictions make the drone delivery problem harder for both objectives: For minimizing the delivery time, we show that the problem is *NP*-hard to approximate within ratio $O(n^{1-\epsilon})$ or $O(k^{1-\epsilon})$. For minimizing the energy consumption, we show *NP*-hardness. These results hold even if all agents have the same speed and the same energy consumption rate.

In Section 4, we propose an $O(\min\{n, k\})$-approximation algorithm for the problem of minimizing the delivery time. The algorithm first computes a schedule with minimum delivery time for the problem variant where an arbitrary number of copies of each agent is available.

Then it transforms the schedule into one that is feasible for the problem with a single copy of each agent. The algorithm can also handle handovers on edges. In Section 5, we give a 2-approximation algorithm for the problem of minimizing the total energy consumption. We again first compute an optimal schedule that may use several copies of each agent and then transform the schedule into one with a single copy of each agent.

In Section 6, we first consider the special case where the graph is a path (line) and the subgraph of each agent is a subpath. For this case, we show that the problem of minimizing the delivery time is *NP*-hard if the agents have different speeds. If the agents have the same speed or the same energy consumption rate, we show that the problem of minimizing the delivery time and the problem of minimizing the total energy consumption are both polynomial-time solvable even for the more general case when the graph is a tree, or when the graph is arbitrary but the subgraph of every agent is isometric (defined in Section 6). Conclusions are presented in Section 7. Proofs omitted due to space restrictions can be found in the full version [11].

2 Preliminaries

We now define the *drone delivery* (DD) problem formally. We are given a connected graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$, where $\ell(u, v)$ represents the distance between u and v along edge $\{u, v\}$. (We sometimes write $G = (V, E, \ell)$ to include an explicit reference to ℓ.) Let $n = |V|$ and $m = |E|$. We are also given a set A containing $k \geq 1$ mobile agents (representing drones). Each agent $a \in A$ is specified by $a = (p_a, v_a, w_a, V_a, E_a)$, where $p_a \in V$ is the agent's *initial position*, and $v_a > 0$ and $w_a \geq 0$ are the agent's *velocity* (or speed) and *energy consumption rate*, respectively. To traverse a path of length x, agent a takes time x/v_a and consumes $x \cdot w_a$ units of energy. Furthermore, $V_a \subseteq V$ and $E_a \subseteq E$ are the *node-range* and *edge-range* of agent a, respectively. Agent a can only travel to/via nodes in V_a and edges in E_a. We require $p_a \in V_a$. To ensure meaningful solutions, we make the following two natural assumptions:

- For each agent a, the graph $G_a = (V_a, E_a)$ is a connected subgraph of G.
- The union of the subgraphs G_a over all agents is the graph G, i.e., $\bigcup_{a \in A} V_a = V$ and $\bigcup_{a \in A} E_a = E$. This implies that there is a feasible schedule for any package (s, y).

The package is specified by (s, y), where $s, y \in V$ are the *start node* (start location) and *destination node* (target location), respectively. The task is to find a schedule for delivering the package from the start node to the destination node while achieving a specific objective. The problem of minimizing the delivery time is denoted by DDT, and the problem of minimizing the consumption by DDC.

To describe solutions of the problems, we need to define how a *schedule* can be represented. A schedule is given as a list of *trips* $\mathcal{S} = \{S_1, S_2, \ldots, S_h\}$:

$$\{(a_1, t_1, \langle o_1, \ldots, u_0\rangle, \langle u_0, \ldots, u_1\rangle), \ldots, (a_h, t_h, \langle o_h, \ldots, u_{h-1}\rangle, \langle u_{h-1}, \ldots, u_h\rangle)\}$$

The i-th trip $S_i = (a_i, t_i, \langle o_i, \ldots, u_{i-1}\rangle, \langle u_{i-1}, \ldots, u_i\rangle)$ represents two consecutive trips taken by agent a_i starting at time $t_i \geq 0$: an *empty movement* trip traversing nodes $O_i = \langle o_i, \ldots, u_{i-1}\rangle$, and a *delivery trip* (during which a_i carries the package) of traversing nodes $U_i = \langle u_{i-1}, \ldots, u_i\rangle$. The agent a_i picks up the package at node u_{i-1} and drops it off at node u_i. With slight abuse of notation, we also use O_i and U_i to denote the set of edges in each of these two trips. If S_i is the first trip of agent a_i, then o_i is the agent's initial location. Otherwise, o_i is the location where the agent dropped off the package at the end of its previous trip. Initially, each agent $a \in A$ is at node p_a at time 0. In the definition of

schedules, when we allow two agents to meet on an edge to hand over the package, we allow the nodes also to be points on edges: For example, a node u_i could be the point on an edge $\{u, v\}$ with length 5 that is at distance 2 from u (and hence at distance 3 from v).

Let $T(u_i)$ (resp. $C(u_i)$) denote the time passed (resp. the energy consumed) until the package is delivered to node u_i in schedule $\mathcal{S}$, i.e.,

$$T(u_i) = \max \left\{ T(u_{i-1}), t_i + \sum_{e \in O_i} \frac{\ell(e)}{v_{a_i}} \right\} + \sum_{e \in U_i} \frac{\ell(e)}{v_{a_i}},$$

$$C(u_i) = C(u_{i-1}) + w_{a_i} \cdot \sum_{e \in O_i} \ell(e) + w_{a_i} \cdot \sum_{e \in U_i} \ell(e).$$

The pick-up location of the first agent must be the start node, i.e., $u_0 = s$, and the drop-off location of the last agent must be the destination node, $u_h = y$. We let $T(u_0) = 0$ and $C(u_0) = 0$. The goal of the DDT problem is to find a feasible schedule $\mathcal{S}$ that minimizes the delivery time $T(y)$, and the goal of the DDC problem is to find a feasible schedule $\mathcal{S}$ that minimizes the energy consumption $C(y)$.

So far, we have defined the DDT and DDC problem. As in previous studies [6], we further distinguish variants of these problems based on the *handover manner*.

Handover manner. The handover of the package between two agents may occur at a node or at some interior point of an edge. When the handovers are restricted to be on nodes, we say the drone delivery problem is handled with *node handovers*, and when the handover can be done on a node or at an interior point of an edge, we say that the drone delivery problem is handled with *edge handovers*. We use the subscripts N and E to represent node handovers and edge handovers, respectively. Thus, we get four variants of the drone delivery problem: DDT_N, DDT_E, DDC_N and DDC_E.

With or without initial positions. We additionally consider problem variants in which the initial positions of the agents are not fixed (given), which means that the initial positions p_a for $a \in A$ can be chosen by the algorithm. When the initial positions are fixed, we say that the problem is *with initial positions*, and when the initial positions are not fixed, we say that the problem is *without initial positions*. When we do not specify that a problem is without initial positions, we always refer to the problem with initial positions by default.

In the rest of the paper, in order to simplify notation, the i-th trip in a schedule $\mathcal{S}$ is usually written in simplified form as (a_i, u_{i-1}, u_i), where a_i is the agent, u_{i-1} is the pick-up location of the package, and u_i is the drop-off location of the package. We can omit the agent's empty movement trip (including its previous position) and its start time because the agent a_i always takes the path in $G_{a_i} = (V_{a_i}, E_{a_i})$ with minimum cost (travel time or energy consumption) from its previous position to the current pick-up location u_{i-1} and then from u_{i-1} to u_i.

For each node $v \in V$, we use $A(v)$ to denote the set of agents that can visit the node v; and for every edge $\{u, v\}$ in E, we use $A(u, v)$ to denote the set of agents that can traverse the edge $\{u, v\}$. For any pair of nodes $u, v \in V_a$ for some $a \in A$, we denote by $\mathsf{dist}_a(u, v)$ the length of the shortest path between node u and v in the graph $G_a = (V_a, E_a)$.

Useful properties. We show some basic properties for the drone delivery problem. The following lemma can be shown by an exchange argument: If an agent was involved in the package delivery at least twice, letting that agent keep the package from its first involvement to its last involvement does not increase the delivery time nor the energy consumption.

▶ **Lemma 1.** *For every instance of* $\mathsf{DDT_N}$, $\mathsf{DDT_E}$, $\mathsf{DDC_N}$ *and* $\mathsf{DDC_E}$ *(with or without initial positions), there is an optimal solution in which each of the involved agents picks up and drops off the package exactly once.*

For DDC, we can show that handovers at interior points of edges cannot reduce the energy consumption. If two agents carry the package consecutively over parts of the same edge, letting one of the two agents carry the package over those parts can be shown not to increase the energy consumption.

▶ **Lemma 2.** *For any instance of* $\mathsf{DDC_N}$ *and* $\mathsf{DDC_E}$ *(with or without initial positions), there is a solution that is simultaneously optimal for both problems. In other words, there is an optimal solution for* $\mathsf{DDC_E}$ *in which all handovers of the package take place at nodes of the graph.*

For the case that $G_a = G$ for all $a \in A$, it has been observed in previous work [4, 6] that there is an optimal schedule for $\mathsf{DDT_N}$ and $\mathsf{DDT_E}$ in which the velocities of the involved agents are strictly increasing, and an optimal schedule for $\mathsf{DDC_N}$ and $\mathsf{DDC_E}$ in which the consumption rates of the involved agents are strictly decreasing. We remark that this very useful property does not necessarily hold in our setting with agent movement restrictions. This is the main reason why the problem becomes harder, as shown by our hardness results in Section 3 and, even for path networks, in Section 6.

3 Hardness results

In this section, we prove several hardness results for DDT and DDC via reductions from the *NP*-complete 3-*dimensional matching problem* (3DM) [12]. All of them apply to the case where all agents have the same speed and the same energy consumption rate. We first present the construction of a *base instance* of DDT, which we then adapt to obtain the hardness results for DDT and DDC. The instances that we create have the property that every edge e of the graph is only in the set E_a of a single agent a. Therefore, handovers on edges are not possible for these instances, and so the variants of the problems with node handovers and with edge handovers are equivalent for these instances. Hence, we only need to consider the problem variant with node handovers in the proofs.

The problem 3DM is defined as follows: Given are three sets X, Y, Z, each of size n, and a set $\mathcal{F} \subseteq X \times Y \times Z$ consisting of m triples. Each triple in $\mathcal{F}$ is of the form (x, y, z) with $x \in X$, $y \in Y$, and $z \in Z$. The question is: Is there a set of n triples in $\mathcal{F}$ such that each element of $X \cup Y \cup Z$ is contained in exactly one of these n triples?

Let an instance of 3DM be given by X, Y, Z and $\mathcal{F}$. Assume $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$ and $Z = \{z_1, z_2, \ldots, z_n\}$. Number the triples in $\mathcal{F}$ from 1 to $m = |\mathcal{F}|$ arbitrarily, and let the j-th triple be $t_j = (x_{f(j)}, y_{g(j)}, z_{h(j)})$ for suitable functions $f, g, h : [m] \to [n]$. We create from this a base instance I of $\mathsf{DDT_N}$ that consists of n selection gadgets and $3n$ agent gadgets. Carrying the package through a selection gadget corresponds to selecting one of the m triples of $\mathcal{F}$. The n selection gadgets are placed consecutively, so that the package travels through all of them (unless it makes a detour that increases the delivery time). For each element q of $X \cup Y \cup Z$, there is an agent gadget containing the start position of a unique agent. The agent gadget for element q ensures that the agent can carry the package on the edge corresponding to element q on a path through a selection gadget if and only if that path corresponds to a triple that contains q. If the instance of 3DM is a yes instance, then each of the agents from the agent gadgets only needs to transport

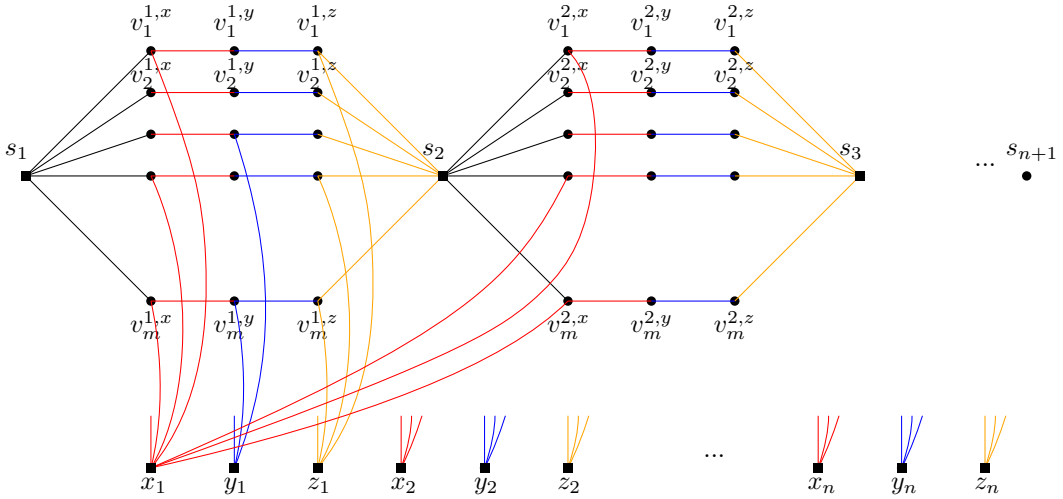


Figure 1 The graph $\bar{G} = (\bar{V}, \bar{E})$ used for the inapproximability and hardness proofs for DDT and DDC. The inner edges have length 0, and the outer edges have length M. Each square node represents a drone's initial location. Each colored (red, blue, and orange) path represents a set of edges that can be visited by the same agent which is initially located at a location in $\{x_i, y_i, z_i \mid i \in [n]\}$. For each $i \in [n]$, there is an additional agent that is initially located at s_i and which can visit edges $\{\{s_i, v_x^{i,x}\} \mid j \in [m]\}$.

the package on one edge. Otherwise, an agent must travel (with or without the package) from one selection gadget to another one, and the extra time consumed by this movement increases the delivery time.

Formally, the instance of DDT_N with graph $\bar{G} = (\bar{V}, \bar{E})$ is created as follows. See Figure 1 for an illustration. The vertex set $\bar{V}$ contains $3n$ *agent nodes* and $n(3m+1)+1$ nodes in selection gadgets, a total of $4n + 3mn + 1$ nodes. The agent nodes comprise a node x_i for each $x_i \in X$, a node y_i for each $y_i \in Y$, and a node z_i for each $z_i \in Z$. The node set of selection gadget i, for $1 \leq i \leq n$, is $\{s_i, s_{i+1}\} \cup \{v_j^{i,x}, v_j^{i,y}, v_j^{i,z} \mid 1 \leq j \leq m\}$, a set containing $3m+2$ nodes. For $1 \leq i < n$, the node s_{i+1} is contained in selection gadget i and in selection gadget $i+1$. The path through the three nodes $v_j^{i,x}, v_j^{i,y}, v_j^{i,z}$ (in selection gadget i for any i) represents the triple t_j in $\mathcal{F}$. The package is initially located at node s_1 and must be delivered to node s_{n+1}.

The edge set $\bar{E}$ contains two types of edges: *inner* edges, each of which connects two nodes in a selection gadget, and *outer* edges, each of which connects an agent node and a node in a selection gadget. For every $i \in [n]$ and $j \in [m]$, there are four inner edges $\{s_i, v_j^{i,x}\}$, $\{v_j^{i,x}, v_j^{i,y}\}$, $\{v_j^{i,y}, v_j^{i,z}\}$ and $\{v_j^{i,z}, s_{i+1}\}$ of length 0.[1] So there are $4mn$ inner edges. The outer edges are as follows: For every $i \in [n]$ and $j \in [m]$, there are the three outer edges $\{x_{f(j)}, v_j^{i,x}\}$, $\{y_{g(j)}, v_j^{i,y}\}$, $\{z_{h(j)}, v_j^{i,z}\}$ of length M for some $M > 0$. (We can set $M = 1$.)

This completes the description of the graph $\bar{G} = (\bar{V}, \bar{E})$ with edge lengths.

Now we define the set of agents. We have $4n$ agents with unit speed and unit energy consumption rate. Their initial locations are $\{x_i, y_i, z_i, s_i \mid i \in [n]\}$. The agents with initial location in $\{x_i, y_i, z_i \mid i \in [n]\}$ are called *element agents*. For each $i \in [n]$, the node ranges and edge ranges of the agents with initial location with subscript i are defined as follows:

[1] We could set the lengths of these edges to a small $\epsilon > 0$ if we wanted to avoid edges of length 0.

- The agent located at x_i has node range $\{x_i, v_j^{i',x}, v_j^{i',y} \mid i' \in [n], j \in [m], f(j) = i\}$ and edge range $\{\{x_i, v_j^{i',x}\}, \{v_j^{i',x}, v_j^{i',y}\} \mid i' \in [n], j \in [m], f(j) = i\}$.
- The agent located at y_i has node range $\{y_i, v_j^{i',y}, v_j^{i',z} \mid i' \in [n], j \in [m], g(j) = i\}$ and edge range $\{\{y_i, v_j^{i',y}\}, \{v_j^{i',y}, v_j^{i',z}\} \mid i' \in [n], j \in [m], g(j) = i\}$.
- The agent located at z_i has node range $\{z_i, v_j^{i',z}, s_{i'+1} \mid i' \in [n], j \in [m], h(j) = i\}$ and edge range $\{\{z_i, v_j^{i',z}\}, \{v_j^{i',z}, s_{i+1}\} \mid i' \in [n], j \in [m], h(j) = i\}$.
- The agent located at s_i has node range $\{s_i, v_j^{i,x} \mid j \in [m]\}$ and edge range $\{\{s_i, v_j^{i,x}\} \mid j \in [m]\}$.

We now show that the given instance of 3DM is a yes-instance if and only if the constructed base instance of $\mathsf{DDT_N}$ has an optimal schedule with delivery time M. First, assume that the given instance of 3DM is a yes-instance, and let $\{t_{k_1}, t_{k_2}, \ldots, t_{k_n}\} \subseteq \mathcal{F}$ be a perfect matching. We let the package travel from s_1 to s_{n+1} via the n selection gadgets, using the path via nodes $v_{k_i}^{i,x}, v_{k_i}^{i,y}, v_{k_i}^{i,z}$ in selection gadget i, for $i \in [n]$. This path consists of $4n$ edges. Each of the $4n$ agents needs to carry the package on exactly one edge of this path. All the element agents reach the node where they pick up the package at time M. As all edges in the selection gadgets have length 0, this shows that the package reaches s_{n+1} at time M. It is clear that this solution is optimal because at least one element agent must take part in the delivery and cannot pick up the package before time M.

Now, assume that the base instance of $\mathsf{DDT_N}$ admits a schedule with delivery time M. It is not hard to see that the schedule must be of the above format, using each agent on exactly one edge in one selection gadget. This is because it takes time $2M$ for an element agent to move from one selection gadget to another one via its initial location.

This shows that there is a schedule with delivery time M if and only if the given instance of 3DM is a yes-instance. Furthermore, as any element agent needs time M to reach a pick-up point in a selection gadget and time $2M$ to move to a different selection gadget (with or without the package), it is clear that the optimal schedule will have length at least $3M$ if the given instance of 3DM is a no-instance. This already shows that $\mathsf{DDT_N}$ is *NP*-hard and does not admit a polynomial-time approximation algorithm with approximation ratio smaller than 3 unless $P = NP$, but we can strengthen the inapproximability result by concatenating $q = (4n+3mn)^c$ copies of the base instance (identifying node s_{n+1} of one copy with node s_1 of the next copy) for an arbitrarily large constant c and letting the package be transported from s_1 in the first copy to s_{n+1} in the last copy. If the given instance of 3DM is a yes-instance, the delivery time is still M, but if it is a no-instance, it is $M + 2Mq = M(2q + 1)$.

▶ **Theorem 3.** *For any constant $\epsilon > 0$, it is NP-hard to approximate $\mathsf{DDT_N}$ (or $\mathsf{DDT_E}$) within a factor of $O(\min\{n^{1-\epsilon}, k^{1-\epsilon}\})$ even if all agents have the same speed.*

To show *NP*-hardness for DDC, we adapt the base instance by numbering the columns of the selection gadgets to which outer edges are attached from 1 to $3n$ (from left to right) and letting the outer edges attached to column i have length 2^{3n+1-i}. If the given instance of 3DM is a yes-instance, exactly one outer edge attached to each column will be used, for a total energy consumption of $2^{3n+1} - 2$. Otherwise, some column will be the first column in which an outer edge is used twice, and the total energy consumption will be at least 2^{3n+1}.

▶ **Theorem 4.** *The problems $\mathsf{DDC_N}$ (and $\mathsf{DDC_E}$) are NP-hard even if all agents have the same energy consumption.*

Finally, for the problem variants without initial positions, we observe that the base instance admits a delivery schedule with delivery time 0 and energy consumption 0 if and only if the given instance of 3DM is a yes-instance. This shows that there cannot be any

polynomial-time approximation algorithm for any of the DDT and DDC problem variants without initial positions. This holds since we allow zero-length edges. If we were to require strictly positive edge lengths, it would be possible to obtain approximation algorithms with ratios that depend on the ratio of maximum to minimum edge length.

4 Approximation algorithm for the DDT problem

We first present an optimal algorithm for DDT_N under the assumption that there are as many copies of each agent as we need. We start by introducing some notation used in the algorithm. For any edge $\{u, v\} \in E_a$ for some $a \in A$, we denote by $\mathsf{eT}_a(v, u \prec v)$ the earliest time for the package to arrive at v if the package is carried over the edge $\{u, v\}$ by a copy of agent a. In addition, we use $\mathsf{eT}(v, u \prec v)$ to denote the earliest time for the package to arrive at node v if the package is carried over the edge $\{u, v\}$ by some agent, i.e., $\mathsf{eT}(v, u \prec v) = \min\{\mathsf{eT}_a(v, u \prec v) \mid \{u, v\} \in E_a, a \in A\}$. For every $v \in V$, we use $\mathsf{eT}(v)$ to denote the earliest arrival time for the package at node v, i.e., $\mathsf{eT}(v) = \min\{\mathsf{eT}(v, u \prec v) \mid \{u, v\} \in E\}$. Note that the package is initially at location s at time 0, i.e., $\mathsf{eT}(s) = 0$. Given a node v, we denote by $\mathcal{S}(v)$ the schedule for carrying the package from s to v, i.e., $\mathcal{S}(v) = \{(a_1, s, u_1), (a_2, u_1, u_2), \ldots, (a_h, u_{h-1}, v)\}$.

Our algorithm adapts the approach of a time-dependent Dijkstra's algorithm [4, 6]. Algorithm 1 shows the pseudo-code. For each node $v \in V$, we maintain a value $\mathsf{eT}(v)$ that represents the current upper bound on the earliest time when the package can reach v (line 3–4). Initially, we set the earliest arrival time for s to 0 and for all other nodes to ∞. We maintain a priority queue Q of nodes v with finite $\mathsf{eT}(v)$ value that have not been processed yet (line 8). In each iteration of the while-loop, we process a node u in Q having the earliest arrival time (line 10), where $u = s$ in the first iteration. For each unprocessed neighbor node v of u, we calculate the earliest arrival time $\mathsf{eT}(v, u \prec v)$ at node v if the package is carried over the edge between u and v by some agent (and we store the identity of that agent in $L(v, u \prec v)$), by calling the subroutine NeiDelivery(u, v, t) (Algorithm 2). If this earliest arrival time is smaller than $\mathsf{eT}(v)$, then $\mathsf{eT}(v)$ is updated and v is inserted into Q (if it is not yet in Q) or its priority reduced to the new value of $\mathsf{eT}(v)$. The algorithm terminates and returns the value $\mathsf{eT}(y)$ when the node being processed is y (line 12). The schedule $\mathcal{S} = \mathcal{S}(y)$ can be constructed in a backward manner because we store for each node v the involved agent $L(v)$ and its predecessor node $\mathsf{prev}(v)$ (line 20 and 21).

To compute $\mathsf{eT}(v, u \prec v)$ in line 17 of Algorithm 1, we call NeiDelivery(u, v, t) (Algorithm 2): That subroutine calculates for each agent $a \in A$ with $\{u, v\} \in E_a$, i.e., for all $a \in A(u, v)$, the time when the package reaches v if that agent picks it up at u and carries it over $\{u, v\}$ to v. The earliest arrival time at v via the edge $\{u, v\}$ is returned as $\mathsf{eT}(v, u \prec v)$, and the agent a^* achieving that arrival time is returned as $L(v, u \prec v)$.

▶ **Lemma 5.** *The following statements hold for the schedule found by Algorithm 1.*

(i) *It may happen that an agent picks up and drops off the package more than once. Each time an agent a carries the package over a path of consecutive edges, a copy of the agent starts at p_a, travels to the start node u of the path, picks up the package at time $\max\{\mathsf{eT}(u), \frac{\mathsf{dist}_a(p_a, u)}{v_a}\}$, and then carries the package over the edges of the path.*

(ii) *The package is carried to each node $v \in V$ at most once, and thus the schedule carries the package over at most $|V| - 1$ edges.*

▶ **Lemma 6.** *There is an algorithm that computes an optimal schedule in time $O(k(m + n \log n))$ for DDT_N under the assumption that an arbitrary number of copies of each agent can be used. The package gets delivered from s to y along a simple path with at most $|V| - 1$ edges.*

Algorithm 1 Algorithm for DDT.

Data: Graph $G = (V, E, \ell)$; package source node s and target node y; agent a with velocity v_a and initial location p_a for $a \in A$
Result: earliest arrival time for package at target location y, i.e., $\mathsf{eT}(y)$

```
1  begin
2      compute dist_a(p_a, v) for a ∈ A and v ∈ V_a
3      eT(s) ← 0
4      eT(v) ← ∞ for all v ∈ V \ {s}
5      L(v) ← ∅ for all v ∈ V // agent bringing the package to v
6      proc(v) ← 0 for all v ∈ V // all nodes v are unprocessed
7      prev(v) ← ∅ for all v ∈ V // previous node on optimal package path to v
8      Q ← {s} // priority queue of pending nodes
9      while Q ≠ ∅ do
10         u ← arg min{eT(v) | v ∈ Q} // node with minimum arrival time in Q
11         Q ← Q \ {u} and proc(u) ← 1
12         if u = y then
13             break
14         end
15         t ← eT(u) // arrival time when package reaches u
16         for neighbors v of u with proc(v) = 0 and A(u, v) ≠ ∅ do
17             {eT(v, u ≺ v), L(v, u ≺ v)} ← NeiDelivery(u, v, t)
18             if eT(v, u ≺ v) < eT(v) then
19                 eT(v) ← eT(v, u ≺ v)
20                 L(v) ← L(v, u ≺ v)
21                 prev(v) ← {u}
22                 if v ∉ Q then
23                     Q ← Q ∪ {v} with earliest arrival time eT(v)
24                 end
25             end
26         end
27     end
28     return eT(y)
29 end
```

Proof. We claim that the arrival time $\mathsf{eT}(u)$ for each node u is minimum by the time u gets processed. Obviously, the first processed node s has arrival time $\mathsf{eT}(s) = 0$. Whenever a node u is processed, the earliest arrival time for each unprocessed neighbor is updated (line 19 of Algorithm 1) if the package can reach that neighbor earlier via node u (line 4 in Algorithm 2 identifies the agent a^* that can bring the package from u to the neighbor the fastest). At the time when a node u is removed from the priority queue Q and starts to be processed, all nodes v with $\mathsf{eT}(v) < \mathsf{eT}(u)$ have already been processed, and so its value $\mathsf{eT}(u)$ must be equal to the earliest time when the package can reach u. The algorithm terminates when y is removed from Q.

In lines 19–21 of Algorithm 1, we update the arrival time $\mathsf{eT}(v)$ and the agent $L(v)$ as well as the predecessor node $\mathsf{prev}(v)$ without explicitly maintaining the schedules $\mathcal{S}(v)$. The schedule $\mathcal{S}(v)$ can be retraced from $L(\cdot)$ and $\mathsf{prev}(\cdot)$ since the schedule found for $\mathsf{eT}(v)$ visits each node in V at most once, cf. Lemma 5. This also shows that the package gets delivered from s to y along a simple path (with at most $|V| - 1$ edges) in G.

Algorithm 2 Algorithm NeiDelivery(u, v, t) for $\mathsf{DDT_N}$.

Data: Edge $\{u, v\}$, arrival time for node u $\mathsf{eT}(u) = t$, agents $A(u, v)$
Result: $\mathsf{eT}(v, u \prec v)$
1 **for** $a \in A(u, v)$ **do**
2 $\quad \mathsf{eT}_a(v, u \prec v) = \max\{t, \frac{\mathsf{dist}_a(p_a, u)}{v_a}\} + \frac{\ell(u,v)}{v_a}$
3 **end**
4 $a^* = \arg\min\{\mathsf{eT}_a(v, u \prec v) \mid a \in A(u, v)\}$;
5 $\mathsf{eT}(v, u \prec v) \leftarrow \mathsf{eT}_{a^*}(v, u \prec v)$
6 $L(v, u \prec v) \leftarrow a^*$
7 **return** $\mathsf{eT}(v, u \prec v)$ and $L(v, u \prec v)$

We can analyze the running-time of Algorithm 1 as follows. The distances $\mathsf{dist}_a(p_a, v)$ can be pre-computed by running Dijkstra's algorithm with Fibonacci heaps in time $O(m + n \log n)$ [10] for each agent a with source node p_a in the graph G_a, taking total time $O(k(m + n \log n))$ (line 2). Selecting and removing a minimum element from the priority queue Q in lines 10–11 takes time $O(\log n)$. At most n nodes will be added into Q (and later removed from Q), so the running time for inserting and removing elements from Q is $O(n \log n)$. For each processed node u, we compute the value $\mathsf{eT}(v, u \prec v)$ for each unprocessed adjacent node v with $A(u, v) \neq \emptyset$ in time $O(|A(u, v)|)$. Overall we get a running time of $O(k(m + n \log n) + n \log n + km) = O(k(m + n \log n))$. ◀

Next, we can show how to convert the delivery schedule $\mathcal{S}$ with delivery time $\mathcal{T}$ produced by the algorithm of Lemma 6 into a schedule that is feasible with a single copy per agent, and we bound the resulting increase in the delivery time $\mathcal{T}$ to obtain an approximation algorithm for $\mathsf{DDT_N}$. The conversion consists of the repeated application of modification steps. Each step considers the first agent a that is used at least twice. Let a pick up the package at u_{i-1} in its first trip and carry it from u_{j-1} to u_j in its last trip. We then modify the schedule so that a picks up the package at u_{i-1} and carries it all the way to u_j along a shortest path in G_a. We have $\frac{1}{v_a}\mathsf{dist}_a(p_a, u_{i-1}) \leq \mathcal{T}$ and $\frac{1}{v_a}(\mathsf{dist}_a(p_a, u_{j-1}) + \mathsf{dist}_a(u_{j-1}, u_j)) \leq \mathcal{T}$. By the triangle inequality, $\mathsf{dist}_a(u_{i-1}, u_j) \leq \mathsf{dist}_a(u_{i-1}, p_a) + \mathsf{dist}_a(p_a, u_{j-1}) + \mathsf{dist}_a(u_{j-1}, u_j)$. Hence, agent a needs time at most $2\mathcal{T}$ to carry the package from u_{i-1} to u_j, and so the delivery time increases by at most $2\mathcal{T}$. Furthermore, we can bound the number of modification steps by $\min\{\frac{n-1}{3}, k - 1\}$.

▶ **Theorem 7.** *There is a* $\min\{2n/3 + 1/3, 2k - 1\}$*-approximation algorithm for* $\mathsf{DDT_N}$.

To adapt the approach from $\mathsf{DDT_N}$ to $\mathsf{DDT_E}$, we can adapt the algorithm of Lemma 6 to edge handovers by using as a subroutine the FastLineDelivery(u, v, t) method from [6], which calculates in $O(k \log k)$ time an optimal delivery schedule using the agents in $A(u, v)$ to transport the package that arrives at u at time t from u to v over the edge $\{u, v\}$. When transforming the resulting package delivery schedule into one that uses each agent at most once, the number of modification steps can be bounded by $\min\{n - 1, k - 1\}$.

▶ **Theorem 8.** *There is a* $\min\{2n - 1, 2k - 1\}$*-approximation algorithm for* $\mathsf{DDT_E}$.

5 Approximation algorithm for the DDC problem

By Lemma 2, handovers on interior points of edges are not needed for DDC, so the results for $\mathsf{DDC_N}$ that we present in this section automatically apply to $\mathsf{DDC_E}$ as well. Therefore, we only consider $\mathsf{DDC_N}$ in the proofs. We first give an algorithm that solves $\mathsf{DDC_N}$ optimally if there is a sufficient number of copies of every agent.

Let an instance of $\mathsf{DDC_N}$ be given by a graph $G = (V, E, \ell)$, package start node s and destination node y, and a set A of k agents where $a = (p_a, v_a, w_a, V_a, E_a)$ for $a \in A$. We create a directed graph G' in which a shortest path from s' to y' corresponds to an optimal delivery schedule. This approach is motivated by a method used by Bärtschi et al. [3]. We construct the directed edge-weighted graph $G' = (V', E', \ell')$ as follows:

- For each node $u \in V$ and each agent $a \in A$ with $u \in V_a$, create a node u_a in V'. In addition, add a node s' and a node y'.
- For all $a \in A$ with $s \in V_a$, add an arc (s', s_a) with $\ell'(s', s_a) = w_a \cdot \mathsf{dist}_a(p_a, s)$. For all $a \in A$ with $y \in V_a$, add an arc (y_a, y') with $\ell'(y_a, y') = 0$.
- For $\{u, x\} \in E$, for each a with $\{u, x\} \in E_a$, create two arcs (u_a, x_a) and (x_a, u_a) with $\ell'(x_a, u_a) = \ell'(u_a, x_a) = w_a \cdot \ell(u, x)$.
- For $u \in V$ and agents $a, \bar{a} \in A(u)$, create the following two arcs: $(u_a, u_{\bar{a}})$ with $\ell'(u_a, u_{\bar{a}}) = w_{\bar{a}} \cdot \mathsf{dist}_{\bar{a}}(p_{\bar{a}}, u)$, and $(u_{\bar{a}}, u_a)$ with $\ell'(u_{\bar{a}}, u_a) = w_a \cdot \mathsf{dist}_a(p_a, u)$.

Intuitively, a node u_a in G' represents the agent a carrying the package at node u in G. An arc (u_a, x_a) represent the agent a carrying the package over edge $\{u, x\}$ from u to x. An arc $(u_a, u_{\bar{a}})$ represents a copy of agent $\bar{a}$ traveling from $p_{\bar{a}}$ to u and taking over the package from agent a there. We can show that a shortest s'-y' path in G' corresponds to an optimal schedule with multiple copies per agent.

▶ **Lemma 9.** *An optimal schedule for* $\mathsf{DDC_N}$ *(and* $\mathsf{DDC_E}$*) can be computed in time* $O(nk^2 + n^2k)$ *under the assumption that an arbitrary number of copies of each agent can be used.*

Proof. We claim that a shortest s'-y' path in G' corresponds to an optimal delivery schedule. First, assume that an optimal delivery schedule $\mathcal{S}$ is

$$\{(a_1, s, u_1), (a_2, u_1, u_2), \ldots, (a_h, u_{h-1}, y)\},$$

where it is possible that $a_i = a_j$ for $j > i + 1$ because we allow copies of agents to be used. Then we can construct an s'-y' path in G' whose length equals the total energy consumption of $\mathcal{S}$ as follows: Start with the arc (s', s_{a_1}). Then use the arcs $(s_{a_1}, z^{(1)}_{a_1}), \ldots, (z^{(g)}_{a_1}, u_{1a_1})$ corresponding to the path $s, z^{(1)}, \ldots, z^{(g)}, u_1$ (for some $g \geq 0$) along which agent a_1 carries the package from s to u_1 in $\mathcal{S}$. Next, use the arc (u_{1a_1}, u_{1a_2}) representing the handover from a_1 to a_2 at u_1. Continue in this way until node y_{a_h} is reached, and then follow the arc from there to y'. Similarly, any s'-y' path P in G' can be translated into a delivery schedule in G whose total energy consumption is equal to the length of P in G'.

Finally, let us consider the running-time of the algorithm. First, we compute $\mathsf{dist}_a(p_a, v)$ for each agent a and each node $v \in V$ by running Dijkstra's algorithm with Fibonacci heaps [10] once in G_a with source node p_a for each $a \in A$, taking time $O(k(n \log n + m)) = O(kn^2)$. The graph G' has at most $n \cdot k + 2 \in O(nk)$ vertices and at most $2k + (n^2 \cdot k + n \cdot k^2) \in O(nk^2 + n^2k)$ arcs. It can be constructed in $O(nk^2 + n^2k)$ time as we have pre-computed the values $\mathsf{dist}_a(p_a, v)$. We can compute the shortest s'-y' path in G' in time $O(nk^2 + n^2k + nk \log(nk)) = O(nk^2 + n^2k)$ time using Dijkstra's algorithm. ◀

▶ **Theorem 10.** *There is a 2-approximation algorithm for* $\mathsf{DDC_N}$ *(and* $\mathsf{DDC_E}$*).*

Proof. Let an instance of $\mathsf{DDC_N}$ be given by a graph $G = (V, E, \ell)$, package start node s and destination node y, and a set A of k agents where $a = (p_a, v_a, w_a, V_a, E_a)$ for $a \in A$. Compute an optimal delivery schedule that may use multiple copies of agents using the algorithm of Lemma 9. Then we transform the schedule into one that uses each agent at most once as follows: Let a be the first agent that is used more than once in the delivery schedule.

Assume that a carries the package from node u to node v during its first involvement in the delivery and from node u' to node v' during its last involvement in the delivery. The energy consumed by these two copies of agent a is:

$$W_a = w_a(\mathsf{dist}_a(p_a, u) + \mathsf{dist}_a(u, v) + \mathsf{dist}_a(p_a, u') + \mathsf{dist}_a(u', v'))$$

We modify the schedule and let agent a pick up the package at u and carry it along a shortest path in G_a from u to v'. The trips in the original schedule that bring the package from v to u' are removed. The new energy consumption by agent a is $w_a(\mathsf{dist}_a(p_a, u) + \mathsf{dist}_a(u, v'))$. By the triangle inequality, $\mathsf{dist}_a(u, v') \leq \mathsf{dist}_a(u, p_a) + \mathsf{dist}_a(p_a, u') + \mathsf{dist}_a(u', v')$. Hence, the new energy consumption is bounded by $2W_a$. As long as there is an agent that is used more than once, we apply the same modification step to the first such agent. The procedure terminates after at most $k - 1$ modification steps. During the execution of the procedure, the energy consumption of every agent at most doubles. Therefore, the total energy consumption of the resulting schedule for $\mathsf{DDC_N}$ with a single copy of each agent is at most twice the total energy consumption of $\mathcal{S}$, which is a lower bound on the optimal energy consumption.

The algorithm of Lemma 9 takes $O(nk^2 + n^2k)$ time, which dominates the time needed to carry out the modification steps. Thus, the overall running time is $O(nk^2 + n^2k)$. ◀

6 Drone delivery on path and tree networks

In Section 6.1, we present hardness results for the drone delivery problems if the graph is a path. In Section 6.2, we show that the problems can be solved optimally in polynomial time for trees (and even for arbitrary graphs if the subgraphs of the agents satisfy a certain condition) provided that all agents have the same speed or the same energy consumption rate.

6.1 Hardness of DDT on the path

▶ **Theorem 11.** $\mathsf{DDT_N}$ *and* $\mathsf{DDT_E}$ *with initial positions are NP-complete if the given graph is a path.*

Proof. We give a reduction from the *NP*-complete Even-Odd Partition (EOP) problem [12] that is defined as follows: Given a set of $2n$ positive integers $X = \{x_1, x_2, \ldots, x_{2n}\}$, is there a partition of X into two subsets X_1 and X_2 such that $\sum_{x_i \in X_1} x_i = \sum_{x_i \in X_2} x_i$ and such that, for each $i \in 1, 2, \ldots, n$, X_1 (and also X_2) contains exactly one element of $\{x_{2i-1}, x_{2i}\}$?

Let an instance of EOP be given by $2n$ numbers $\{x_1, x_2, \ldots, x_{2n-1}, x_{2n}\}$. Construct a path (see Figure 2) with set of nodes $\{s, b_1, b_2, \ldots, b_{2n}, z, z', c_{2n}, c_{2n-1}, \ldots, c_1, y, u\}$ ordered from left to right. The length of each edge corresponds to the Euclidean distance between its endpoints. We let $z' - z = b_{2i} - b_{2i-1} = 1$, $c_{2i-1} - c_{2i} = 1/2$, $z - b_{2n} = c_{2n} - z' = y - c_1 = L$, and $b_{2i+1} - b_{2i} = c_{2i} - c_{2i+1} = L$ for all $i \in [n-1]$ where $L = 3Cn/2 + 7/4 + 1$. Furthermore, we let $b_1 - s = S$, where $S > 0$ is specified later. The node u is placed so that $u - z = b_1 - s + Cn + 0.5$, where C is a constant that we can set to $C = 3$. There are $4n + 3$ agents:

- Two agents h_1, h_2: These agents have speed 1. Agent h_1 is initially at node s and can traverse the interval $[s, b_1]$. Agent h_2 is initially at node u and can traverse the interval $[z, u]$.
- $2n$ agents $p_1, p_2, \ldots, p_{2n}$: These agents are initially at node z; each agent p_i has speed $v_{p_i} = \frac{1}{C + x_i/M}$ where $M = \sum_{i \in [2n]} x_i$. Agents p_{2i-1} and p_{2i} for $i \in [n]$ can traverse the interval $[b_{2i-1}, c_{2i-1}]$.

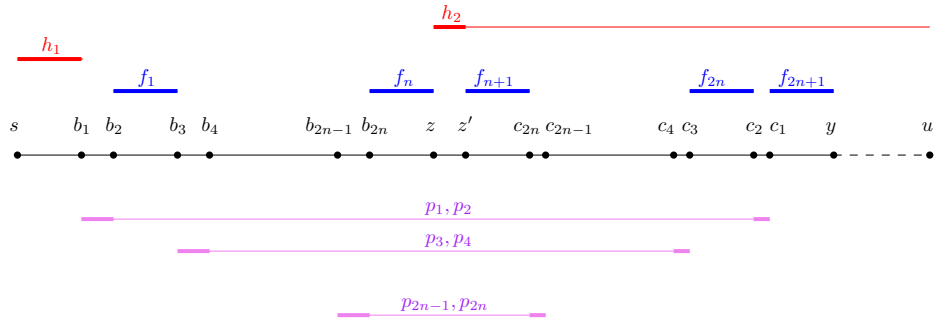


Figure 2 Instance of DDT_N on a path.

- $2n+1$ agents $f_1, f_2, \ldots, f_{2n}, f_{2n+1}$: These agents have infinite speed. Each agent f_i for $i < n$ is initially at node b_{2i} and can traverse the interval $[b_{2i}, b_{2i+1}]$; agent f_n is initially at node b_{2n} and can traverse the interval $[b_{2n}, z]$; agent f_{n+1} is initially at node z' and can traverse the interval $[z', c_{2n}]$; each agent f_i for $n+1 < i \leq 2n$ is initially at node $c_{2(2n+1-i)+1}$ and can traverse the interval $[c_{2(2n+1-i)+1}, c_{2(2n+1-i)}]$; agent f_{2n+1} is initially at node c_1 and can traverse the interval $[c_1, y]$.

The goal is to deliver a package from s to y as quickly as possible. We claim that there is a schedule with delivery time at most $T = S + 3Cn/2 + 7/4$ if and only if the original instance of EOP is a yes-instance.

First, assume that the instance of EOP is a yes-instance with partition (X_1, X_2). Let (P_1, P_2) be the corresponding partition of the set containing the $2n$ agents p_i. We construct a delivery schedule for the package as follows (the colors we mention refer to those shown in Figure 2). Until time $S = b_1 - s$, the agent h_1 carries the package to b_1, and all agents p_i pick a bold purple interval and arrive at its left endpoint. This is done in such a way that agent p_i picks its left bold purple interval (the interval of length 1 at the left end of its range) if $p_i \in P_1$, and its right bold purple interval (the interval of length $\frac{1}{2}$ at the right end of its range) otherwise. To guarantee that any agent p_i arrives at the left endpoint of its interval by time S, we set $S = \max_{i \in [n]} \frac{(n+1-i)(L+1)}{\min\{p_{2i-1}, p_{2i}\}}$. Next, the package travels from b_1 to z by always being alternatingly carried by an agent p_i over a bold purple interval of length 1 and an infinite speed agent f over a blue interval of length L. The package thus reaches z at time $S + Cn + 1/2$ (because $\sum_{x_i \in X_1} x_i/M = 1/2$). As $u - z = S + Cn + 1/2$, the agent h_2 arrives at z at exactly the same time. Then, the agent h_2 carries the package from z to z' in time 1. After that, the package is alternatingly carried by an infinite speed agent and an agent p_i until it reaches y, taking time $Cn/2 + 1/4$. The total delivery time is $S + Cn + 1/2 + 1 + Cn/2 + 1/4 = S + 3Cn/2 + 7/4 = T$ as required.

Now consider the case that the original EOP instance is a no-instance. Assume that there exists a delivery schedule with delivery time $T = S + 3Cn/2 + 7/4$. First, we claim that neither an agent p_i nor the agent h_2 can carry the package over an interval of length L (these are the intervals $[b_{2n}, z]$, $[z', c_{2n}]$, $[c_1, y]$, and the intervals $[b_{2i}, b_{2i+1}]$, $[c_{2i+1}, c_{2i}]$ for any $i \in [n-1]$). Otherwise, the delivery time is larger than $S + L > S + 3Cn/2 + 7/4$ because these agents have speed at most 1 and these blue intervals have length $L = 3Cn/2 + 7/4 + 1$. Therefore, it is clear that each of the $2n$ agents p_i and the agent h_2 will carry the package over exactly one of the following $2n+1$ intervals: the interval $[b_1, b_2]$, and the $2n$ intervals that lie between two consecutive blue intervals. Next, we claim that h_2 must carry the package over the interval $[z, z']$ with length 1. Otherwise, h_2 would have to carry the package over an interval $[c_{2i}, c_{2i-1}]$ with length $1/2$, and an agent p_i with speed less than $1/C$ would

have to carry the package through $[z, z']$ with length 1 instead. Even if we assume that all agents p_i have the faster speed $1/C$, the resulting schedule would have delivery time at least $S + nC + C + (n-1)C/2 + 1/2 = S + 3Cn/2 + C/2 + 1/2$, which is larger than $T = S + 3Cn/2 + 7/4$ as $C = 3 > 5/2$.

This implies that agent h_2 carries the package over $[z, z']$ and, for each $i \in [n]$, one agent among $\{p_{2i-1}, p_{2i}\}$ carries the package on a bold purple interval to the left of z and the other agent carries the package on a bold purple interval to the right of z'. Let the resulting partition of the agents p_i be (P_1, P_2), and let the corresponding partition of the original EOP instance be (X_1, X_2). Suppose the package arrives at node z at time $S + W$. Since the instance of EOP is a no-instance, either $W > Cn + 1/2$ or $W < Cn + 1/2$ holds. As the agents p_i that carry the package to the left of z take time W in total, the agents p_i that carry the package over intervals to the right of z' take time $\frac{2Cn+1-W}{2}$ in total. Consider the two cases:

- $W > Cn + 1/2$. As agent h_2 carries the package from z to z', the delivery time is $S + W + 1 + \frac{2Cn+1-W}{2} = S + Cn + W/2 + 3/2 > S + 3Cn/2 + 7/4 = T$, a contradiction.
- $W < Cn + 1/2$. As agent h_2 carries the package from z to z', the package must wait at z until time $S + Cn + 1/2$ when h_2 reaches z. The delivery time is $S + Cn + 1/2 + 1 + \frac{2Cn+1-W}{2} = S + 2Cn + 2 - W/2 > S + 2Cn + 2 - (Cn + 1/2)/2 = S + 3Cn/2 + 7/4 = T$, a contradiction.

Finally, we observe that in the constructed instance of DDT, the availability of edge handovers has no impact on the existence of a schedule with delivery time $S + 3Cn/2 + 7/4$. Therefore, the reduction establishes *NP*-hardness of both $\mathsf{DDT_N}$ and $\mathsf{DDT_E}$. ◀

6.2 Algorithms for drone delivery on a tree

Now, we show that all variants of the drone delivery problem can be solved optimally in polynomial time if the graph is a tree and all agents have the same speed (for minimizing the delivery time) or the same energy consumption rate (for minimizing the total energy consumption). In fact, the algorithms extend to the case of general graphs if the subgraph $G_a = (V_a, E_a)$ of each agent is *isometric*, i.e., it satisfies the following condition: For any two nodes $u, v \in V_a$, the length of the shortest u-v path in G_a is equal to the length of the shortest u-v path in G. If the given graph is a tree, the subgraph G_a of every agent a is necessarily isometric because we assume that G_a is connected and there is a unique path between any two nodes in a tree.

If all agents have the same speed (for DDT) or the same consumption rate (for DDC), handovers at internal points of edges can never improve the objective value. Therefore, we only need to consider $\mathsf{DDT_N}$ and $\mathsf{DDC_N}$ in the following. The crucial ingredient of our algorithms for the case of isometric subgraphs G_a is:

▶ **Lemma 12.** *Consider a delivery schedule $\mathcal{S}$ that may use an arbitrary number of copies of each agent. If the subgraph of each agent is isometric and all agents have the same speed (or the same energy consumption rate), then $\mathcal{S}$ can be transformed in time $O(k(m + n \log n))$ into a schedule in which each agent is used at most once without increasing the delivery time (or the energy consumption).*

Proof. Consider the first agent a that is used at least twice in $\mathcal{S}$. Assume that it carries the package from u_{i-1} to u_i in its first trip and from u_{j-1} to u_j in its last trip. Change the schedule so that a carries the package from u_{i-1} to u_j, and discard the trips by agents $i + 1, \ldots, j - 1$. As G_a is isometric, the agent a carries the package from u_{i-1} to u_j along

a shortest path in G, and hence neither the time (in case of equal speed) nor the energy consumption (in case of equal energy consumption rate) increase by this modification. Repeat the modification step until every agent is used at most once.

There are at most $k-1$ modification steps, and each of them can be implemented in $O(m + n \log n)$ time using Dijkstra's algorithm with Fibonacci heaps [10]. ◀

▶ **Theorem 13.** DDT_N *(and* DDT_E*) can be solved optimally in time* $O(k(n \log n + m))$ *if all agents have the same speed and the subgraph of every agent is isometric.*

Proof. Compute an optimal delivery schedule that may use multiple copies of each agent using Lemma 6 and then apply Lemma 12. ◀

▶ **Theorem 14.** DDC_N *(and* DDC_E*) can be solved optimally in time* $O(nk^2 + n^2k)$ *if all agents have the same speed and the subgraph of every agent is isometric.*

Proof. Compute an optimal delivery schedule that may use multiple copies of each agent using Lemma 9 and then apply Lemma 12. ◀

The problem variants without initial positions can also be solved optimally in polynomial time: We simply compute a shortest s-y-path P and place on each edge e of P a copy of an arbitrary agent in $A(e)$ and then apply Lemma 12.

For the special case where G is a tree, the running-time for DDT_N can be improved to $O(kn)$ by using a simple algorithm that can even be implemented in a distributed way: The package acts as a magnet, and each agent moves towards the package until it meets the package and then follows it (or carries it) towards y as long as its range allows. When several agents are at the same location as the package, the one whose range extends furthest towards y carries the package. We refer to the full version [11] for details.

7 Conclusions

In this paper we have studied drone delivery problems in a setting where the movement area of each drone is restricted to a subgraph of the whole graph. For DDT, we have presented a strong inapproximability result and given a matching approximation algorithm. For DDC, we have shown *NP*-hardness and presented a 2-approximation algorithm. For the interesting special case of a path, we have shown that DDT is *NP*-hard if the agents can have different speeds. For trees (or, more generally, the case where the subgraph of each agent is isometric), we have shown that all problem variants can be solved optimally in polynomial time if the agents have the same speed or the same energy consumption.

We leave open the complexity of DDC on a path. For the case without initial positions, the complexity of both DDC and DDT on a path remains open. For DDT with initial positions on a path, a very interesting question is how well the problem can be approximated.

References

1 Amazon Staff. Amazon Prime Air prepares for drone deliveries. https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries, 13 june 2022. Accessed: 2022-06-19.

2 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matús Mihalák. Collaborative delivery with energy-constrained mobile robots. *Theor. Comput. Sci.*, 810:2–14, 2020. doi:10.1016/j.tcs.2017.04.018.

3 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna. Energy-efficient delivery by heterogeneous mobile agents. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.10.

4 Andreas Bärtschi, Daniel Graf, and Matús Mihalák. Collective fast delivery by energy-efficient agents. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, pages 56:1–56:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.56.

5 Andreas Bärtschi and Thomas Tschager. Energy-efficient fast delivery by mobile agents. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, volume 10472 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2017. doi:10.1007/978-3-662-55751-8_8.

6 Iago A. Carvalho, Thomas Erlebach, and Kleitos Papadopoulos. On the fast delivery problem with one or two packages. *J. Comput. Syst. Sci.*, 115:246–263, 2021. doi:10.1016/j.jcss.2020.09.002.

7 Jérémie Chalopin, Shantanu Das, Yann Disser, Arnaud Labourel, and Matús Mihalák. Collaborative delivery on a fixed path with homogeneous energy-constrained agents. *Theor. Comput. Sci.*, 868:87–96, 2021. doi:10.1016/j.tcs.2021.04.004.

8 Jérémie Chalopin, Shantanu Das, Matúš Mihalák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2013)*, volume 8243 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2013. doi:10.1007/978-3-642-45346-5_9.

9 Jérémie Chalopin, Riko Jacob, Matúš Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In *41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8573 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 2014. doi:10.1007/978-3-662-43951-7_36.

10 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press, 2022. URL: https://mitpress.mit.edu/books/introduction-algorithms-fourth-edition.

11 Thomas Erlebach, Kelin Luo, and Frits C.R. Spieksma. Package delivery using drones with restricted movement areas. *CoRR*, abs/2209.12314, 2022. doi:10.48550/arXiv.2209.12314.

12 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Parameterized Approximation Algorithms for TSP

Jianqi Zhou
School of Computer Science and Technology, Shandong University, Qingdao, China

Peihua Li
School of Computer Science and Technology, Shandong University, Qingdao, China

Jiong Guo
School of Computer Science and Technology, Shandong University, Qingdao, China

Abstract

We study the Traveling Salesman problem (TSP), where given a complete undirected graph $G = (V, E)$ with n vertices and an edge cost function $c : E \mapsto R_{\geqslant 0}$, the goal is to find a minimum-cost cycle visiting every vertex exactly once. It is well-known that unless P = NP, TSP cannot be approximated in polynomial time within a factor of $\rho(n)$ for any computable function ρ, while the metric case of TSP, that the edge cost function satisfies the $\triangle$-inequality, admits a polynomial-time 1.5-approximation. We investigate TSP on general graphs from the perspective of parameterized approximability. A parameterized ρ-approximation algorithm returns a ρ-approximation solution in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function and k is a parameter of the input I. We introduce two parameters, which measure the distance of a given TSP-instance from the metric case, and achieve the following two results:

- A 3-approximation algorithm for TSP in $O((3k_1)!\, 8^{k_1} \cdot n^2 + n^3)$ time, where k_1 is the number of triangles in which the edge costs violate the $\triangle$-inequality.
- A 3-approximation algorithm for TSP in $O(n^{O(k_2)})$ time and a $(6k_2+9)$-approximation algorithm for TSP in $O(k_2^{O(k_2)} \cdot n^3)$ time, where k_2 is the minimum number of vertices, whose removal results in a metric graph.

To our best knowledge, the above algorithms are the first non-trivial parameterized approximation algorithms for TSP on general graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases FPT-approximation algorithms, the Traveling Salesman problem, the triangle inequality, fixed-parameter tractability, metric graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.50

Funding The work has been supported by the National Natural Science Foundation of China (No. 61772314, 61761136017, and 62072275).

Acknowledgements The authors thank the reviewers for their valuable comments and constructive suggestions.

1 Introduction

The Traveling Salesman problem (TSP) is one of the most prominent combinatorial optimization problems, where given a complete undirected graph $G = (V, E)$ with n vertices and an edge cost function $c : E \mapsto R_{\geqslant 0}$, the goal is to find a minimum-cost cycle visiting every vertex exactly once. TSP is NP-hard [16, 23]. Bellman [4] and Held and Karp [21] independently proposed an exact algorithm for TSP in $O(2^n)$ time by the dynamic programming technique.

Approximation algorithms. A ρ-approximation algorithm for a minimization problem returns a feasible solution A with $c(A) \leq \rho \cdot c(\text{opt})$ in polynomial time, where opt is an optimal solution. Unfortunately, TSP cannot be approximated in polynomial time within a

factor of $\rho(n)$ for any computable function ρ, unless P = NP [34]. There is a special case of TSP called the metric TSP, $\triangle$TSP for short, where $c(u,v) \leq c(u,w) + c(w,v)$ for every $u, v, w \in V$. Hereby, $c(u,v)$ is the cost of the edge between the vertices u and v. $\triangle$TSP is NP-hard [23], but there is a 3/2-approximation algorithm for $\triangle$TSP in $O(n^3)$ time, proposed independently by Christofides [11] and Serdyukov [35]. This has been the best approximation ratio for more than 40 years until recently Karlin, Klein and Gharan [22] gave a randomized $(3/2 - \epsilon)$-approximation algorithm for $\triangle$TSP for some constant $\epsilon > 10^{-36}$. There is a generalization of $\triangle$TSP by relaxing the $\triangle$-inequality to the parameterized $\triangle$-inequality, i.e., for some $\tau \geq 1$, $c(u,v) \leq \tau[c(u,w) + c(w,v)]$ for every $u, v, w \in V$. The parameterized $\triangle$-inequality is denoted by $\triangle_\tau$-inequality and TSP satisfying the $\triangle_\tau$-inequality is denoted by $\triangle_\tau$TSP. Andreae and Bandelt [2] gave a $(3\tau^2 + \tau)/2$-approximation algorithm for $\triangle_\tau$TSP, and then Bender and Chekuri [5] improved the approximation ratio to 4τ. There are some other improved approximation algorithms for $\triangle_\tau$TSP with $\tau \leq 13/3$, e.g., [1, 6, 30]. These approximation results represent no conflict with the polynomial-time inapproximability of TSP, since τ is not a constant and even not a function of n. Mohan [29] designed a 7/2-approximation algorithm for Biased-TSP, where all vertices can be partitioned into two parts such that every triangle with three vertices in the same part satisfies the $\triangle$-inequality, while the triangles with vertices in different parts may violate the $\triangle$-inequality. However, not all TSP-instances admit such a partition.

Parameterized algorithms. An exact algorithm with running time of the form $f(k) \cdot |I|^{O(1)}$ is called a parameterized algorithm and a problem admitting a parameterized algorithm is fixed-parameter tractable (FPT). The idea is to limit the exponential running time to the parameter k instead of the input size $|I|$. Deineko and Hoffmann [12] studied TSP in the 2-dimensional Euclidean plane called 2DTSP that is NP-hard [32], and gave a parameterized algorithm in $O(2^k k^2 \cdot n)$ time for 2DTSP with k inner points, where a point is inner if it lies strictly inside the convex hull of the input. This was then improved to $O(k^{11\sqrt{k}} k^{1.5} \cdot n^3)$ by Knauer and Spillner [26].

Parameterized approximation algorithms. A parameterized ρ-approximation algorithm returns a ρ-approximation solution in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function and k is a parameter of the input I. Many prominent NP-hard problems have been studied from the perspective of parameterized approximability, for instance, Independent Set [8, 13, 18, 19, 27, 28] and Dominating Set [9, 10, 33]. Concerning TSP, Arora [3] proposed a $(1+\epsilon)$-approximation algorithm for TSP in the k-dimensional Euclidean space running in $O(n(\log n)^{O(\sqrt{k}/\epsilon)^{k-1}})$ time for any $\epsilon \geq 0$, which can be upper bounded by $O(k^{O(\sqrt{k}/\epsilon)^{k-1}} \cdot n^2)$ as shown in [15, 24]. Some parameterized approximation algorithms for the metric TSP with other dimensional parameters are introduced in [15, 17]. Bockenhauer and Hromkovic [7] studied TSP with deadlines, DLTSP for short, where k vertices must be visited before a given time. DLTSP cannot be approximated in polynomial time within a factor of $\rho(n)$ for any polynomial function ρ. Bockenhauer and Hromkovic [7] gave a parameterized 5/2-approximation algorithm for the metric DLTSP running in $O(k!\,k + n^3)$ time, where k is the number of deadline vertices. We refer the readers to the survey [15, 28] of the parameterized approximation algorithms.

Our contributions. We aim at designing parameterized approximation algorithms for TSP on general graphs. Motivated by the polynomial-time inapproximability of TSP and the 3/2-approximation algorithm for $\triangle$TSP, we propose a concept for designing parameterized

approximation algorithms, the so-called "distance from approximability", that is, introducing a parameter measuring the distance between the inapproximable and approximable cases and designing approximation algorithms with the exponential running time restricted to the parameter. A similar concept called "distance from triviality" has been used in the parameterized algorithmics [20]. Here, we introduce two parameters to measure the distance between a general TSP-instance and a metric TSP-instance. First, we consider the number k_1 of triangles in a general TSP-instance, which do not satisfy the $\triangle$-inequality, that is, the triangles in which the edge costs violate the $\triangle$-inequality. Given a TSP-instance, the value of k_1 can be easily computed in $O(n^3)$ time. Using k_1 as parameter, we achieve an approximation of TSP with an approximation factor of 3 and a running time of $O((3k_1)!\,8^{k_1} \cdot n^2 + n^3)$. Our second parameter k_2 is set equal to the minimum number of vertices, whose removal turns a general instance into a metric instance. Clearly, we can apply a search tree [14, 31] to determine the value of k_2 in $O(3^{k_2} \cdot n^3)$ time. Moreover, we always have $k_2 \leq k_1$. With k_2 as parameter, we first present an algorithm running in $O(n^{O(k_2)})$ time, returning a TSP-solution with an approximation factor of 3 and then a parameterized approximation algorithm with a factor of $6k_2+9$ and a running time of $O(k_2^{O(k_2)} \cdot n^3)$. To our best knowledge, the above algorithms are the first non-trivial parameterized approximation algorithms for general TSP. We are also confident that the "distance from approximability" concept might be useful for other problems.

2 Preliminaries

We introduce some basic definitions and notations. We consider a simple undirected complete graph $G = (V, E)$, with an edge cost function $c : E \mapsto R_{\geqslant 0}$. The graph G is metric if $c(u, v) \leq c(u, w) + c(w, v)$ for every $u, v, w \in V$. A triangle $\triangle(u, v, w)$ is called "violating" if it does not satisfy the $\triangle$-inequality, that is, at least one of $c(u, v) \leq c(u, w) + c(w, v)$, $c(u, w) \leq c(u, v) + c(v, w)$ and $c(v, w) \leq c(v, u) + c(u, w)$ does not hold. A set of vertices whose removal transforms a non-metric graph into a metric graph, is called a violating vertex set. For an edge set $E' \subseteq E$, the cost of E', denoted by $c(E')$, is the total cost of its edges, and the vertex set of E' is denoted by $V(E')$. For a vertex set $V' \subseteq V$, $E(V')$ is the set of edges that connect two vertices in V', and $G[V'] = (V', E(V'))$ is the subgraph of G induced by V'. For a positive integer i, set $[i] = \{1, \cdots, i\}$.

An acyclic graph consisting of t connected components is called a t-forest. A spanning subgraph of G, which is a t-forest, is called a spanning t-forest of G. A spanning t-forest having the minimum cost is called a t-minimum spanning forest (t-MSF) of G.

3 The number of violating triangles as parameter

In this section, we consider a TSP-instance with k_1 triangles violating the $\triangle$-inequality. The parameter k_1 can be computed in $O(n^3)$ time by checking all triangles in the input. We call a vertex "bad" if it is in one of the k_1 triangles, and we denote by V^b the set of all bad vertices, $|V^b| \leq 3k_1$. The remaining vertices are called "good" and we denote by V^g the set of good vertices. For $b_u, b_v, b_w \in V^b$, the triangle formed by these three vertices $\triangle(b_u, b_v, b_w)$ might violate the $\triangle$-inequality, that is, $c(b_u, b_w) + c(b_w, b_v)$ might be arbitrarily less than $c(b_u, b_v)$.

Our algorithm consists mainly of two steps. The first step "guesses" the occurrences of bad vertices in an optimal TSP-solution, that is, the occurrence order of bad vertices and the "gaps" between bad vertices, where the good vertices should be inserted to form the

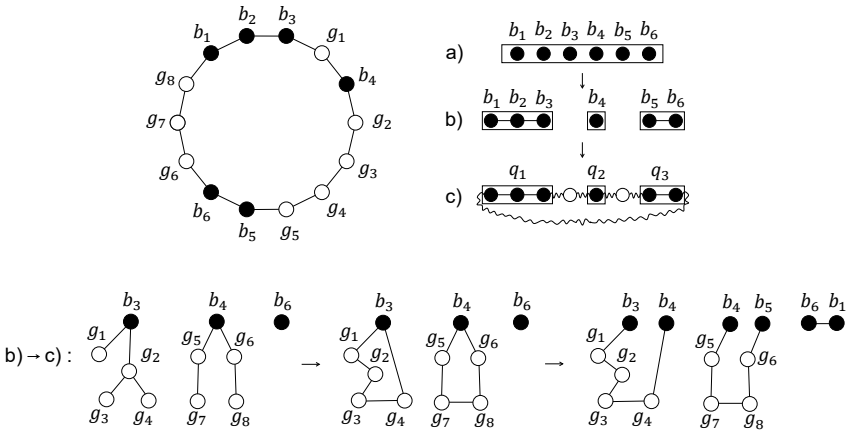


Figure 1 An illustration of Algorithm 1. b_i's are the bad vertices, g_i's are the good vertices and q_i's are the subsets of the bad vertices. Top left: the cycle represents an optimal TSP-solution. Top right: a) a permutation of bad vertices; b) a partition of bad vertices respecting the order of the permutation in a); c) inserting good vertices into the gaps between bad vertices. Bottom: the first graph is the MSF rooted at the end-vertices of the subsets of the partition; the second graph is a collection of cycles (and possibly isolated vertices) obtained by doubling the edges of the MSF in the first graph; the third graph is the paths with only good internal vertices connecting consecutive subsets (and possibly edges directly connecting consecutive subsets).

optimal TSP-solution. This can be done by enumerating all possible permutations of bad vertices and for each permutation, all partitions of bad vertices, which respect the order of the permutation. This means that the bad vertices in one subset of the partition occur together in the optimal TSP-solution and obey the order of the permutation. The subsets also occur in the order of the permutation. The optimal TSP-solution contains at least one good vertex between two consecutive subsets. The second step computes paths of good vertices to fill in the gaps between the subsets in each partition of permutations. Hereby, we compute the minimum-cost spanning forest rooted at the end-vertices of the subsets of the partition and transform it to a collection of paths with only good internal vertices (and possibly edges directly connecting consecutive subsets). See Figure 1 for an illustration.

We introduce some notations to describe the bad vertices. A "bad chain" denoted by $q = (b_1, b_2, \ldots, b_l)$ with $l \geq 1$ is one bad vertex or a path consisting of distinct bad vertices, where $b_i \in V^b$ for $i \in [l]$ and there is an edge connecting b_i and b_{i+1} for each $i \in [l-1]$. We use $b_s(q)$, $b_e(q)$ to denote the starting and ending vertices, respectively, and use $c(q)$ to denote the total cost of the edges in q. In particular, if a bad chain q consists of only one bad vertex b_1, then $b_s(q) = b_1$, $b_e(q) = b_1$, and $c(q) = 0$.

3.1 The algorithm and time complexity

The algorithm is as follows (see Figure 1 for an illustration).

Algorithm 1 Ratio-3 approximation algorithm for parameterization with k_1.

1. Compute the k_1 triangles violating the $\triangle$-inequality, the set of bad vertices V^b and the set of good vertices V^g.
2. Enumerate all possible permutations of bad vertices. For each permutation, enumerate all possible partitions of bad vertices into t subsets for each $t \in [|V^b|]$, respecting the corresponding permutation order. For each t-partition, do the following:

(a) Connect the bad vertices in each subset of the partition in the order of the permutation, resulting in t bad chains $Q = (q_1, \ldots, q_t)$, ordered according to their orders in the permutation.
(b) Compute a t-minimum spanning forest (t-MSF) $F = (T_1, \ldots, T_t)$ of $G[V^g \cup \{b_e(q_i) | i \in [t]\}]$ rooted at $\{b_e(q_i) | i \in [t]\}$.
(c) Double the edges of T_i's. Compute an Euler tour of each T_i and shortcut repeated vertices in the Euler tour, resulting in C_i.
(d) For each $i \in [t]$, do the following: if $C_i = (b_e(q_i))$, then define $p_i = (b_e(q_i), b_s(q_{i+1}))$; if $C_i = (b_e(q_i), g_{i_1}, \ldots, g_{i_{l_i}}, b_e(q_i))$ with $l_i \geq 1$, then define $p_i = (b_e(q_i), g_{i_1}, \ldots, g_{i_{l_i}}, b_s(q_{i+1}))$. Hereby, $q_{t+1} = q_1$.
(e) Connect $b_e(q_i)$ and $b_s(q_{i+1})$ in this t-partition by p_i for each $i \in [t]$, and obtain a TSP-solution $A = (q_1, p_1, q_2, p_2, \ldots, q_t, p_t)$.

3. Return the solution $A^{\min}$ with the minimum cost among all enumeration cases.

▶ **Lemma 1.** *A minimum spanning forest of $G[V \cup V']$ rooted at V' can be computed in $O(|V \cup V'|^2)$ time.*

Proof. First, we introduce a root vertex and connect it to all vertices in V' by zero-cost edges to get a new graph $\tilde{G}$, transforming computing a minimum spanning forest rooted at V' into computing a minimum spanning tree (MST).

An MST T of $\tilde{G}$ can be found in $O(|V(\tilde{G})|^2)$ time. If T does not contain all edges connecting the root vertex to the vertices in V', then we can add those missing edges, remove other edges incident to the vertices in V' to ensure acyclic, and obtain a new MST T' without increasing costs. By removing the root vertex and all edges incident to it in T', we get a minimum spanning forest of $G[V \cup V']$ rooted at V' in $O(|V \cup V'|^2)$ time. ◀

▶ **Lemma 2.** *Algorithm 1 runs in $O((3k_1)!\, 8^{k_1} \cdot n^2 + n^3)$ time.*

Proof. Step 1 computes k_1 triangles violating the $\triangle$-inequality in $O(n^3)$ time. The number of t-partitions of permutations enumerated in Step 2 is $|V^b|!\, 2^{|V^b|} = O((3k_1)!\, 8^{k_1})$. For each possible t-patition, Step 2(a) to Step 2(d) take $O(n^2)$ time by Lemma 1. Hence, Algorithm 1 runs in $O((3k_1)!\, 8^{k_1} \cdot n^2 + n^3)$ time. ◀

3.2 Analysis of approximation factor

An optimal TSP-solution opt can be decomposed into an ordered collection of $2t^{\text{opt}}$ many paths $q_1^{\text{opt}}, p_1^{\text{opt}}, q_2^{\text{opt}}, \ldots, p_{t^{\text{opt}}}^{\text{opt}}$, with q_i^{opt} being a bad chain and $p_i^{\text{opt}} = (b_e(q_i^{\text{opt}}), g_{i_1}^{\text{opt}}, \ldots, g_{i_{l_i^{\text{opt}}}}^{\text{opt}}, b_s(q_{i+1}^{\text{opt}}))$ being the path connecting $b_e(q_i^{\text{opt}})$ and $b_s(q_{i+1}^{\text{opt}})$ with only good internal vertices. Here $l_i^{\text{opt}} \geq 1$, $q_{t^{\text{opt}}+1}^{\text{opt}} = q_1^{\text{opt}}$. Clearly, $t^{\text{opt}} \leq 3k_1$. See Figure 1 for an example: $t^{\text{opt}} = 3$, $q_1^{\text{opt}} = (b_1, b_2, b_3)$, $q_2^{\text{opt}} = (b_4)$, $q_3^{\text{opt}} = (b_5, b_6)$. Moreover, $p_1^{\text{opt}} = (b_3, g_1, b_4)$ where $l_1^{\text{opt}} = 1$, connects $b_e(q_1) = b_3$ and $b_s(q_2) = b_4$. Further, $p_2^{\text{opt}} = (b_4, g_2, g_3, g_4, g_5, b_5)$ and $p_3^{\text{opt}} = (b_6, g_6, g_7, g_8, b_1)$. Then, $c(\text{opt}) = \sum_{i=1}^{t^{\text{opt}}} c(q_i^{\text{opt}}) + \sum_{i=1}^{t^{\text{opt}}} c(p_i^{\text{opt}})$.

We enumerate all partitions of every permutation of bad vertices in Step 2 of Algorithm 1. Thus, there must exist an enumeration case α in Step 2, where we have t^{opt} bad chains which contain the same bad vertices in the same order as in opt. Therefore, the solution A^α returned in Step 2(e) of this case differs from opt only in the paths connecting the bad chains. In the following, we give an analysis of these paths for both opt and A^α.

▶ **Lemma 3.** *For each $i \in [t^{opt}]$, $c(p_i^{opt}) \geq c(b_e(q_i^{opt}), b_s(q_{i+1}^{opt}))$, where $q_{t^{opt}+1}^{opt} = q_1^{opt}$.*

Proof. Note that a triangle containing at least one good vertex satisfies the $\triangle$-inequality. Thus, $\triangle(b_e(q_i^{\text{opt}}), g_{i_1}^{\text{opt}}, b_s(q_{i+1}^{\text{opt}}))$ and $\triangle(g_{i_1}^{\text{opt}}, g_{i_{l_i^{\text{opt}}}}^{\text{opt}}, b_s(q_{i+1}^{\text{opt}}))$ satisfy the $\triangle$-inequality. Here $l_i^{\text{opt}} \geq 1$, $q_{t^{\text{opt}}+1}^{\text{opt}} = q_1^{\text{opt}}$. Thus, for each $i \in [t^{\text{opt}}]$, we have

$$\begin{aligned}
c(p_i^{\text{opt}}) &= c(b_e(q_i^{\text{opt}}), g_{i_1}^{\text{opt}}) + c(g_{i_1}^{\text{opt}}, \ldots, g_{i_{l_i^{\text{opt}}}}^{\text{opt}}) + c(g_{i_{l_i^{\text{opt}}}}^{\text{opt}}, b_s(q_{i+1}^{\text{opt}})) \\
&\geq c(b_e(q_i^{\text{opt}}), g_{i_1}^{\text{opt}}) + c(g_{i_1}^{\text{opt}}, g_{i_{l_i^{\text{opt}}}}^{\text{opt}}) + c(g_{i_{l_i^{\text{opt}}}}^{\text{opt}}, b_s(q_{i+1}^{\text{opt}})) \\
&\geq c(b_e(q_i^{\text{opt}}), g_{i_1}^{\text{opt}}) + c(g_{i_1}^{\text{opt}}, b_s(q_{i+1}^{\text{opt}})) \\
&\geq c(b_e(q_i^{\text{opt}}), b_s(q_{i+1}^{\text{opt}})).
\end{aligned}$$

◀

In Step 2(b) of the case α, we obtain a t^{opt}-MSF $F^\alpha = (T_1^\alpha, \ldots, T_{t^{\text{opt}}}^\alpha)$ of $G[V^g \cup \{b_e(q_i^{\text{opt}}) | i \in [t^{\text{opt}}]\}]$ rooted at $\{b_e(q_i^{\text{opt}}) | i \in [t^{\text{opt}}]\}$. Then, let C_i^α for $i \in [t^{\text{opt}}]$ be the resulting cycle (or the isolated vertex) in Step 2(c) of the case α. Let p_i^α for $i \in [t^{\text{opt}}]$ be the path returned in Step 2(d) of the case α. Then $c(A^\alpha) = \sum_{i=1}^{t^{\text{opt}}} c(q_i^{\text{opt}}) + \sum_{i=1}^{t^{\text{opt}}} c(p_i^\alpha)$. The following lemma establishes the relation of the cost of p_i^α with the cost of p_i^{opt}.

▶ **Lemma 4.** *For each $i \in [t^{opt}]$, $c(p_i^\alpha) \leq c(p_i^{opt}) + 2c(T_i^\alpha)$.*

Proof. For each $i \in [t^{\text{opt}}]$, if $C_i^\alpha = (b_e(q_i^{\text{opt}}))$, then as defined in Step 2(d) of the case α, $p_i^\alpha = (b_e(q_i^{\text{opt}}), b_s(q_{i+1}^{\text{opt}}))$, thus $c(p_i^\alpha) \leq c(p_i^{\text{opt}})$ by Lemma 3.
Otherwise, $C_i^\alpha = (b_e(q_i^{\text{opt}}), g_{i_1}^\alpha, \ldots, g_{i_{l_i^\alpha}}^\alpha, b_e(q_i^{\text{opt}}))$ with $l_i^\alpha \geq 1$, Step 2(d) of the case α defines $p_i^\alpha = (b_e(q_i^{\text{opt}}), g_{i_1}^\alpha, \ldots, g_{i_{l_i^\alpha}}^\alpha, b_s(q_{i+1}^{\text{opt}}))$. Note that a triangle containing at least one good vertex satisfies the $\triangle$-inequality, and as shown in Lemma 3, we have

$$\begin{aligned}
c(p_i^\alpha) &= c(b_e(q_i^{\text{opt}}), g_{i_1}^\alpha, \ldots, g_{i_{l_i^\alpha}}^\alpha) + c(g_{i_{l_i^\alpha}}^\alpha, b_s(q_{i+1}^{\text{opt}})) \\
&\leq c(b_e(q_i^{\text{opt}}), g_{i_1}^\alpha, \ldots, g_{i_{l_i^\alpha}}) + c(g_{i_{l_i^\alpha}}^\alpha, b_e(q_i^{\text{opt}})) + c(b_e(q_i^{\text{opt}}), b_s(q_{i+1}^{\text{opt}})) \\
&= c(C_i^\alpha) + c(b_e(q_i^{\text{opt}}), b_s(q_{i+1}^{\text{opt}})) \\
&\leq 2c(T_i^\alpha) + c(p_i^{\text{opt}}).
\end{aligned}$$

◀

Next, we compare the cost of opt with the cost of F^α, the following lemma is easy to see.

▶ **Lemma 5.** *$c(opt) \geq c(F^\alpha)$.*

Proof. We remove all bad vertices not in $\{b_e(q_i^{\text{opt}}) | i \in [t^{\text{opt}}]\}$ and all edges incident to them from opt. Then for each $i \in [t^{\text{opt}}]$, if q_i^{opt} consists of only one bad vertex, then we remove the edge connecting it and its preceding good vertex. The above removal process results in a spanning t^{opt}-forest of $G[V^g \cup \{b_e(q_i^{\text{opt}}) | i \in [t^{\text{opt}}]\}]$ rooted at $\{b_e(q_i^{\text{opt}}) | i \in [t^{\text{opt}}]\}$. Thus, $c(\text{opt}) \geq c(F^\alpha)$. ◀

Finally, we arrive at proving the main result of this section.

▶ **Theorem 6.** *Algorithm 1 is a parameterized 3-approximation algorithm for TSP running in $O((3k_1)!\, 8^{k_1} \cdot n^3)$ time, where k_1 is the number of triangles in which edge costs violate the $\triangle$-inequality.*

Proof. Algorithm 1 returns $A^{\min}$ in $O((3k_1)!\, 8^{k_1} \cdot n^2 + n^3)$ time by Lemma 2 and clearly provides a cycle over all vertices. The approximation factor follows from Lemmas 4 and 5: $c(A^{\min}) \leq c(A^\alpha) = \sum_{i=1}^{t^{\text{opt}}} c(q_i^{\text{opt}}) + \sum_{i=1}^{t^{\text{opt}}} c(p_i^\alpha) \leq \sum_{i=1}^{t^{\text{opt}}} c(q_i^{\text{opt}}) + \sum_{i=1}^{t^{\text{opt}}} c(p_i^{\text{opt}}) + 2\sum_{i=1}^{t^{\text{opt}}} c(T_i^\alpha) = c(\text{opt}) + 2c(F^\alpha) \leq c(\text{opt}) + 2c(\text{opt}) = 3c(\text{opt})$. ◀

4 The minimum size of violating vertex sets as parameter

In this section, we consider the minimum size k_2 of violating vertex sets, that is, the minimum number of vertices whose removal transforms a given graph into a metric graph. The violating vertex set with at most k_2 vertices can be found in $O(3^{k_2} \cdot n^3)$ time by the search tree technique [14, 31]. The basic idea is that at least one vertex of every triangle violating the $\triangle$-inequality has to be added to the violating vertex set, which also implies that $k_2 \leq k_1$. We use the bad vertices to refer to the vertices in the violating vertex set and thus, there are k_2 bad vertices. The remaining vertices are called good. A bad chain also consists solely of bad vertices.

In contrast to Section 3, a triangle violating the $\triangle$-inequality might contain only one or two bad vertices, leading to the consequence that the idea of Algorithm 1 does not directly apply to the parameterization of k_2. In Section 3, triangles containing at least one good vertex satisfy the $\triangle$-inequality, which provides the foundation of Lemmas 3 and 4. Notice that, as shown in Lemma 3, the cost of the edge directly connecting two consecutive bad chains in opt does not exceed the cost of the path in opt connecting the same two bad chains. Here, this property does not hold. However, by a closer observation, we can conclude that a path connecting bad chains only involves the starting and ending bad vertices of bad chains. If we can fix the two good vertices, which are the direct neighbors of the bad chains in the optimal TSP-solution, then we can extend bad chains to paths, which are between two good vertices and whose internal vertices are all bad vertices. Then, we use the same strategy as in Steps 2(b)-2(d) in Algorithm 1 to find paths of good vertices to connect these paths, where we deal only with edges between good vertices, to which we can apply the analysis in Section 3. We show in the following that a brute-force way of fixing the direct neighbors of bad chains, which is Algorithm 2, leads to a 3-approximation with the running time of $O(n^{O(k_2)})$, while a more involved selection of the direct neighbors, i.e., Algorithm 3, gives a parameterized approximation with the running time of $O(k_2^{O(k_2)} \cdot n^3)$ but a worse approximation factor $6k_2 + 9$.

To this end, we need the following notations. An "alternating chain" denoted by $h = (g_1, q_1, g_2, \ldots, g_l, q_l, g_{l+1})$ with $l \geq 1$, is a path consisting alternatively of distinct good vertices and bad chains. Here, $g_1, \ldots, g_{l+1} \in V^g$, and $q_1, \ldots, q_l$ are vertex-disjoint bad chains. Moreover, g_{i+1} for $i \in [l-1]$ is connected by edges to $b_e(q_i)$ and $b_s(q_{i+1})$, g_1 is adjacent to $b_s(q_1)$, and g_{l+1} is adjacent to $b_e(q_l)$. We use $g_s(h)$, $g_e(h)$ to denote the starting and ending good vertices, respectively, and use $c(h)$ to denote the total cost of the edges in h, i.e., $c(h) = \sum_{i=1}^{l} c(q_i) + \sum_{i=1}^{l} c(g_i, b_s(q_i)) + \sum_{i=1}^{l} c(b_e(q_i), g_{i+1})$.

4.1 A 3-approximation algorithm

In this section, we use a brute-force strategy to fix the direct good neighbors of the bad chains in the optimal TSP-solution.

Algorithm 2 Ratio-3 approximation algorithm for parameterization with k_2.

1. Compute the violating vertex set of k_2 vertices V^b by the search tree technique and the set of good vertices V^g.
2. Enumerate all possible permutations of bad vertices. For each permutation, enumerate all possible partitions of bad vertices into t_1 subsets for each $t_1 \in [k_2]$, respecting the corresponding permutation order. For each t_1-partition, do the following:
   (a) Connect the bad vertices in each subset of the partition in the order of the permutation, resulting in t_1 bad chains $Q = (q_1, \ldots, q_{t_1})$, ordered according to their orders in the permutation.

(b) For all possible t_1-tuples of pairs of good vertices $((x_1, y_1), (x_2, y_2), \ldots, (x_{t_1}, y_{t_1}))$ satisfying $x_i, y_i \in V^g \setminus \bigcup_{j=1}^{i-1} \{x_j, y_j\}$ for each $i \in [t_1]$: (Note that it might be $x_i = y_i$ for some i.)

(i) Transform q_i for each $i \in [t_1]$ to an alternating chain $h_i = (y_{i-1}, q_i, x_i)$. Here, $y_0 = y_{t_1}$. If $x_i = y_i$, then merge two alternating chains h_i and h_{i+1} into one, resulting in $H = (h_1, \ldots, h_{t_2})$ with $t_2 \leq t_1$ and no two consecutive alternating chains being mergeable.

(ii) Apply the following FindPaths procedure.
Procedure FindPaths:

(1) Compute a t_2-minimum spanning forest (t_2-MSF) $F = (T_1, \ldots, T_{t_2})$ of $G[V^g \setminus \bigcup_{j=1}^{t_2} (V(h_j) \setminus \{g_e(h_j)\})]$ rooted at $\{g_e(h_j) | j \in [t_2]\}$.

(2) Double the edges of T_j's. Compute an Euler tour of each T_j and shortcut repeated vertices in the Euler tour, resulting in C_j.

(3) For each $j \in [t_2]$, do the following: if $C_j = (g_e(h_j))$, then define $p_j = (g_e(h_j), g_s(h_{j+1}))$; if $C_j = (g_e(h_j), g_{j_1}, \ldots, g_{j_{l_j}}, g_e(h_j))$ with $l_j \geq 1$, then define $p_j = (g_e(h_j), g_{j_1}, \ldots, g_{j_{l_j}}, g_s(h_{j+1}))$. Hereby, $h_{t_2+1} = h_1$.

(4) Connect $g_e(h_j)$ and $g_s(h_{j+1})$ in this t_2-partition by p_j for each $j \in [t_2]$, and obtain a TSP-solution $A = (h_1, p_1, h_2, p_2, \ldots, h_{t_2}, p_{t_2})$.

3. Return the solution $A^{\min}$ with the minimum cost among all enumeration cases.

The running time of this algorithm is dominated by the enumeration of the tuples of pairs of good vertices in Step 2(b).

▶ **Lemma 7.** *Algorithm 2 runs in $O(n^{O(k_2)})$ time.*

Proof. Step 1 computes k_2 bad vertices by search tree technique in $O(3^{k_2} \cdot n^3)$ time. The number of t_1-partitions of permutations enumerated in Step 2 is the same as in Lemma 2, $k_2! \, 2^{k_2}$. For each possible t_1-partition, the number of the t_1-tuples enumerated in Step 2(b) is $O(n^{2t_1}) = O(n^{2k_2})$. By the same argument as in Lemma 1, Algorithm 2 needs $O(n^2)$ time in Step 2(b)(ii). Hence, Algorithm 2 runs in $O(3^{k_2} \cdot n^3 + k_2! \, 2^{k_2} \cdot n^{2k_2} \cdot n^2) = O(n^{O(k_2)})$ time. ◀

Given an ordered collection of vertex-disjoint alternating chains $H = (h_1, \ldots, h_{t_2})$ containing all bad vertices, the FindPaths procedure deals only with edges between good vertices, to which we can apply the similar analysis as in Lemmas 3-5 and Theorem 6. Let opt^H be a minimum-cost TSP-solution, where the vertices in $\bigcup_{i=1}^{t_2} V(h_i)$ occur in the same alternating chains with the same order as specified in H. The following lemma is easy to see.

▶ **Lemma 8.** *Given an ordered collection of vertex-disjoint alternating chains $H = (h_1, \ldots, h_{t_2})$ containing all bad vertices, the FindPaths procedure returns a TSP-solution A with $c(A) \leq 3c(opt^H)$.*

We obtain the main result of this subsection as a consequence of Lemmas 7 and 8.

▶ **Theorem 9.** *Algorithm 2 is a 3-approximation algorithm for TSP running in $O(n^{O(k_2)})$ time, where k_2 is the minimum number of vertices whose removal results in a metric graph.*

Proof. Algorithm 2 returns $A^{\min}$ in $O(n^{O(k_2)})$ time by Lemma 7 and the output $A^{\min}$ is clearly a cycle over all vertices. Given an optimal TSP-solution opt, let Q^{opt} denote the ordered collection of all bad chains in opt, and H^{opt} denote the corresponding ordered collection of all alternating chains in opt. By the same analysis as in Section 3, there is a

case enumerated in Step 2, which gives the same order of bad chains as Q^{opt} in Step 2(a). Since good vertices occur only once in opt, there must be case α enumerated in Step 2(b), where we have the same alternating chains and they are in the same order as H^{opt} in Step 2(b)(i). By Lemma 8, in this case, the FindPaths procedure returns a TSP-solution A^{α} with $c(A^{\alpha}) \leq 3c(\text{opt}^{H^{\text{opt}}}) = 3c(\text{opt})$. Thus, $c(A^{\min}) \leq c(A^{\alpha}) \leq 3c(\text{opt})$. ◀

4.2 A parameterized $(6k_2 + 9)$-approximation algorithm

In this section, we apply a more involved strategy to fix the good vertices, that are direct neighbors of the bad chains in the optimal TSP-solution. We give an upper bound $f(k_2)$ on the number of possible good neighbors with f being a polynomial function. To achieve this, we slightly modify Algorithm 2. Given t_1 bad chains, we divide all good vertices into t_1 connected components with minimum edge costs by computing a t_1-minimum spanning forest. For each bad chain, we choose some good vertices from each connected component to form a set of good vertices that may be direct neighbors of the bad chain.

4.2.1 The algorithm and time complexity

The Step 1 and Step 2(a) of the parameterized approximation algorithm (Algorithm 3) are the same as Algorithm 2. Thus, we skip them in the description of Algorithm 3.

Algorithm 3 Ratio-$6k_2 + 9$ approximation algorithm for parameterization with k_2.

2. (b) Use Khachay and Neznakhina's algorithm [25] to get a t_1-minimum spanning forest of $G[V^g]$, $\mathcal{F} = (\mathcal{T}_1, \ldots, \mathcal{T}_{t_1})$. Let $V_j = V(\mathcal{T}_j)$ for $j \in [t_1]$.
For each bad vertex b, let $N(b, V_j)$ be the set of $\min\{4t_1, |V_j|\}$ good vertices in V_j, which are closest to b. That is, $\forall g' \in V_j \backslash N(b, V_j)$ and $\forall g \in N(b, V_j)$, $c(g', b) \geq c(g, b)$. Set $N(b) = \bigcup_{j=1}^{t_1} N(b, V_j)$.

(c) For all possible t_1-tuples of pairs of good vertices $((x_1, y_1), (x_2, y_2), \ldots, (x_{t_1}, y_{t_1}))$ satisfying $x_i \in N(b_e(q_i)) \backslash \bigcup_{j=1}^{i-1}\{x_j, y_j\}$ and $y_i \in N(b_s(q_{i+1})) \backslash \bigcup_{j=1}^{i-1}\{x_j, y_j\}$ for each $i \in [t_1]$: (Note that it might be $x_i = y_i$ for some i and $q_{t_1+1} = q_1$.)

(i) Transform q_i for each $i \in [t_1]$ to an alternating chain $h_i = (y_{i-1}, q_i, x_i)$. Here, $y_0 = y_{t_1}$. If $x_i = y_i$, then merge two alternating chains h_i and h_{i+1} into one, resulting in $H = (h_1, \ldots, h_{t_2})$ with $t_2 \leq t_1$ and no two consecutive alternating chains being mergeable.

(ii) Apply the FindPaths procedure.

3. Return the solution $A^{\min}$ with the minimum cost among all enumeration cases.

▶ **Lemma 10.** *Algorithm 3 runs in $O(k_2^{O(k_2)} \cdot n^3)$ time.*

Proof. As for Algorithm 2, Step 1 needs $O(3^{k_2} \cdot n^3)$ time. The number of t_1-partitions of permutations enumerated in Step 2 is $k_2! 2^{k_2}$. For each possible t_1-partition, a t_1-MSF of the graph $G[V^g]$ can be computed in $O(n^2 \log n)$ time [25]. The number of t_1-tuples enumerated in Step 2(c) is bounded by $O((t_1 \cdot 4t_1)^{2t_1}) = O((2k_2)^{4k_2})$. FindPaths needs $O(n^2)$ time. Hence, Algorithm 3 runs in $O(3^{k_2} \cdot n^3 + k_2! 2^{k_2} \cdot (n^2 \log n + (2k_2)^{4k_2} \cdot n^2)) = O(k_2^{O(k_2)} \cdot n^3)$ time. ◀

4.2.2 Analysis of approximation factor

Again, for an optimal TSP-solution opt with an order of bad chains, $Q^{\text{opt}} = (q_1^{\text{opt}}, \ldots, q_{t_1^{\text{opt}}}^{\text{opt}})$, we have a case α in Step 2 of Algorithm 3 with the same order of bad chains. In Step 2(b) of this case α, we obtain a t_1^{opt}-MSF $\mathcal{F}^{\alpha} = (\mathcal{T}_1^{\alpha}, \ldots, \mathcal{T}_{t_1^{\text{opt}}}^{\alpha})$ of the graph $G[V^g]$, and $V_j^{\alpha} = V(\mathcal{T}_j^{\alpha})$ for $j \in [t_1^{\text{opt}}]$. The following lemma is easy to see.

▶ **Lemma 11.** $c(opt) \geq c(\mathcal{F}^{\alpha})$.

Proof. We remove all bad vertices and all edges incident to them from opt, resulting in a spanning t_1^{opt}-forest of the graph $G[V^g]$. Thus, $c(\text{opt}) \geq c(\mathcal{F}^{\alpha})$. ◀

The main difficulty with the analysis of Algorithm 3, compared with Algorithm 2, lies in that the order of alternating chains H^{opt} might not be enumerated in Step 2(c). Therefore, we have to adopt a different approach. Hereby, we modify opt to construct another TSP-solution $\tilde{A}$, which satisfies on the one hand $c(\tilde{A}) = O(k_2 \cdot c(\text{opt}))$, and whose order of alternating chains on the other hand occurs in the case α' enumerated by Step 2(c). Thus, we know from Lemma 8 that Step 2(c)(ii) returns a solution $A^{\alpha'}$ in the case α' with $c(A^{\alpha'}) \leq 3c(\tilde{A})$, which completes the proof of approximation ratio.

Construction of opt* from opt. The construction of $\tilde{A}$ consists of three steps. The first step constructs a cycle opt* from opt. For each $j \in [t_1^{\text{opt}}]$, we partition $V_j^{\alpha} = V(\mathcal{T}_j^{\alpha})$ into three disjoint subsets X_j^1, X_j^2 and Y_j. The subset X_j^1 contains the good vertices in V_j^{α}, each of which is adjacent to one bad vertex and one good vertex in opt. The subset X_j^2 contains the good vertices in V_j^{α}, each of which is adjacent to two bad vertices in opt. The subset Y_j contains the good vertices in V_j^{α}, each of which is adjacent to two good vertices in opt. We aim to remove the vertices in Y_j's to get opt*. If $Y_j \neq \emptyset$ and $X_j^1 \neq \emptyset$ for $j \in [t_1^{\text{opt}}]$, then shortcut in opt all vertices in Y_j. If $Y_j \neq \emptyset$ and $X_j^1 = \emptyset$, then pick an arbitrary vertex in Y_j, say y_j, and shortcut in opt all vertices in $Y_j \backslash \{y_j\}$. After completing the above operation for all $j \in [t_1^{\text{opt}}]$, we get a new cycle, denoted by opt*.

Since we shortcut no bad vertex and no good vertex adjacent to bad vertices in opt, we have $c(\text{opt}^*) \leq c(\text{opt})$. The vertices in $X_j^1 \cup X_j^2$ and the possibly existing vertices in Y_j in opt* still keep their properties in opt: each good vertex in X_j^1 is adjacent to one bad vertex and one good vertex in opt*, each good vertex in X_j^2 is adjacent to two bad vertices in opt*, and if exists, y_j is adjacent to two good vertices in opt*.

Next, we partition X_j^1 into two disjoint subsets Z_j^0 and Z_j^1. The subset Z_j^0 contains vertices in X_j^1, which are adjacent in opt* to no vertex in $V_{j'}^{\alpha}$ with $j' \neq j$, i.e., each vertex in Z_j^0 is adjacent to one bad vertex and one good vertex in Z_j^0. The subset Z_j^1 contains vertices in X_j^1, which are adjacent in opt* to one vertex in $V_{j'}^{\alpha}$ with $j' \neq j$, i.e., each vertex in Z_j^1 is adjacent to one bad vertex and one good vertex in $Y_{j'} \cup Z_{j'}^1$ for some $j' \neq j$. The following two simple lemmas concerning opt* are useful.

▶ **Lemma 12.** $\sum_{j=1}^{t_1^{opt}} (2|X_j^2| + |Z_j^0| + |Z_j^1|) = 2t_1^{opt}$.

Proof. This lemma follows from the definitions of Z_j^0, Z_j^1 and X_j^2. ◀

▶ **Lemma 13.** *For each* $j \in [t_1^{opt}]$, $1 \leq |V_j^{\alpha} \cap V(opt^*)| \leq 2t_1^{opt}$.

Proof. For each $j \in [t_1^{\text{opt}}]$, exact one of the following three cases applies:

(a) If $Y_j = \emptyset$, then $V_j^{\alpha} = Z_j^0 \cup Z_j^1 \cup X_j^2 \subseteq V(\text{opt}^*)$ and $1 \leq |V_j^{\alpha} \cap V(\text{opt}^*)| = |V_j^{\alpha}| = |Z_j^0| + |Z_j^1| + |X_j^2| \leq 2t_1^{\text{opt}}$.

(b) If $Y_j \neq \emptyset$ and $X_j^1 = Z_j^0 \cup Z_j^1 \neq \emptyset$, then $V_j^{\alpha} \cap V(\text{opt}^*) = Z_j^0 \cup Z_j^1 \cup X_j^2$ and $1 \leq |V_j^{\alpha} \cap V(\text{opt}^*)| = |Z_j^0| + |Z_j^1| + |X_j^2| \leq 2t_1^{\text{opt}}$.

(c) If $Y_j \neq \emptyset$ and $X_j^1 = Z_j^0 \cup Z_j^1 = \emptyset$, then $V_j^{\alpha} \cap V(\text{opt}^*) = X_j^2 \cup \{y_j\}$ and $1 \leq |V_j^{\alpha} \cap V(\text{opt}^*)| = |X_j^2| + 1 \leq t_1^{\text{opt}} + 1 \leq 2t_1^{\text{opt}}$. ◀

Construction of A^* from opt*. Next we construct a cycle A^* from opt*. Since we shortcut no bad vertex and no good vertex adjacent to bad vertices in opt, $b_s(q_i^{\text{opt}})$ and $b_e(q_i^{\text{opt}})$ for each $i \in [t_1^{\text{opt}}]$ are adjacent in opt* to the same good vertices as in opt, denoted by $g_s(q_i^{\text{opt}})$ and $g_e(q_i^{\text{opt}})$. As defined in Step 2(b) of the case α, for each bad vertex b, $N(b, V_j^\alpha)$ is the set of $\min\{4t_1^{\text{opt}}, |V_j^\alpha|\}$ good vertices in V_j^α, which are closest to b, i.e., $\forall g' \in V_j^\alpha \backslash N(b, V_j^\alpha)$ and $\forall g \in N(b, V_j^\alpha)$, $c(g', b) \geq c(g, b)$. And $N(b) = \bigcup_{j=1}^{t_1^{\text{opt}}} N(b, V_j^\alpha)$. We apply the following procedure to opt*. Initially, set $W^* = \emptyset$. Here, $q_{t_1^{\text{opt}}+1}^{\text{opt}} = q_1^{\text{opt}}$.

From $i = 1$ to $i = t_1^{\text{opt}}$, do the following operations to opt*:

1. Consider V_j^α with $g_e(q_i^{\text{opt}}) \in V_j^\alpha$.
   If $g_e(q_i^{\text{opt}}) \notin N(b_e(q_i^{\text{opt}}), V_j^\alpha)$, then
   - **(a)** Pick an arbitrary vertex denoted by $g_e^*(q_i^{\text{opt}})$ from $N(b_e(q_i^{\text{opt}}), V_j^\alpha)\backslash(W^* \cup V(\text{opt}^*))$.
   - **(b)** Insert $g_e^*(q_i^{\text{opt}})$ between $b_e(q_i^{\text{opt}})$ and $g_e(q_i^{\text{opt}})$ in opt*.
   - **(c)** If $g_e(q_i^{\text{opt}}) \in Z_j^0$, then shortcut $g_e(q_i^{\text{opt}})$.
2. Consider $V_{j'}^\alpha$ with $g_s(q_{i+1}^{\text{opt}}) \in V_{j'}^\alpha$.
   If $g_s(q_{i+1}^{\text{opt}}) \notin N(b_s(q_{i+1}^{\text{opt}}), V_{j'}^\alpha)$, then
   - **(a)** Pick an arbitrary vertex denoted by $g_s^*(q_{i+1}^{\text{opt}})$ from $N(b_s(q_{i+1}^{\text{opt}}), V_{j'}^\alpha)\backslash(W^* \cup V(\text{opt}^*) \cup \{g_e^*(q_i^{\text{opt}})\})$.
   - **(b)** Insert $g_s^*(q_{i+1}^{\text{opt}})$ between $b_s(q_{i+1}^{\text{opt}})$ and $g_s(q_{i+1}^{\text{opt}})$ in opt*.
   - **(c)** If $g_s(q_{i+1}^{\text{opt}}) \in Z_{j'}^0$, then shortcut $g_s(q_{i+1}^{\text{opt}})$.
3. Add $g_e^*(q_i^{\text{opt}})$ and $g_s^*(q_{i+1}^{\text{opt}})$ to W^*.

The above procedure returns a cycle A^* for the following reason. If for some $i \in [t_1^{\text{opt}}]$, we have $g_e(q_i^{\text{opt}}) \in V_j^\alpha$ and $g_e(q_i^{\text{opt}}) \notin N(b_e(q_i^{\text{opt}}), V_j^\alpha)$, then $|N(b_e(q_i^{\text{opt}}), V_j^\alpha)| \geq 4t_1^{\text{opt}}$ by the definition of $N(b_e(q_i^{\text{opt}}), V_j^\alpha)$. By the fact that $|W^*| \leq 2t_1^{\text{opt}} - 2$ during the procedure and Lemma 13, $N(b_e(q_i^{\text{opt}}), V_j^\alpha)\backslash(W^* \cup V(\text{opt}^*)) \neq \emptyset$. By the same argument, $N(b_s(q_{i+1}^{\text{opt}}), V_{j'}^\alpha)\backslash(W^* \cup V(\text{opt}^*) \cup \{g_e^*(q_i^{\text{opt}})\}) \neq \emptyset$. Thus, the output is a cycle. Observe that $c(b_e(q_i^{\text{opt}}), g_e^*(q_i^{\text{opt}})) \leq c(b_e(q_i^{\text{opt}}), g_e(q_i^{\text{opt}}))$ and $c(b_s(q_{i+1}^{\text{opt}}), g_s^*(q_{i+1}^{\text{opt}})) \leq c(b_s(q_{i+1}^{\text{opt}}), g_s(q_{i+1}^{\text{opt}}))$.

Let $H^* = (h_1^*, \ldots, h_{t_2^*}^*)$ be the ordered collection of all alternating chains in A^*. Let opt^{H^*} be a minimum-cost TSP-solution, where the vertices in $\bigcup_{i=1}^{t_2^*} V(h_i^*)$ occur in the same alternating chains with the same order as specified in H^*. We get the following lemma as a consequence of Lemma 8.

▶ **Lemma 14.** *$c(A^{min}) \leq 3c(opt^{H^*})$, where A^{min} is the output of Algorithm 3.*

Proof. It is easy to observe that all good vertices in A^*, which are direct neighbors of the t_1^{opt} bad chains, are from $N(b_e(q_i^{\text{opt}}))$ and $N(b_s(q_i^{\text{opt}}))$ for $i \in [t_1^{\text{opt}}]$. Thus, there is a case α' in Step 2(c) where we get an order of alternating chains equal to H^* and by Lemma 8, we have in this case a solution $A^{\alpha'}$ with $c(A^{\alpha'}) \leq 3c(\text{opt}^{H^*})$. Finally, $c(A^{\text{min}}) \leq c(A^{\alpha'}) \leq 3c(\text{opt}^{H^*})$. ◀

Before moving to the third step, we give an upper bound on $c(A^*)$. Let $E_{jj'}^{A^*}$ be the set of edges in A^* connecting a good vertex in V_j^α and a good vertex in $V_{j'}^\alpha$ for $j, j' \in [t_1^{\text{opt}}]$. Then $c(A^*) = \sum_{i=1}^{t_2^*} c(h_i^*) + \sum_{j,j':1\leq j<j'\leq t_1^{\text{opt}}} c(E_{jj'}^{A^*}) + \sum_{j=1}^{t_1^{\text{opt}}} c(E_{jj}^{A^*})$. According to the construction of A^* from opt*, the following three lemmas are easy to see, which provide the foundation of upper-bounding $c(A^*)$.

▶ **Lemma 15.** *$\sum_{i=1}^{t_2^*} c(h_i^*) \leq \sum_{i=1}^{t_2^{opt}} c(h_i^{opt})$, where $H^{opt} = (h_1^{opt}, \ldots, h_{t_2^{opt}}^{opt})$ is the ordered collection of all alternating chains in opt.*

Proof. For each $i \in [t_1^{\mathrm{opt}}]$, $(b_e(q_i^{\mathrm{opt}}), g_e(q_i^{\mathrm{opt}})) \in E(\mathrm{opt})$, and exact one of the following two cases applies:

(a) $b_e(q_i^{\mathrm{opt}})$ is adjacent to $g_e(q_i^{\mathrm{opt}})$ in A^*.

(b) $b_e(q_i^{\mathrm{opt}})$ is adjacent to $g_e^*(q_i^{\mathrm{opt}})$ in A^*, and $c(b_e(q_i^{\mathrm{opt}}), g_e^*(q_i^{\mathrm{opt}})) \le c((b_e(q_i^{\mathrm{opt}}), g_e(q_i^{\mathrm{opt}}))$.

This property also holds for the good vertex in A^* adjacent to $b_s(q_i^{\mathrm{opt}})$, for each $i \in [t_1^{\mathrm{opt}}]$. At the same time, A^* contains the same bad chains as opt. Thus, $\sum_{i=1}^{t_2^*} c(h_i^*) \le \sum_{i=1}^{t_2^{\mathrm{opt}}} c(h_i^{\mathrm{opt}})$. ◀

▶ **Lemma 16.** *For all $j' \neq j$ and $j, j' \in [t_1^{opt}]$, $E_{jj'}^{A^*} = E_{jj'}^{opt^*}$, where $E_{jj'}^{opt^*}$ is the set of edges in opt* that connect a vertex in V_j^α and a vertex in $V_{j'}^\alpha$.*

Proof. For each $i \in [t_1^{\mathrm{opt}}]$, there exists some $j \in [t_1^{\mathrm{opt}}]$ such that $g_e(q_i^{\mathrm{opt}}) \in V_j^\alpha$ and $g_e^*(q_i^{\mathrm{opt}}) \in V_j^\alpha$. Inserting $g_e^*(q_i^{\mathrm{opt}})$ between $b_e(q_i^{\mathrm{opt}})$ and $g_e(q_i^{\mathrm{opt}})$ does not add or remove an edge between a good vertex in $V_{j''}^\alpha$ and a good vertex $V_{j'''}^\alpha$ with $j'' \neq j'''$. This property also holds for inserting $g_s^*(q_i^{\mathrm{opt}})$ between $b_s(q_i^{\mathrm{opt}})$ and $g_s(q_i^{\mathrm{opt}})$, for each $i \in [t_1^{\mathrm{opt}}]$.

For each edge $(g, g') \in E_{jj'}^{\mathrm{opt}^*}$ $(j' \neq j)$ where $g \in V_j^\alpha$ and $g' \in V_{j'}^\alpha$, $g \notin Z_j^0$ and $g' \notin Z_{j'}^0$. Then, according to the construction of A^* from opt*, we do not shortcut g or g'. ◀

▶ **Lemma 17.** *For each $j \in [t_1^{opt}]$, $|E_{jj}^{A^*}| \le 2|X_j^2| + |Z_j^0| + |Z_j^1| \le 2t_1^{opt}$.*

Proof. According to the construction of A^* from opt*, it is easy to see that $|E_{jj}^{A^*}| \le 2|X_j^2| + |Z_j^0| + |Z_j^1|$. By Lemma 12, $2|X_j^2| + |Z_j^0| + |Z_j^1| \le 2t_1^{\mathrm{opt}}$, and thus, $|E_{jj}^{A^*}| \le 2t_1^{\mathrm{opt}}$. ◀

▶ **Lemma 18.** *$c(A^*) \le (2k_2 + 1)c(opt)$.*

Proof. Recall that $\mathcal{F}^\alpha = (\mathcal{T}_1^\alpha, \ldots, \mathcal{T}_{t_1^{\mathrm{opt}}}^\alpha)$ is the minimum-cost spanning t_1^{opt}-forest of the graph $G[V^g]$, and $V_j^\alpha = V(\mathcal{T}_j^\alpha)$ for each $j \in [t_1^{\mathrm{opt}}]$. Thus, for each $(g, g') \in E_{jj}^{A^*}$, it trivially holds $c(g, g') \le c(\mathcal{T}_j^\alpha)$. Since opt* has the order H^{opt}, $c(\mathrm{opt}^*) \ge \sum_{i=1}^{t_2^{\mathrm{opt}}} c(h_i^{\mathrm{opt}}) + \sum_{j,j':1\le j<j'\le t_1^{\mathrm{opt}}} c(E_{jj'}^{\mathrm{opt}^*})$. By Lemmas 11, 15, 16, and 17, we have

$$
\begin{aligned}
c(A^*) &= \sum_{i=1}^{t_2^*} c(h_i^*) + \sum_{j,j':1\le j<j'\le t_1^{\mathrm{opt}}} c(E_{jj'}^{A^*}) + \sum_{j=1}^{t_1^{\mathrm{opt}}} c(E_{jj}^{A^*}) \\
&\le \sum_{i=1}^{t_2^{\mathrm{opt}}} c(h_i^{\mathrm{opt}}) + \sum_{j,j':1\le j<j'\le t_1^{\mathrm{opt}}} c(E_{jj'}^{\mathrm{opt}^*}) + \sum_{j=1}^{t_1^{\mathrm{opt}}} (|E_{jj}^{A^*}| \cdot c(\mathcal{T}_j^\alpha)) \\
&\le c(\mathrm{opt}^*) + 2t_1^{\mathrm{opt}} \sum_{j=1}^{t_1^{\mathrm{opt}}} c(\mathcal{T}_j^\alpha) \\
&= c(\mathrm{opt}^*) + 2t_1^{\mathrm{opt}} \cdot c(\mathcal{F}^\alpha) \\
&\le c(\mathrm{opt}) + 2k_2 \cdot c(\mathrm{opt}) \\
&= (2k_2 + 1)c(\mathrm{opt}).
\end{aligned}
$$
◀

Construction of $\tilde{A}$ from A^*. Finally, we insert all remaining vertices in $V_j^\alpha \backslash V(A^*)$ into A^* for all $j \in [t_1^{\mathrm{opt}}]$ to get a TSP-solution $\tilde{A}$. By the analysis of cases, we obtain the following lemma, which is useful for the choice of locations of insertions.

▶ **Lemma 19.** *For each $j \in [t_1^{opt}]$, at least one of the following three cases applies:*

(a) *$V_j^\alpha \subseteq V(A^*)$.*

(b) *$E_{jj}^{A^*} \neq \emptyset$.*

(c) *There exists $j' \neq j$ such that $E_{jj'}^{A^*} \neq \emptyset$.*

Proof. For each $j \in [t_1^{\text{opt}}]$, at least one of the following four cases applies:

(a) If $Y_j = \emptyset$, then $V_j^\alpha = Z_j^0 \cup Z_j^1 \cup X_j^2 \subseteq V(\text{opt}^*)$ and $|V_j^\alpha| = |Z_j^0| + |Z_j^1| + |X_j^2| \leq 2t_1^{\text{opt}} < 4t_1^{\text{opt}}$. For each $i \in [t_1^{\text{opt}}]$, $N(b_e(q_i^{\text{opt}}), V_j^\alpha) = V_j^\alpha$ and $N(b_s(q_i^{\text{opt}}), V_j^\alpha) = V_j^\alpha$. According to the construction of A^* from opt^*, we insert or shortcut no vertex in V_j^α. Thus, $V_j^\alpha \subseteq V(A^*)$.

(b) If $Y_j \neq \emptyset$ and $Z_j^0 \neq \emptyset$, then $E_{jj}^{\text{opt}^*} \neq \emptyset$. According to the construction of A^* from opt^*, $E_{jj}^{A^*} \neq \emptyset$.

(c) If $Y_j \neq \emptyset$ and $Z_j^1 \neq \emptyset$, then there exists $j' \neq j$ such that $E_{jj'}^{\text{opt}^*} \neq \emptyset$. By Lemma 16, $E_{jj'}^{A^*} = E_{jj'}^{\text{opt}^*} \neq \emptyset$.

(d) If $Y_j \neq \emptyset$ and $X_j^1 = Z_j^0 \cup Z_j^1 = \emptyset$, then $Y_j \cap V(\text{opt}^*) = \{y_j\}$. There exists $j' \neq j$ such that $E_{jj'}^{\text{opt}^*} \neq \emptyset$. By Lemma 16, $E_{jj'}^{A^*} = E_{jj'}^{\text{opt}^*} \neq \emptyset$. ◀

We insert the vertices in $V_j^\alpha \backslash V(A^*)$ as follows. We double the edges in $\mathcal{T}_j^\alpha$ and shortcut repeated vertices on the $\mathcal{T}_j^\alpha$'s Euler tour, resulting in a cycle C_j^α, where $V(C_j^\alpha) = V_j^\alpha$ and $c(C_j^\alpha) \leq 2c(\mathcal{T}_j^\alpha)$ for each $j \in [t_1^{\text{opt}}]$. We insert the remaining vertices in $V_j^\alpha \backslash V(A^*)$ into A^* according to their orders in C_j^α.

From $j = 1$ to $j = t_1^{\text{opt}}$, do the following operation to A^*:

1. If $V_j^\alpha \subseteq V(A^*)$, then go to the next iteration.
2. If $E_{jj}^{A^*} \neq \emptyset$, then let (g_{j_s}, g_{j_e}) be an arbitrary edge in $E_{jj}^{A^*}$. Traverse the cycle C_j^α, starting from g_{j_s} and shortcutting all vertices not in $(V_j^\alpha \backslash V(A^*)) \cup \{g_{j_s}\}$. Let $C_j' = (g_{j_s}, g_{j_1}, \ldots, g_{j_{l_j}}, g_{j_s})$ be the resulting cycle. Insert the path $(g_{j_1}, \ldots, g_{j_{l_j}})$ between g_{j_s} and g_{j_e} in A^* and remove (g_{j_s}, g_{j_e}). Go to the next iteration.
3. If there is a $j' \neq j$ with $E_{jj'}^{A^*} \neq \emptyset$, then let $(g_{j_s}, g_{j'_e})$ be an arbitrary edge in $E_{jj'}^{A^*}$, where $g_{j_s} \in V_j^\alpha$ and $g_{j'_e} \in V_{j'}^\alpha$. Traverse the cycle C_j^α, starting from g_{j_s} and shortcutting all vertices not in $(V_j^\alpha \backslash V(A^*)) \bigcup \{g_{j_s}\}$. Let $C_j' = (g_{j_s}, g_{j_1}, \ldots, g_{j_{l_j}}, g_{j_s})$ be the resulting cycle.
   (a) If $V_{j'}^\alpha \subseteq V(A^*)$, then insert the path $(g_{j_1}, \ldots, g_{j_{l_j}})$ between g_{j_s} and $g_{j'_e}$ in A^* and remove $(g_{j_s}, g_{j'_e})$.
   (b) If $V_{j'}^\alpha \backslash V(A^*) \neq \emptyset$, then traverse the cycle $C_{j'}^\alpha$, starting from $g_{j'_e}$ and shortcutting all vertices not in $(V_{j'}^\alpha \backslash V(A^*)) \bigcup \{g_{j'_e}\}$. Let $C_{j'}' = (g_{j'_e}, g_{j'_1}, \ldots, g_{j'_{l_{j'}}}, g_{j'_e})$ be the resulting cycle. Then insert the path $(g_{j_1}, \ldots, g_{j_{l_j}}, g_{j'_1}, \ldots, g_{j'_{l_{j'}}})$ between g_{j_s} and $g_{j'_e}$ in A^* and remove $(g_{j_s}, g_{j'_e})$.

After the above iteration, we obtain a new cycle $\tilde{A}$. By Lemma 19, $\tilde{A}$ is a TSP-solution. For each $j \in [t_1^{\text{opt}}]$, since $c(C_j^\alpha) \leq 2c(\mathcal{T}_j^\alpha)$, there is only a small cost increase caused by the insertion of all remaining vertices in $V_j^\alpha \backslash V(A^*)$.

▶ **Lemma 20.** $c(\tilde{A}) \leq (2k_2 + 3)c(opt)$.

Proof. We use Δc_j to denote the cost increase caused by the insertion of the vertices in $V_j^\alpha \backslash V(A^*)$ and give an upper bound on Δc_j. In the first case, Δc_j is clearly 0. In the second case, we insert the path $(g_{j_1}, \ldots, g_{j_{l_j}})$ between g_{j_s} and g_{j_e} in A^*. Thus, $\Delta c_j = c(C_j') - c(g_{j_{l_j}}, g_{j_s}) - c(g_{j_s}, g_{j_e}) + c(g_{j_{l_j}}, g_{j_e}) \leq c(C_j') \leq c(C_j^\alpha) \leq 2c(\mathcal{T}_j^\alpha)$. Case 3(a) is identical to the second case. In Case 3(b) we add two paths in A^* involving V_j^α and $V_{j'}^\alpha$. By a similar analysis, we have both $\Delta c_j \leq 2c(\mathcal{T}_j^\alpha)$ and $\Delta c_{j'} \leq 2c(\mathcal{T}_{j'}^\alpha)$. By Lemmas 11 and 18, we conclude $c(\tilde{A}) = c(A^*) + \sum_{j=1}^{t_1^{\text{opt}}} \Delta c_j \leq c(A^*) + 2\sum_{j=1}^{t_1^{\text{opt}}} c(\mathcal{T}_j^\alpha) = c(A^*) + 2c(\mathcal{F}^\alpha) \leq (2k_2 + 1)c(\text{opt}) + 2c(\text{opt}) = (2k_2 + 3)c(\text{opt})$. ◀

Now, we have all tools to prove the main result of this subsection.

▶ **Theorem 21.** *Algorithm 3 is a parameterized* $(6k_2+9)$*-approximation algorithm for TSP running in* $O(k_2^{O(k_2)} \cdot n^3)$ *time, where* k_2 *is the minimum number of vertices whose removal results in a metric graph.*

Proof. Recall that H^* is the order of all alternating chains in A^*. From A^* to $\tilde{A}$, we insert the remaining good vertices between two good vertices without changing the good vertices which are direct neighbors of bad chains. Thus, the alternating chains in $\tilde{A}$ occur in the same order in A^*, and then $c(\tilde{A}) \geq c(\text{opt}^{H^*})$. By Lemmas 14 and 20, $c(A^{\min}) \leq 3c(\text{opt}^{H^*}) \leq 3c(\tilde{A}) \leq (6k_2+9)c(\text{opt})$. Combining with Lemma 10, we conclude that Algorithm 3 returns a $(6k_2+9)$-approximation solution in $O(k_2^{O(k_2)} \cdot n^3)$ time. ◀

5 Conclusion

Based on the concept of "distance from approximability", we present two parameterized approximation algorithms for TSP on general graphs, parameterized by the number k_1 of violating triangles or the minimum size k_2 of violating vertex sets, which achieve approximation factors of 3 and $6k_2+9$, respectively. These seem to be the first parameterized approximation algorithms for TSP on general graphs. It remains open whether the factor $6k_2+9$ can be improved to a constant. Moreover, in comparison with the $O(2^n)$-time exact algorithm [4, 21], it is crucial to improve the running time of the approximation algorithms to increase their practical relevance. Finally, it is an interesting research direction to apply the concept of "distance from approximability" to other optimization problems.

References

1 Thomas Andreae. On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. *Networks*, 38(2):59–67, 2001.

2 Thomas Andreae and Hans-Jürgen Bandelt. Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM Journal on Discrete Mathematics*, 8(1):1–16, 1995.

3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

4 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

5 Michael A. Bender and Chandra Chekuri. Performance guarantees for the TSP with a parameterized triangle inequality. *Information Processing Letters*, 73(1-2):17–21, 2000.

6 Hans-Joachim Böckenhauer, Juraj Hromkovič, Ralf Klasing, Sebastian Seibert, and Walter Unger. Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. *Theoretical Computer Science*, 285(1):3–24, 2002.

7 Hans-Joachim Bockenhauer, Juraj Hromkovic, Joachim Kneis, and Joachim Kupke. The parameterized approximability of TSP with deadlines. *Theory of Computing Systems*, 41(3):431–444, 2007.

8 Edouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and FPT-time inapproximability. *Algorithmica*, 71(3):541–565, 2015.

9 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *Proceedings of the IEEE 58th Annual Symposium on Foundations of Computer Science*, pages 743–754. IEEE, 2017.

10 Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science*, pages 505–514. IEEE, 2016.

11 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Carnegie-Mellon University, 1976.

12 Vladimir G. Deĭneko, Michael Hoffmann, Yoshio Okamoto, and Gerhard J. Woeginger. The traveling salesman problem with few inner points. *Operations Research Letters*, 34(1):106–110, 2006.

13 Erik D. Demaine and MohammadTaghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 840–849. SIAM, 2004.

14 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

15 Andreas Emil Feldmann, C.S. Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

16 Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. *Bulletin (New Series) of the AMS*, 3(2):898–904, 1980.

17 Lee-Ad Gottlieb. A light metric spanner. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 759–772. IEEE, 2015.

18 Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23:613–632, 2003.

19 Martin Grohe, Ken-ichi Kawarabayashi, and Bruce Reed. A simple algorithm for the graph minor decomposition - logic meets structural graph theory-. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 414–431. SIAM, 2013.

20 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation*, pages 162–173. Springer, 2004.

21 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the SIAM*, 10(1):196–210, 1962.

22 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 32–45. ACM, 2021.

23 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

24 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k,r)-center. *Discrete Applied Mathematics*, 264:90–117, 2019.

25 Michael Khachay and Katherine Neznakhina. Approximability of the minimum-weight k-size cycle cover problem. *Journal of Global Optimization*, 66(1):65–82, 2016.

26 Christian Knauer and Andreas Spillner. A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 49–57. Springer, 2006.

27 Bingkai Lin. Constant approximating k-clique is W[1]-hard. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1749–1756. ACM, 2021.

28 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

29 Usha Mohan, Sivaramakrishnan Ramani, and Sounaka Mishra. Constant factor approximation algorithm for TSP satisfying a biased triangle inequality. *Theoretical Computer Science*, 657:111–126, 2017.

30 Tobias Mömke. An improved approximation algorithm for the traveling salesman problem with relaxed triangle inequality. *Information Processing Letters*, 115(11):866–871, 2015.

31 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

32 Christos H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

33 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1283–1296. ACM, 2018.

34 Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

35 Anatolii Ivanovich Serdyukov. On some extremal walks in graphs. *Upravliaemie systemy*, 17:76–79, 1978.

Partial and Simultaneous Transitive Orientations via Modular Decompositions

Miriam Münch
Faculty of Computer Science and Mathematics, Universität Passau, Germany

Ignaz Rutter
Faculty of Computer Science and Mathematics, Universität Passau, Germany

Peter Stumpf
Faculty of Computer Science and Mathematics, Universität Passau, Germany

Abstract

A natural generalization of the recognition problem for a geometric graph class is the problem of extending a representation of a subgraph to a representation of the whole graph. A related problem is to find representations for multiple input graphs that coincide on subgraphs shared by the input graphs. A common restriction is the sunflower case where the shared graph is the same for each pair of input graphs. These problems translate to the setting of comparability graphs where the representations correspond to transitive orientations of their edges. We use modular decompositions to improve the runtime for the orientation extension problem and the sunflower orientation problem to linear time. We apply these results to improve the runtime for the partial representation problem and the sunflower case of the simultaneous representation problem for permutation graphs to linear time. We also give the first efficient algorithms for these problems on circular permutation graphs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases representation extension, simultaneous representation, comparability graph, permutation graph, circular permutation graph, modular decomposition

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.51

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2209.13175

Funding Funded by grant RU 1903/3-1 of the German Research Foundation (DFG).

1 Introduction

Representations and drawings of graphs have been considered since graphs have been studied [28]. A *geometric intersection representation* of a graph $G = (V, E)$ with regards to a class of geometric objects $\mathcal{C}$, is a map $R\colon V \to \mathcal{C}$ that assigns objects of $\mathcal{C}$ to the vertices of G such that G contains an edge uv if and only if the intersection $R(u) \cap R(v)$ is non-empty. In this way, the class $\mathcal{C}$ gives rise to a class of graphs, namely the graphs that admit such a representation. As an example, consider *permutation diagrams* where $\mathcal{C}$ consists

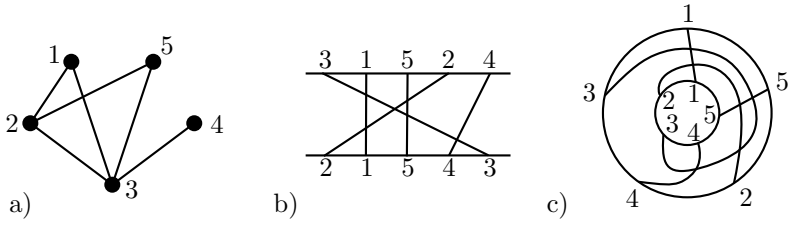


Figure 1 (a) a graph G with (b) a permutation diagram and (c) a circular permutation diagram.

of segments connecting two parallel lines ℓ_1, ℓ_2, see Figure 1b, which defines the class Perm of *permutation graphs*. Similarly, the class CPerm of *circular permutation graphs* is obtained by replacing ℓ_1, ℓ_2 with concentric circles and the geometric objects with curves from ℓ_1 to ℓ_2 that pairwise intersect at most once; see Figure 1c.

A key problem in this context is the *recognition problem*, which asks whether a given graph admits such a representation for a fixed class $\mathcal{C}$. Klavík et al. introduced the *partial representation extension problem* (RepExt($\mathcal{C}$)) for intersection graphs where a representation R' is given for a subset of vertices $W \subseteq V$ and the question is whether R' can be extended to a representation R of G, in the sense that $R|_W = R'$ [28]. They showed that RepExt can be solved in linear time for interval graphs. The problem has further been studied for proper/unit interval graphs [26], function and permutation graphs (Perm) [25], circle graphs [11], chordal graphs [27], and trapezoid graphs [29]. Related extension problems have also been considered, e.g., for planar topological [1, 24] and straight-line [32] drawings, for 1-planar drawings [14], for contact representations [10], and for rectangular duals [12].

A related problem is the *simultaneous representation problem* (SimRep($\mathcal{C}$)) where input graphs $G_1 \ldots, G_r$ that may share subgraphs are given and the question is whether they have representations $R_1, \ldots, R_r$ such that for $i, j \in \{1, \ldots, r\}$ the shared graph $H = G_i \cap G_j$ has the same representation in R_i and R_j, i.e., $R_i|_{V(H)} = R_j|_{V(H)}$. If more than two input graphs are allowed, usually the *sunflower case* (SimRep*($\mathcal{C}$)) is considered, where the shared graph $H = G_i \cap G_j$ is the same for any $i \neq j \in \{1, \ldots, r\}$. I.e., here the question is whether H has a representation that can be simultaneously extended to $G_1, \ldots, G_r$. Simultaneous representations were first studied in the context of planar drawings [4, 9], where the goal is to embed each input graph without edge crossings while shared subgraphs have the same induced embedding. Unsurprisingly, many variants are NP-complete [17, 35, 2, 15].

Motivated by applications in visualization of temporal relationships, and for overlapping social networks or schedules, DNA fragments of similar organisms and adjacent layers on a computer chip, Jampani and Lubiw introduced the problem SimRep for intersection graphs [23]. They provided polynomial-time algorithms for two chordal graphs and for SimRep*(Perm). They also showed that in general SimRep is NP-complete for three or more chordal graphs. The problem was also studied for interval graphs [22, 5, 6], proper/unit interval graphs [34], circular-arc graphs [6] and circle graphs [11].

Many of the considered graph classes are related to the class Comp of *comparability graphs* [19]. An *orientation* O of a graph $G = (V, E)$ assigns to each edge of G a direction. The orientation O is *transitive* if $uv, vw \in O$ implies $uw \in O$. A comparability graph is a graph for which there is a *transitive orientation*. A *partial orientation* is an orientation of a (not necessariliy induced) subgraph of G. Similar to RepExt and SimRep*, the problems OrientExt and SimOrient for comparability graphs ask for a transitive orientation of a graph that extends a given partial orientation and for transitive orientations that coincide on the shared graph, respectively. The key ingredient for the $O(n^3)$ algorithm solving RepExt(Perm) by Klavík et al. [25] is a polynomial-time solution for OrientExt based on the transitive orientation algorithm by Gilmore and Hoffman [18]. Likewise, the $O(n^3)$ algorithm solving SimRep*(Perm) by Jampani and Lubiw [23] is based on a polynomial-time algorithm for SimOrient based on the transitive orientation algorithm by Golumbic [19].

Contribution and Outline. In Section 2, we introduce modular decompositions which can be used to describe certain subsets of the set of all transitive orientations of a graph, e.g., those that extend a given partial representation. Based on this, we give a simple linear-time algorithm for OrientExt in Section 3. Afterwards, in Section 4, we develop an algorithm

Table 1 Known runtimes on the left and new runtimes on the right. For SimRep* we set $n = \sum_{i=1}^{r} |V(G_i)|$ and $m = \sum_{i=1}^{r} |E(G_i)|$. We use RepExt(Comp) and SimRep*(Comp) to refer to OrientExt and SimOrient, respectively, in a slight abuse of notation.

	RepExt	SimRep*
Comp	$O((n+m)\Delta)$ [25]	$O(nm)$ [23]
Perm	$O(n^3)$ [25]	$O(n^3)$ [23]
CPerm	open	open

	RepExt	SimRep*
Comp	$O(n+m)$	$O(n+m)$
Perm	$O(n+m)$	$O(n+m)$
CPerm	$O(n+m)$	$O(n^2)$

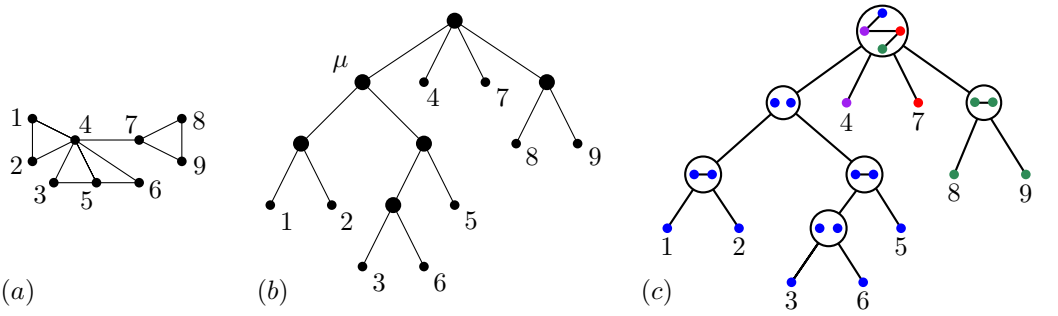


Figure 2 (a) A graph G. (b) The canonical modular decomposition of G with $L(\mu) = \{1, 2, 3, 5, 6\}$. (c) The quotient graphs corresponding to the nodes in the canonical modular decomposition.

for intersecting subsets of transitive orientations represented by modular decompositions and use this to give a linear-time algorithm for SimOrient. In Section 5 we give linear-time algorithms for RepExt(Perm) and SimRep*(Perm), improving over the $O(n^3)$-algorithms of Klavík et al. and Jampani and Lubiw, respectively. We also give the first efficient algorithms for RepExt(CPerm) and SimRep*(CPerm). Table 1 gives an overview of the state of the art and our results. Proofs marked with a $\star$ are omitted due to space restrictions. For the full proofs we refer to the long version [31] .

2 Modular Decompositions

Let $G = (V, E)$ be an undirected graph. We write $G[U]$ for the subgraph induced by a vertex set $U \subseteq V$. For a rooted tree T and a node μ of T, we write $T[\mu]$ for the subtree of T with root μ and $L(\mu)$ for the leaf-set of $T[\mu]$.

A *module* of G is a non-empty set of vertices $M \subseteq V$ such that every vertex $u \in V \setminus M$ is either adjacent to all vertices in M or to none of them. The singleton subsets and V itself are called the *trivial modules*. A module $M \subsetneq V$ is *maximal*, if there exists no module M' such that $M \subsetneq M' \subsetneq V$. If G has at least three vertices and no non-trivial modules, then it is called *prime*. We call a rooted tree T with root ρ and $L(\rho) = V$ a *(general) modular decomposition* for G if for every node μ of T the set $L(\mu)$ is a module; see Figure 2. Observe that for any two nodes $\mu_1, \mu_2 \in T$ such that neither of them is an ancestor of the other, G contains either all edges with one endpoint in $L(\mu_1)$ and one endpoint in $L(\mu_2)$ or none of them. For two vertices $u, v \in V$ we denote the lowest common ancestor of their corresponding leaves in T by $\mathrm{lca}_T(u, v)$. For a set of leaves L, we denote the lowest common ancestor by $\mathrm{lca}_T(L)$.

With each inner node μ of T we associate a *quotient graph* $G[\mu]$ that is obtained from $G[L(\mu)]$ by contracting $L(\nu)$ into a single vertex for each child ν of μ; see Figure 2. In the rest of this paper we identify the vertices of $G[\mu]$ with the corresponding children of μ.

Every edge $uv \in E$ is *represented* by exactly one edge $\text{rep}_T(uv)$ in one of the quotient graphs of T, namely in the quotient graph $G[\mu]$ of the lowest common ancestor μ of u and v. More precisely, if ν and λ are the children of μ with $u \in L(\nu)$ and $v \in L(\lambda)$, then $\text{rep}_T(uv) = \nu\lambda$. For an oriented edge uv, $\text{rep}_T(uv)$ is also oriented towards its endpoint ν with $v \in L(\nu)$. If T is clear from the context, the subscript can be omitted. Let μ be a node in T. For a vertex $u \in L(\mu)$ we denote the child λ of μ with $u \in L(\lambda)$ by $\text{rep}_\mu(u)$.

A node μ in a modular decomposition T is called *empty*, *complete* or *prime* if the quotient graph $G[\mu]$ is *empty*, *complete* or *prime*, respectively. By $K(T)$, $P(T)$ we denote the set of all complete and prime nodes in T, respectively. For every graph G there exists a uniquely defined modular decomposition, that we call *the canonical modular decomposition* of G, introduced by Gallai [16], such that each quotient graph is either prime, complete or empty and, additionally, no two adjacent nodes are both complete or are both empty; see Figure 2. Note that in the literature, these are referred to as modular decompositions, whereas we use that term for general modular decompositions. For a prime node μ in the canonical modular decomposition of G, for every child ν of μ, $L(\nu)$ is a maximal module in $G[L(\mu)]$ and for every maximal module M in $G[L(\mu)]$ there exists a child ν of μ with $L(\nu) = M$. McConnell and Spinrad showed that the canonical modular decomposition can be computed in $O(|V| + |E|)$ time [30]. Let μ be a node in a modular decomposition for G. A *μ-set* U is a subset of $L(\mu)$ that contains for each child λ of μ at most one leaf in $L(\lambda)$. If U contains for every child λ of μ a vertex in $L(\lambda)$, we call it *maximal*.

▶ **Lemma 1** (⋆). *Let T be a modular decomposition of a graph G. After a linear-time preprocessing we can assume that each node of T is annotated with its quotient graph. Moreover, the following queries can be answered in $O(1)$ time:*

1) *Given a non-root node ν of T, find the vertex of the quotient graph of ν's parent that corresponds to ν.*
2) *Given a vertex v in a quotient graph $G[\mu]$, find the child of μ that corresponds to v.*
3) *Given an edge e of G, determine* $\text{rep}(e)$*, the quotient graph that contains* $\text{rep}(e)$*, and which endpoint of* $\text{rep}(e)$ *corresponds to which endpoint of e.*

Additionally, given a node μ in T one can find a maximal μ-set U in $O(|U|)$ time.

Let μ be a node in a modular decomposition T of G and let $\mu_1\mu_2$ be an edge in $G[\mu]$. Let $\vec{T}[G]$ denote an assignment of directions to all edges in $G[\mu]$ for every node μ in T. Such a $\vec{T}[G]$ is *transitive* if it is transitive on every $G[\mu]$. We obtain an orientation of G from an orientation $\vec{T}[G]$ of the quotient graphs of T as follows. Every undirected edge uv in E with $\text{rep}(uv) = \mu_1\mu_2 \in \vec{T}[G]$, is directed from u to v. We say that T *represents* an orientation O of G, if there exists an orientation $\vec{T}[G]$ of the quotient graphs of T that gives us O. We denote the set of all transitive orientations of G represented by T by $\text{to}(T)$. We get an orientation of the quotient graphs of T from an orientation of G, if for each oriented edge $\mu_1\mu_2$, all edges represented by $\mu_1\mu_2$ are oriented from $L(\mu_1)$ to $L(\mu_2)$. Let T now be the canonical modular decomposition of G. Then T represents exactly the transitive orientations of G [16]. It follows that G is a comparability graph if and only if T can be oriented transitively.

If G is a comparability graph, every prime quotient graph $G[\mu] = (V_\mu, E_\mu)$ has exactly two transitive orientations, one the reverse of the other [19], and with the algorithm by McConnell and Spinrad [30] we can compute one of them in $O(|V_\mu| + |E_\mu|)$ time. Hence the time to compute the canonical modular decomposition in which every prime node is labeled with a corresponding transitive orientation is $O(|V| + |E|)$.

3 Transitive Orientation Extension

The *partial orientation extension problem* for comparability graphs OrientExt is to decide for a comparability graph G with a partial orientation W, i.e. an orientation of some of its edges, whether there exists a transitive orientation O of G with $W \subseteq O$. The notion of partial orientations and extensions extends to modular decompositions. We get a partial orientation of the quotient graphs of T from W such that exactly the edges that represent at least one edge in W are oriented and all edges in W that are represented by the same oriented edge $\mu_1\mu_2$, are directed from $L(\mu_1)$ to $L(\mu_2)$.

▶ **Lemma 2** ($\star$). *Let T be the canonical modular decomposition of a comparability graph G and let W be a partial orientation of G that gives us a partial orientation P of the quotient graphs of T. Then W extends to a transitive orientation of G if and only if P extends to a transitive orientation of the quotient graphs of T.*

▶ **Theorem 3** ($\star$). *OrientExt can be solved in linear time.*

Sketch of proof. We confirm that the partial orientation actually gives us a partial orientation P of the quotient graphs of the canonical modular decomposition T. Otherwise we can reject. By Lemma 2 we now just need to check for each node μ of T whether P can be extended to $G[\mu]$. To this end, we use that μ is empty, complete or prime. Since transitive orientations of cliques are total orders and prime graphs have at most two transitive orientations, the existence of an extension can easily be decided in each case. ◀

4 Sunflower Orientations

The idea to solve SimOrient is to obtain for each input graph G_i a restricted modular decomposition of the shared graph H that represents exactly those transitive orientations of H that can be extended to G_i. The restricted modular decompositions can be expressed by constraints for the canonical modular decomposition of H. These constraints are then combined to represent all transitive orientations of H that can be extended to each input graph G_i. With this the solution is straightforward.

Let H be a comparability graph. Then we define a *restricted modular decomposition* (T, D) of H to be a tuple where T is a modular decomposition of H where every node is labeled as complete, empty or prime, such that for every node μ labeled as complete or empty, $H[\mu]$ is complete or empty, respectively, and D is a function that assigns to each prime labeled node μ a transitive orientation D_μ, called *default orientation*. In the following, when referring to the type of a node μ in a restricted modular decomposition, we mean the type that μ is labeled with. A transitive orientation of (T, D), is a transitive orientation of the quotient graphs of T where every prime node μ has orientation D_μ or its reversal D_μ^{-1}. Let $\mathrm{to}(T, D)$ denote the set of transitive orientations of H we get from transitive orientations of (T, D). We say (T, D) *represents* these transitive orientations. Note that for the canonical modular decomposition B of H we have $\mathrm{to}(T, D) \subseteq \mathrm{to}(B)$.

Let G be a comparability graph with an induced subgraph H. A modular decomposition T of G gives us a restricted modular decomposition $(T|_H, D)$ of H as follows; see Figure 3. We obtain $T|_H$ from T by (i) removing all leaves that do not correspond to a vertex of H and then (ii) iteratively contracting all inner nodes of degree at most 2. With a bottom-up traversal we can compute $T|_H$ in time linear in the size of T. A node μ in $T|_H$ *stems from* $\mathrm{lca}_T(L(\mu))$. Every node $\mu \in T|_H$ that stems from a prime node $\mu' \in T$ we label as prime and set D_μ to a transitive orientation of $G[\mu']$ restricted to the edges of H. The remaining nodes are labeled according to the type of their quotient graph. Note that $H[\mu]$ is isomorphic to an induced subgraph of $G[\mu']$.

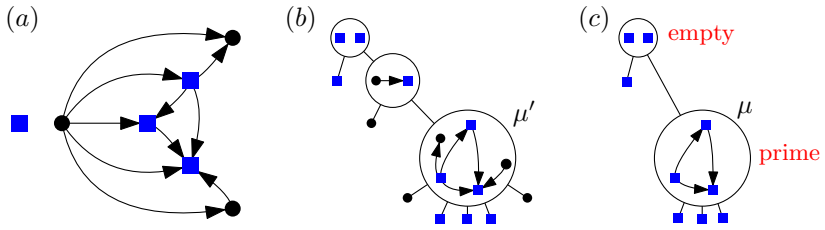


Figure 3 (a) A graph G with an induced subgraph H (blue square vertices). (b) A modular decomposition T of G with a transitive orientation. (c) The restricted modular decomposition $(T|_H, D)$ of H derived from the transitive orientation in (b). Note that $H[\mu]$ is a clique but μ is labeled prime.

▶ **Lemma 4** ($\star$). *Let T be a modular decomposition of a graph G with an induced subgraph H. Then $\text{to}(T|_H, D)$ is the set of orientations of H extendable to transitive orientations of G.*

Consider the canonical modular decompositions $T^1, \ldots, T^r$ of the input graphs $G_1, \ldots, G_r$, and let $(T^1|_H, D_1), \ldots, (T^r|_H, D_r)$ be the corresponding restricted modular decompositions. Then we are interested in the intersection $\text{to}(G_1, \ldots, G_r) = \bigcap_{i=1}^{r} \text{to}(T^i|_H, D_i)$ since it contains all transitive orientations of H that can be extended to all input graphs. However, the trees $T^1|_H, \ldots, T^r|_H$ have different shapes, which makes it difficult to compute a representation of $\text{to}(G_1, \ldots, G_r)$ directly. Instead, we describe $\text{to}(T^1|_H, D_1), \ldots, \text{to}(T^r|_H, D_r)$ (whose intersection is simple to compute) with constraints on the canonical modular decomposition B of H.

Let (T, D) be a restricted modular decomposition of H and let B be the canonical modular decomposition of H. We collect constraints on the orientations of individual nodes μ of B that are imposed by $\text{to}(T, D)$. Afterwards we show that the established constraints are sufficient to describe $\text{to}(T, D)$. If μ is empty, then $H[\mu]$ is empty and has a unique transitive orientation, which requires no constraints. The other types of μ are discussed in the following two sections.

4.1 Constraints for Prime Nodes

In this section we observe that prime nodes in B correspond to prime nodes in (T, D) and that the dependencies between their orientations can be described with 2-SAT formulas. Recall that the transitive orientation of a prime comparability graph is unique up to reversal. We consider each prime node $\mu \in B$ equipped with a transitive orientation D_μ, also called a *default orientation*. All default orientations for B can be computed in linear time [30].

▶ **Lemma 5.** *Let $\mu \in B$ be prime. Then $\mu' = \text{lca}_T(L(\mu))$ is prime, every μ-set is a μ'-set, and for any edge uv with $\text{rep}_B(uv) \in H[\mu]$ we have $\text{rep}_T(uv) \in H[\mu']$.*

Proof. We first show that every μ-set is also a μ'-set. Assume there is a μ-set U that is not a μ'-set. Since $U \subseteq L(\mu) \subseteq L(\mu')$, there exist two vertices $u \neq v \in U$ with $\lambda = \text{rep}_{\mu'}(u) = \text{rep}_{\mu'}(v)$. From $L(\lambda)$ being a module of $H[L(\mu')]$ and $L(\mu) \subseteq L(\mu')$ it follows that $X = L(\lambda) \cap L(\mu)$ is a module of $H[L(\mu)]$. By the definition of μ', we have $X \subsetneq L(\mu)$. Since μ is prime, there is a child ν of μ with $u, v \in X \subseteq L(\nu)$ contradicting U being a μ-set.

As a direct consequence of μ-sets being μ'-sets, we also have for any edge uv with $\text{rep}_B(uv) \in H[\mu]$ that $\text{rep}_T(uv) \in H[\mu']$ since $\{u, v\}$ is a μ-set. Now let U be a maximal μ-set. Then $H[U]$ is isomorphic to $H[\mu]$ and thus prime. Since U is also a μ'-set and the subgraph of $H[\mu']$ induced by the vertices representing U is isomorphic to $H[U]$, μ' is neither empty, nor complete and thus prime. ◀

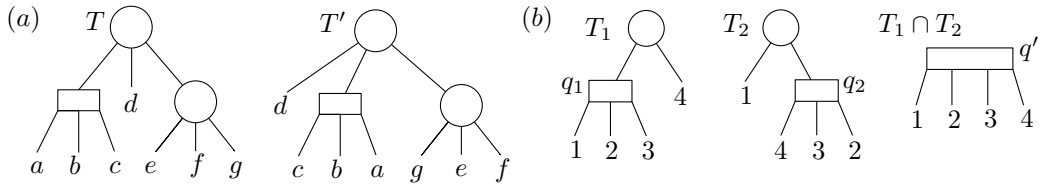


Figure 4 (a) Two equivalent PQ-trees T, T' with $fr(T) = abcdefg$ and $fr(T') = dcbagef$. (b) The Q-node q' in $T_1 \cap T_2$ contains q_1 forwards and q_2 backwards.

For a modular decomposition T' of H with a node λ and $O \in \text{to}(T')$ let $O_{\downarrow\lambda}$ denote the orientation of the quotient graph $H[\lambda]$ we get from O. Note that for a transitive orientation D_λ of $H[\lambda]$ and a λ-set U each orientation $O \in \text{to}(T')$ with $O_{\downarrow\lambda} = D_\lambda$ gives us the same orientation on $H[U]$. We say that D_λ *induces* this orientation on $H[U]$.

Let $\mu \in B$ be prime, let $\mu' = \text{lca}_T(L(\mu))$ and let U be a maximal μ-set (and by Lemma 5 a μ'-set). We set $D^\delta_{\mu'} = D_{\mu'}$ if D_μ, $D_{\mu'}$ induce the same transitive orientation on the prime graph $H[U]$ and we set $D^\delta_{\mu'} = D^{-1}_{\mu'}$ if the induced orientations are the reversal of each other. Note that $D^\delta_{\mu'}$ does not depend on the choice of U. From the definition of $D^\delta_{\mu'}$ and the observation that $O_{\downarrow\mu}, O_{\downarrow\mu'}$ are both determined by O restricted to $H[U]$ we directly get the following lemma.

▶ **Lemma 6.** *For $O \in \text{to}(T, D)$ we have $O_{\downarrow\mu} = D_\mu \Leftrightarrow O_{\downarrow\mu'} = D^\delta_{\mu'}$*

We express the choice of a transitive orientation for a prime node μ by a Boolean variable x_μ that is `true` for the default orientation and `false` for the reversed orientation.

According to Lemma 6 we set ψ_μ to be $x_\mu \leftrightarrow x_{\mu'}$ if $D^\delta_{\mu'} = D_{\mu'}$ and $x_\mu \not\leftrightarrow x_{\mu'}$ if $D^\delta_{\mu'} = D^{-1}_{\mu'}$. Note that for a prime node $\mu' \in T$ there may exist more than one prime node μ in B such that $\mu' = \text{lca}_T(L(\mu))$, and we may hence have multiple prime nodes that are synchronized by these constraints. We describe these dependencies with the formula $\psi = \bigwedge_{\mu \in P(B)} \psi_\mu$. With the above meaning of variables, any choice of orientations for the prime nodes of B that can be induced by T necessarily satisfies ψ. With Lemma 1 we can compute ψ efficiently.

▶ **Lemma 7 (⋆).** *We can compute ψ in $O(|V(H)| + |E(H)|)$ time.*

4.2 Constraints for Complete Nodes

Next we consider the case where μ is complete. The edges represented in $H[\mu]$ may be represented by edges in more than one quotient graph in T, each of which can be complete or prime. Depending on the type of the involved quotient graphs in T we get new constraints for the orientation of $H[\mu]$.

Note that choosing a transitive orientation of $H[\mu]$ is equivalent to choosing a linear order of the vertices of $H[\mu]$. As we will see, each node ν of T that represents an edge of $H[\mu]$ imposes a consecutivity constraint on a subset of the vertices of $H[\mu]$. Therefore, the possible orders can be represented by a *PQ-tree* that allows us to represent all permissible permutations of the elements of a set U in which certain subsets $S \subseteq U$ appear consecutively.

PQ-trees were first introduced by Booth and Lueker [7, 8]. A PQ-tree T over a finite set U is a rooted tree whose leaves are the elements of U and whose internal nodes are labeled as P- or Q-nodes. A P-node is depicted as a circle, a Q-node as a rectangle; see Figure 4. Two PQ-trees T and T' are *equivalent* , if T can be transformed into T' by arbitrarily permuting the children of arbitrarily many P-nodes and reversing the order of arbitrarily many Q-nodes; see Figure 4a. A transformation that transforms T into an equivalent tree T' is called an

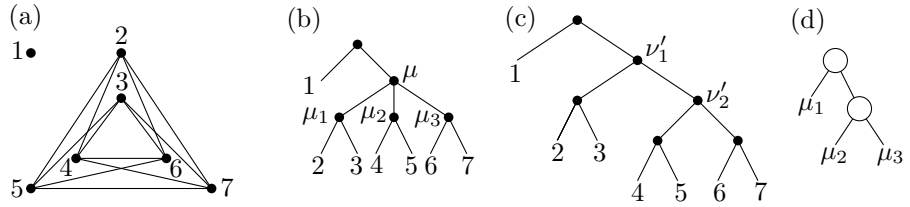


Figure 5 (a) A graph H. (b) The canonical modular decomposition B of H. (c) A restricted modular decompositions T of H. (d) The PQ-tree S_μ. The active nodes of T with regard to the μ-set $\{2, 4, 6\}$ are ν'_1, ν'_2, 2, 4 and 6.

equivalence transformation. The *frontier* $\mathrm{fr}(T)$ of a PQ-tree T is the order of its leaves from left to right. The tree T *represents* the frontiers of all equivalent *PQ*-trees. The PQ-tree that does not have any nodes is called the *null tree*.

Let T_1, T_2 be two PQ-trees over a set U. Their *intersection* $T = T_1 \cap T_2$ is a PQ-tree that represents exactly the linear orders of U represented by both T_1 and T_2. It can be computed in $O(|U|)$ time [7]. For every Q-node q in T_1 node $q' = \mathrm{lca}_T(L(q))$ is also a Q-node. We say that q' *contains* q *forwards*, if $T_1[q]$ can be transformed by an equivalence transformation that does not reverse q into a PQ-tree T' such that $\mathrm{fr}(T[q])$ contains $\mathrm{fr}(T[q'])$; see Figure 4b. Else q' *contains* q *backwards*. Similarly every Q-node in T_2 is contained in exactly one Q-node in T (either forwards or backwards). Haeupler et al. [20] showed that one can modify Booth's algorithm such that given two PQ-trees T_1, T_2 it not only outputs $T_1 \cap T_2$ but also for every Q-node in T_1, T_2 which Q-node in T contains it and in which direction.

▶ **Lemma 8** (⋆). *Let $T_1, \ldots, T_k$ be PQ-trees over a set U. Then we can compute their intersection $T = \bigcap_{i=1}^{k} T_i$ and determine for every Q-node q in $T_1, \ldots, T_k$ the Q-node in T that contains q and in which direction in $O(k \cdot |U|)$ time.*

Let $\mu' = \mathrm{lca}_T(L(\mu))$ for the rest of this section and let $U \subseteq V$ be a maximal μ-set. We call a node of T *active* if it is either a leaf in U or if at least two of its subtrees contain leaves in U. Denote by A the set of active nodes in T and observe that A can be turned into a tree S_μ by connecting each node $\nu' \in A \setminus \{\mu'\}$ to its lowest active ancestor; see Figure 5. Let now $\vec{B}[H]$ be an orientation of the quotient graphs of B induced by an orientation $\vec{T}[G] \in \mathrm{to}(T, D)$ and consider a node $\nu' \neq \mu'$ of S_μ. Let $X = U \cap L(\nu')$ and let $Y = U \setminus L(\nu')$. Since U is a μ-set of a complete node, any pair of vertices in $X \times Y$ is adjacent. Moreover, for each $y \in Y$, the edges from y to X are all oriented towards X, or they are all oriented towards y, since every node of T that determines the orientation of such an edge contains all vertices of X in a single child. This implies that in the order of the μ-set given by the order of $H[\mu]$, the set $L(\nu')$ is consecutive. Moreover, if ν' is prime, its default orientation $D_{\nu'}$ induces a total order on the active children of ν' that is fixed up to reversal. Hence we turn S_μ into a PQ-tree by first turning all complete nodes into P-nodes and all prime nodes into Q-nodes with the children ordered according to the linear order determined by the default orientation which we call the *initial* order of the Q-node. Finally, we replace each leaf $v \in U$ by the corresponding vertex $\mathrm{rep}_\mu(v)$ of $H[\mu]$; see Figure 5. As argued above, the linear order of $H[\mu]$ is necessarily represented by S_μ.

We show that tree S_μ is independent from the choice of the maximal μ-set U. We use that any node of T has a laminar relation to the children of μ.

▶ **Lemma 9** (⋆). *For any child μ_1 of μ and any node κ' of $V(T)$ we have $L(\mu_1) \subseteq L(\kappa') \cap L(\mu)$ or $L(\kappa') \cap L(\mu) \subseteq L(\mu_1)$.*

▶ **Lemma 10** (⋆). *Let S^1_μ, S^2_μ be the PQ-trees for two maximal μ-sets U_1, U_2. Then $S^1_\mu = S^2_\mu$.*

By construction, each Q-node q of S_μ stems from a prime node ν' in T, and the orientation of $T[\nu']$ determines the orientation of q, namely q is reversed if and only if $T[\nu']$ is oriented as $D^{-1}_{\nu'}$. Since a single prime node ν' of T may give rise to Q-nodes in several PQ-trees S_μ, we need to ensure that the orientations of these Q-nodes are either all consistent with the default orientation of $T[\nu']$ or they are all consistent with its reversal. To model this, we introduce a Boolean variable x_q for each Q-node q in one of the PQ-trees with the interpretation that $x_q = \texttt{true}$ if and only if q has its initial order. We require x_q to be equal to the variable that orients the prime node corresponding to q. More precisely, for every prime node ν' in $T[\mu']$ that gives rise to q we add the constraint $(x_{\nu'} \leftrightarrow x_q)$ to χ_μ, where the variable $x_{\nu'}$ is the variable that encodes the orientation of the prime node ν'. We construct a Boolean formula by setting $\chi = \bigwedge_{\mu \in K(B)} \chi_\mu$.

▶ **Lemma 11.** *We can compute all PQ-trees S_μ and the formula χ in $O(n+m)$ time.*

Proof. As a preprocessing we run a DFS on T starting at the root and store for every node ν its discovery-time $\nu.d$, i.e., the timestamp when ν is first discovered, and its finish-time $\nu.f$, i.e., the timestamp after all its neighbors have been examined. We also employ the preprocessing from Lemma 1. We construct all PQ-trees and χ with the following steps.

1. Take a maximal μ-set U_μ for every $\mu \in K(B)$.
2. For every $\mu \in K(B)$ compute the set of active nodes and for every active node compute its parent in S_μ.
3. For every $\mu \in K(B)$ determine for each inner node of S_μ whether it is a P- or a Q-node. If it is a Q-node, determine the linear order of its children, and construct the formula χ_μ.

Step 1 can be done in $O(n)$ time by Lemma 1.

For Step 2, note that each active node is a least common ancestor of two leaves in U_μ. While it is easy to get all active nodes as least common ancestors, getting the edges of S_μ requires more work. Observe that the DFS on T visits the nodes of S_μ in the same order as a DFS on S_μ. Consider S_μ embedded such that the children of each node are ordered from left to right by their discovery-times. This also orders the leaves from left to right by their discovery-times. Let λ be an inner node of S_μ. Let λ_1, λ_2 be two neighboring children of λ with λ_1 to the left of λ_2. Then λ is the least common ancestor of the rightmost leaf in $L(\lambda_1)$ and the leftmost leaf in $L(\lambda_2)$. Hence, each node of S_μ is a least common ancestor for a consecutive pair of leaves.

We add for every node u in a set U_μ a tuple $(\mu, u.d)$ to an initially empty list L. We then sort the tuples in L in linear time using radix sort [13]. In the sorted list, for every $\mu \in K(B)$ all tuples $(\mu, u.d)$ are consecutive and the consecutive sublist is sorted by discovery time.

For $\mu \in K(B)$ let L_μ be a list containing the vertices in U_μ ordered by their discovery time which we get directly from the consecutive sublist of L containing the tuples corresponding to μ. For every pair $u, v \in U_\mu$ adjacent in L_μ we compute $\lambda = \mathrm{lca}_T(u, v)$ using the lowest-common-ancestor data structure for static trees by Harel and Tarjan [21] and insert λ into L_μ between u and v. For a vertex $u \in U_\mu$ its parent in S_μ is the neighbor in L_μ that has a lower position in T. Note that u is a descendent in T of all its neighbors in L_μ. Hence if u has two neighbors in L_μ one of them is a descendent of the other. Thus the parent of u in S_μ is the neighbor with the higher discovery time. Now we remove all vertices in U_μ and possible duplicates of the remaining nodes from L_μ. Note that still every λ is a descendent in T of its neighbors in L_μ. Hence we iteratively choose a node λ in L_μ whose discovery time is higher than the discovery time of its neighbors, compute its parent in S_μ by comparing the discovery times of its neighbors with each other and remove λ from L_μ.

In Step 3, we turn each active node that stems from a complete node into a P-node and each active node that stems from a prime node into a Q-node. For a Q-node q that stems from a prime node ν, we determine the linear order of its children as follows. Take the set X of vertices of $H[\nu]$ that correspond to children of μ in S_μ, determine the orientation of the complete graph on X induced by D_ν and sort it topologically. In total this take $O(n+m)$ time for all active nodes in all PQ-trees. Using the information computed up to this point, it is straightforward to output the formula χ. ◀

Finally, we combine the constraints from the complete nodes with the constraints from the prime nodes by setting $\varphi_T = \psi \wedge \chi$. The formula φ_T allows us to describe a restricted set of transitive orientations of G. We define $S_T = \{S_\mu \mid \mu \in K(B)\}$. The canonical modular decomposition (B, S_T, φ_T) of H where every complete node is labeled with the corresponding PQ-tree together with φ_T we call a *constrained* modular decomposition.

We say that a transitive orientation O of H *induces a variable assignment satisfying* φ_T if it induces an assignments of the variables corresponding to prime nodes in B and Q-nodes such that φ_T is satisfied for an appropriate assignment for the variables corresponding to prime nodes in T. Let $\mathrm{to}(B, S_T, \varphi_T)$ denote the set containing all transitive orientations $O \in \mathrm{to}(B)$ where for every complete node $\mu \in B$ the order $O_{\downarrow\mu}$ corresponds to a total order represented by S_μ and that induces a variable assignment that satisfies φ_T.

4.3 Correctness

We now show that $\mathrm{to}(B, S_T, \varphi_T) = \mathrm{to}(T, D)$. To this end we use that Lemma 5 allows to find for an edge uv that is represented in a prime node μ of B the prime node $\mu' = \mathrm{lca}_T(L(\mu))$ of T where it is represented. This allows us to establish the identity of certain nodes. The following lemma does something similar for complete nodes.

▶ **Lemma 12.** *Let uv, wx be edges of H represented in complete nodes μ, ν of B and by the same edge in a complete node of T. Then $\mu = \nu$.*

Proof. Let ω' be the node in T with $\mathrm{rep}_T(uv) \in H[\omega']$. Assume $\mu \neq \nu$. Then one of them is the ancestor of the other or they are both distinct from $\lambda = \mathrm{lca}_B(\mu, \nu)$. First consider the case that μ is an ancestor of ν. Then μ has a child μ_1 such that $L(\nu) \subseteq L(\mu_1)$. Note that $\mathrm{rep}_\mu(u) \neq \mathrm{rep}_\mu(v)$, hence we have $\mathrm{rep}_\mu(u) \neq \mu_1$ or $\mathrm{rep}_\mu(v) \neq \mu_1$. Without loss of generality assume $\mathrm{rep}_\mu(u) \neq \mu_1$. Since $u \in L(\omega') \cap L(\mu) \setminus L(\mu_1)$ we have $L(\mu_1) \subseteq L(\omega')$ by Lemma 9 and analogously it follows that $L(\mathrm{rep}_\mu(u)) \subseteq L(\omega')$. Since $\mathrm{rep}_T(wx) = \mathrm{rep}_T(uv) \in H[\omega']$ it is $\omega' = lca_T(L(\mu_1)$.

Assume that μ_1 is prime. Then by Lemma 5, ω' is prime which is a contradiction to the assumption that ω' is complete. Hence μ_1 must be complete but as B is the modular decomposition of H, no two adjacent nodes in B are complete. Thus μ is not an ancestor of ν and analogously we get that ν is not an ancestor of μ.

It remains to consider the case that $\nu \neq \lambda \neq \mu$. Let λ_1 be the child of λ such that $L(\mu) \subseteq L(\lambda_1)$ and let λ_2 be the child of λ such that $L(\nu) \subseteq L(\lambda_2)$. Again λ has to be complete since otherwise by Lemma 5 ω' would be prime which is a contradiction. By Lemma 9 we have that $L(\lambda_1) \cup L(\lambda_2) \subseteq L(\omega')$.

Assume that λ_1 is prime. Then by Lemma 5, ω' is prime which leads to a contradiction. Hence λ_1 must be complete but again as B is the modular decomposition of H, no two adjacent nodes in B are complete. ◀

▶ **Theorem 13.** *Let B be the canonical modular decomposition for a graph H and let T be a restricted modular decomposition for H. Then* $\mathrm{to}(B, S_T, \varphi_T) = \mathrm{to}(T, D)$ *and* $\mathrm{to}(B, S_T, \varphi_T)$ *can be computed in time that is linear in the size of H.*

Proof. Let $O_H \in \mathrm{to}(T, D)$ and let $\vec{B}[H]$ be the orientation of the quotient graphs of B inducing O_H. Then we have already seen that it is necessary that every complete node μ in B is oriented according to a total order represented by S_μ and that O_H induces a variable assignment that satisfies φ_T. Hence $O_H \in \mathrm{to}(B, S_T, \varphi_T)$.

Conversely, let $O_H \in \mathrm{to}(B, S_T, \varphi_T)$ and assume $O_H \notin \mathrm{to}(T, D)$. Then O_H is either not represented by T or does not induce D. I.e., O_H contains two directed edges uv, wx with $\mathrm{rep}_T(uv)$ and $\mathrm{rep}_T(wx)$ in the same quotient graph $H[\omega']$, such that $\mathrm{rep}_T(uv) = \mathrm{rep}_T(xw)$, or ω' is prime and $\mathrm{rep}_T(uv) \in D_{\omega'}$ but $\mathrm{rep}_T(wx) \in D_{\omega'}^{-1}$. Note that if ω' is prime, then the first case implies the second one. Let $\mu = \mathrm{lca}_B(u, v)$ and $\nu = \mathrm{lca}_B(w, x)$. We distinguish cases based on the types of μ and ν. Let $\mu' = \mathrm{lca}_T(L(\mu))$, $\nu' = \mathrm{lca}_T(L(\nu))$ and let $\vec{B}[H]$ be the orientation of the quotient graphs of B that induces O_H. Without loss of generality, assume $\mathrm{rep}_B(uv) \in D_\mu$ and $\mathrm{rep}_B(wx) \in D_\nu$.

Case 1: μ and ν are both prime. By Lemma 5 we have that $\mu' = \omega' = \nu'$ is prime. By construction, φ_T enforces that uv, wx are either represented in $D_{\omega'}$ or in $D_{\omega'}^{-1}$. Hence, this case does not occur.

Case 2: μ is prime and ν is complete. By Lemma 5 we have that $\mu' = \omega'$ is prime. Let U be a ν-set containing w and x. Since $\mathrm{rep}_T(wx) \in H[\omega']$, node ω' is active with respect to U. Hence the PQ-tree S_ν contains a Q-node q that stems from ω'.

By construction φ_T contains the constraints $(x_{\omega'} \leftrightarrow x_q)$ and $(x_{\omega'} \leftrightarrow x_\mu)$ but $\vec{B}[H]$ induces $x_\mu = \texttt{true}$, $x_q = \texttt{false}$. Hence the variable assignment induced by $\vec{B}[H]$ does not satisfy φ_T and thus $O_H \notin \mathrm{to}(B, S_T, \varphi_T)$.

Case 3: μ is complete and ν is prime. Similar to Case 2.

Case 4: μ, ν are both complete. Here we further distinguish two subcases depending on the type of ω'. First assume that ω' is prime. Let V_1' be a μ-set containing u, v and let V_2' be a ν-set containing w, x. Since $\mathrm{rep}_T(uv)$ and $\mathrm{rep}_T(wx)$ are edges in $H[\omega']$, node ω' is active with respect to both V_1', V_2'. Hence S_μ, S_ν contain Q-nodes q_1, q_2 stemming from ω'. By construction φ_T contains the constraints $(x_{\omega'} \leftrightarrow x_{q_1})$ and $(x_{\omega'} \leftrightarrow x_{q_2})$, but $\vec{B}[H]$ induces $x_{q_1} = \texttt{true}$, $x_{q_2} = \texttt{false}$. Hence the variable assignment induced by O_H does not satisfy φ_T and thus $O_H \notin \mathrm{to}(B, S_T, \varphi_T)$.

It remains to consider the case that ω' is complete. By Lemma 12 we have $\mu = \nu$. Since ω' is not prime, we must have $\mathrm{rep}_T(uv) = \mathrm{rep}_T(xw)$ by assumption. Let U be a μ-set. Since $\mathrm{rep}_T(uv) = \mathrm{rep}_T(xw) \in H[\omega']$, node ω' is active with respect to U and $\mathrm{rep}_{\omega'}(u) = \mathrm{rep}_{\omega'}(x)$, $\mathrm{rep}_{\omega'}(v) = \mathrm{rep}_{\omega'}(w)$. Hence S_μ contains a P-node that stems from ω' and demands a total order of the children of μ where $\mathrm{rep}_\mu(u)$, $\mathrm{rep}_\mu(x)$ are either both smaller than both $\mathrm{rep}_\mu(v)$, $\mathrm{rep}_\mu(w)$, or $\mathrm{rep}_\mu(u)$, $\mathrm{rep}_\mu(x)$ are both greater than both $\mathrm{rep}_\mu(v)$, $\mathrm{rep}_\mu(w)$. Since $\vec{B}[H]$ induces $\mathrm{rep}_\mu(u) < \mathrm{rep}_\mu(v)$ but $\mathrm{rep}_\mu(x) > \mathrm{rep}_\mu(w)$, $H[\mu]$ is not oriented according to a total order represented by S_μ and thus $O_H \notin \mathrm{to}(B, S_T, \varphi_T)$.

By Lemmas 7 and 11 the formula $\varphi_T = \psi \wedge \chi$ and S_T can be computed in linear time. ◀

Let T be a modular decomposition of a graph G with an induced subgraph H. Let B be the canonical modular decomposition of H and let $T|_H$ be the restricted modular decomposition for H we get from T. From Lemma 4 and Theorem 13 we directly get the following corollary.

▶ **Corollary 14.** *The set* $\mathrm{to}(B, S_{T|_H}, \varphi_{T|_H})$ *contains exactly those transitive orientations of* H *that can be extended to a transitive orientation of* G*.*

Let $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ be constrained modular decompositions for H. Let $S_\mu = \bigcap_{i=1}^r S_\mu^i$ and $S = \{S_\mu \mid \mu \in K(B)\}$. The intersection (B, S, φ) of $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ is the constrained modular decomposition of H where every complete node μ is labeled with the PQ-tree S_μ equipped with the 2-Sat-formula $\varphi = (\bigwedge_{i=1}^r \varphi_i) \wedge \varphi'$ where φ' synchronizes the Q-nodes in the S_μ's with the Q-nodes in the S_μ^i's as follows. Recall that for every Q-node q in a tree S_μ^i there exists a unique Q-node q' in S_μ that contains q; either forward or backward. For every $i \in \{1, \ldots, r\}$, every $\mu \in K(B)$ and every Q-node q in S_μ^i we determine the Q-node q' in S_μ that contains q and add the clause $(x_q \leftrightarrow x_{q'})$ if q has its forward orientation in q' and $(x_q \not\leftrightarrow x_{q'})$ otherwise.

▶ **Lemma 15 (⋆).** *It is* $\mathrm{to}(B, S, \varphi) = \bigcap_{i=1}^r \mathrm{to}(B, S^i, \varphi_i)$ *and we can compute* (B, S, φ) *in linear time from* $\{(B, S^i, \varphi_i) \mid 1 \le i \le r\}$*.*

Now consider the case where $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ are the constrained modular decompositions we get from the restricted modular decompositions $T_1|_H, \ldots, T_r|_H$. By Lemma 12 and Theorem 13 we have $\mathrm{to}(B, S, \varphi) = \bigcap_{i=1}^r \mathrm{to}(B, S_i, \varphi_i) = \bigcap_{i=1}^r \mathrm{to}(T_i, D)$ and by Lemma 4 $G_1, \ldots, G_r$ are simultaneous comparability graphs if and only if φ is satisfiable and S does not contain the null tree.

▶ **Theorem 16.** *SimOrient can be solved in linear time.*

Proof. Let $G_1 = (V_1, E_1), \ldots, G_r = (V_r, E_r)$ be r-sunflower graphs with $n_i = |V_i|$ and $m_i = |E_i|$ for all $1 \le i \le r$ and let $n = \sum_{i=1}^r n_i$, $m = \sum_{i=1}^r m_i$. We solve SimOrient as follows.

1. Compute the canonical modular decomposition T_i for every G_i and the canonical modular decomposition B of H in $O(n + m)$ time by McConnell and Spinrad [30].
2. Compute $T_i|_H$ for every i in $O(n)$ time in total.
3. Compute (B, S^i, φ_i) for every i, in $O(n + m)$ time in total by Theorem 13.
4. Compute (B, S, φ) in linear time by Lemma 12.
5. Check whether S contains the null tree and whether φ is satisfiable in linear time.

We execute Step 5 as follows. For $i \in \{1, \ldots, r\}$, φ_i contains one variable and one constraint per prime node in B, one variable per prime node in $T_i|_H$ and one variable and one constraint per Q-node in S_μ^i. Since S_μ^i has $O(n_i)$ nodes, it contains $O(n_i)$ Q-nodes. Hence in total every φ_i contains $O(n_i)$ variables and clauses. Note that φ' contains one clause per Q-node in the PQ-trees in S. Hence φ' contains $O(n)$ clauses and variables and thus the 2-SAT formula φ can be solved in $O(n)$ time by Aspvall et al. [3].

If S does not contain the null tree and φ has a solution, we get simultaneous transitive orientations of $G_1, \ldots, G_r$ in linear time by proceeding as follows. We orient every complete quotient graph of B according to a total order induced by the corresponding PQ-tree where every Q-node is oriented according to the solution of φ. For a prime quotient graph $H[\mu]$ we choose D_μ if in the chosen solution of φ we have $x_\mu = \texttt{true}$ and D_μ^{-1} otherwise. Together, all these orientations of quotient graphs of B induce a transitive orientation on H and by applying the linear-time algorithm from Section 3 to solve OrientExt we can extend it to a transitive orientation of G_i for every $i \in \{1, \ldots, r\}$. ◀

5 Applications to Permutation Graphs

With the results from Section 3 and 4 we can also solve RepExt(Perm) and SimRep* (Perm) efficiently. For RepExt(Perm) we exploit that a given partial representation D' of a permutation graph G is extendible if and only if for every prime node μ in the canonical modular decomposition of G the partial representation of $G[\mu]$ induced by D' is extendible. Since $G[\mu]$ has only a constant number of representations this can be checked in linear time.

▶ **Theorem 17** ($\star$). *RepExt(Perm) can be solved in linear time.*

Jampani and Lubiw showed that sunflower permutation graphs are simultaneous permutation graphs if and only if they are simultaneous comparability graphs and simultaneous co-comparability graphs [23]. We show that it is possible to use the algorithm from Section 4 to solve SimOrient also for co-comparability graphs and thus SimRep*(Perm) for given sunflower permutation graphs $G_1, \ldots, G_r$ with shared subgraph H in linear time. Recall that a graph G and its complement have the same canonical modular decomposition [16]. In linear time we can not explicitly compute the complements of the input graphs and the corresponding quotient graphs. Hence we can not store a default orientation for the prime quotient graphs $\overline{H}[\mu]$. We can, however compute a *default* permutation diagram representing $H[\mu]$ from its default orientation. This suffices to answer all queries concerning the default orientation of $\overline{H}[\mu]$ in linear time.

▶ **Theorem 18** ($\star$). *SimRep*(Perm) can be solved in linear time.*

Switching a vertex v in a circular permutation graph G, i.e. connecting it to all vertices it was not adjacent to in G and removing all edges to its former neighbors, gives us the graph $G_v = (V, E_v)$ with $E_v = (E \setminus \{vx \in E \mid x \in V\}) \cup \{vx \mid x \in V, vx \notin E\}$ [33]. The graph we obtain by switching all neighbors of a vertex v we denote by $G_{N(v)}$. Switching the neighborhood of a vertex v reduces RepExt(CPerm) and SimRep*(CPerm) to RepExt(Perm) and SimRep*(Perm). Since $G_{N(v)}$ may potentially have a quadratic number of edges we need quadratic time to solve SimRep*(CPerm).

▶ **Theorem 19** ($\star$). *SimRep*(CPerm) can be solved in $O(n^2)$ time.*

For RepExt(Perm) however we can choose a vertex v of minimum degree which ensures that the size of $G_{N(v)}$ is linear in the size of G [36]. Hence the problem can be solved in linear time.

▶ **Theorem 20** ($\star$). *RepExt(CPerm) can be solved in linear time.*

6 Conclusion

We showed that the orientation extension problem and the simultaneous orientation problem for sunflower comparability graphs can be solved in linear time using the concept of modular decompositions. Further we were able to use these algorithms to solve the partial representation problem for permutation and circular permutation graphs and the simultaneous representation problem for sunflower permutation graphs also in linear time. For the simultaneous representation problem for circular permutation graphs we gave a quadratic-time algorithm.

It remains an open problem whether the simultaneous representation problem for sunflower circular permutation graphs can be solved in subquadratic time. Furthermore it would be interesting to examine whether the concept of modular decomposition is also applicable to

solve the partial representation and the simultaneous representation problem for further graph classes, e.g. trapezoid graphs. There may also be other related problems that can be solved for comparability, permutation and circular permutation graphs with the concept of modular decompositions.

References

1 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Transactions on Algorithms*, 11(4):32:1–32:42, 2015.

2 Patrizio Angelini, Giordano Da Lozzo, and Daniel Neuwirth. On some $\mathcal{NP}$-complete SEFE problems. In Sudebkumar Prasant Pal and Kunihiko Sadakane, editors, *In Proceedings of the 8th International Workshop on Algorithms and Computation (WALCOM'14)*, volume 8344 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 2014. doi:10.1007/978-3-319-04657-0_20.

3 Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

4 Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous embedding of planar graphs. *Computing Research Repository*, abs/1204.5853, 2012. arXiv:1204.5853.

5 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Transactions on Algorithms*, 12(2):16:1–16:46, 2015. doi:10.1145/2738054.

6 Jan Bok and Nikola Jedličková. A note on simultaneous representation problem for interval and circular-arc graphs. *Computing Research Repository*, 2018. arXiv:1811.04062.

7 Kellogg S. Booth. *PQ-tree algorithms*. PhD thesis, University of California, Berkeley, 1975.

8 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

9 Peter Brass, Eowyn Cenek, Cristian A. Duncan, Alon Efrat, Cesim Erten, Dan P. Ismailescu, Stephen G. Kobourov, Anna Lubiw, and Joseph S.B. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007. doi:10.1016/j.comgeo.2006.05.006.

10 Steven Chaplick, Paul Dorbec, Jan Kratochvíl, Mickael Montassier, and Juraj Stacho. Contact representations of planar graphs: Extending a partial representation is hard. In Dieter Kratsch and Ioan Todinca, editors, *40th International Workshop on Graph-theoretic concepts in computer science (WG'14)*, volume 8747 of *Lecture Notes in Computer Science*, pages 139–151. Springer, 2014.

11 Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending partial representations of circle graphs. *Journal of Graph Theory*, 91(4):365–394, 2019.

12 Steven Chaplick, Philipp Kindermann, Jonathan Klawitter, Ignaz Rutter, and Alexander Wolff. Extending partial representations of rectangular duals with given contact orientations. In Tiziana Calamoneri and Federico Corò, editors, *Proceedings of the 12th International Conference on Algorithms and Complexity, (CIAC '21)*, volume 12701 of *Lecture Notes in Computer Science*, pages 340–353. Springer, 2021. doi:10.1007/978-3-030-75242-2_24.

13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, Cambridge, 2009.

14 Eduard Eiben, Robert Ganian, Thekla Hamm, Fabian Klute, and Martin Nöllenburg. Extending partial 1-planar drawings. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, volume 168 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.43.

15 Alejandro Estrella-Balderrama, Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous geometric graph embeddings. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Proceedings of 15th International Symposium on Graph Drawing (GD '07)*, pages 280–290. Springer, 2008. doi:10.1007/978-3-540-77537-9_28.

16 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1-2):25–66, 1967.

17 Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous graph embeddings with fixed edges. In Fedor V. Fomin, editor, *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, pages 325–335. Springer, 2006. doi:10.1007/11917496_29.

18 Paul C Gilmore and Alan J Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

19 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, London, 2004.

20 Bernhard Haeupler, Krishnam R. Jampani, and Anna Lubiw. Testing simultaneous planarity when the common graph is 2-connected. *Journal of Graph Algorithms and Applications*, 17(3):147–171, 2013.

21 Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

22 Krishnam Raju Jampani and Anna Lubiw. Simultaneous interval graphs. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC' 10)*, pages 206–217. Springer, 2010. doi:10.1007/978-3-642-17517-6_20.

23 Krishnam Raju Jampani and Anna Lubiw. The simultaneous representation problem for chordal, comparability and permutation graphs. *Journal of Graph Algorithms and Applications*, 16(2):283–315, 2012.

24 Vít Jelínek, Jan Kratochvíl, and Ignaz Rutter. A Kuratowski-type theorem for planarity of partially embedded graphs. *Computational Geometry*, 46(4):466–492, 2013.

25 Pavel Klavík, Jan Kratochvíl, Tomasz Krawczyk, and Bartosz Walczak. Extending partial representations of function graphs and permutation graphs. In Leah Epstein and Paolo Ferragina, editors, *20th Annual European Symposium on Algorithms (ESA'12)*, Lecture Notes in Computer Science, pages 671–682. Springer, 2012.

26 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomáš Vyskočil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 77(4):1071–1104, 2017.

27 Pavel Klavík, Jan Kratochvíl, Yota Otachi, and Toshiki Saitoh. Extending partial representations of subclasses of chordal graphs. *Theoretical Computer Science*, 576:85–101, 2015.

28 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomáš Vyskočil. Extending partial representations of interval graphs. *Algorithmica*, 78(3):945–967, 2017.

29 Tomasz Krawczyk and Bartosz Walczak. Extending partial representations of trapezoid graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'17)*, pages 358–371. Springer, 2017.

30 Ross M McConnell and Jeremy P Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

31 Miriam Münch, Ignaz Rutter, and Peter Stumpf. Partial and simultaneous transitive orientations via modular decomposition, 2022. doi:10.48550/ARXIV.2209.13175.

32 Maurizio Patrignani. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science*, 17(5):1061–1070, 2006.

33 Doron Rotem and Jorge Urrutia. Circular permutation graphs. *Networks*, 12(4):429–437, 1982.

34 Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer. Simultaneous representation of proper and unit interval graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA'19)*, volume 144, page 80. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

35 Marcus Schaefer. Toward a theory of planarity: Hanani-Tutte and planarity variants. In *20th International Symposium on Graph Drawing (GD '12)*, pages 162–173. Springer, 2012. doi:10.1007/978-3-642-36763-2_15.

36 R Sritharan. A linear time algorithm to recognize circular permutation graphs. *Networks: An International Journal*, 27(3):171–174, 1996.

Polynomial Threshold Functions for Decision Lists

Vladimir Podolskii
Courant Institute of Mathematical Sciences, New York University, NY, USA
Steklov Mathematical Institute of Russian Academy of Sciences, Moscow, Russian Federation

Nikolay V. Proskurin
HSE University, Moscow, Russian Federation

Abstract

For $S \subseteq \{0,1\}^n$ a Boolean function $f\colon S \to \{-1,1\}$ is a polynomial threshold function (PTF) of degree d and weight W if there is a polynomial p with integer coefficients of degree d and with sum of absolute coefficients W such that $f(x) = \operatorname{sign} p(x)$ for all $x \in S$. We study a representation of decision lists as PTFs over Boolean cubes $\{0,1\}^n$ and over Hamming balls $\{0,1\}^n_{\leq k}$.

As our first result, we show that for all $d = O\left(\left(\frac{n}{\log n}\right)^{1/3}\right)$ any decision list over $\{0,1\}^n$ can be represented by a PTF of degree d and weight $2^{O(n/d^2)}$. This improves the result by Klivans and Servedio [22] by a $\log^2 d$ factor in the exponent of the weight. Our bound is tight for all $d = O\left(\left(\frac{n}{\log n}\right)^{1/3}\right)$ due to the matching lower bound by Beigel [3].

For decision lists over a Hamming ball $\{0,1\}^n_{\leq k}$ we show that the upper bound on weight above can be drastically improved to $n^{O(\sqrt{k})}$ for $d = \Theta(\sqrt{k})$. We also show that similar improvement is not possible for smaller degrees by proving the lower bound $W = 2^{\Omega(n/d^2)}$ for all $d = O(\sqrt{k})$.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Threshold function, decision list, Hamming ball

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.52

Funding *Nikolay V. Proskurin*: This paper was prepared within the framework of the HSE University Basic Research Program.

1 Introduction

The main object of studies in this paper are polynomial threshold functions.

▶ **Definition 1.** *For $S \subseteq \{0,1\}^n$ a Boolean function $f\colon S \to \{-1,1\}$ is called a polynomial threshold function (PTF) of degree d if there is an integer polynomial p of degree d such that $f(x) = \operatorname{sign} p(x)$ for all $x \in S$. If p has integer coefficients, then the weight of a PTF is defined as a sum of absolute values of coefficients in p.*

Polynomial threshold functions have been studied intensively for decades. Much of this work was motivated by questions in computer science [28], and PTFs are now an important object of study in areas such as Boolean circuit complexity [7, 17, 2, 23], learning theory [20, 21, 16, 4, 15], and communication complexity [30].

In this paper, we study PTFs for the class of *decision lists*.

▶ **Definition 2.** *A decision list L on variables $x_1, \dots, x_n$ is a sequence of h pairs and a bit*

$$(\ell_1, b_1), (\ell_2, b_2), \dots, (\ell_h, b_h), b_{h+1},$$

where for all i ℓ_i is a literal (either a Boolean variable x_j or its negation) and $b_i \in \{-1,1\}$. On the input $x \in \{0,1\}^n$ the value of $L(x)$ is equal to b_i if i is the minimal index for which ℓ_i is true and if all ℓ_i are false, the value of $L(x)$ is equal to b_{h+1}.

Decision lists have been widely studied in computational complexity [1, 5, 12, 22]. Partially, their importance stems from computational learning theory where attribute efficient learning of decision lists is an important research direction. One of the approaches to learning decision lists is through their representation as PTFs, to which several known learning algorithms can be applied [18, 21, 22, 27]. This constitutes one of the reasons to study PTF representations of decision lists. Decision lists turned out to be important in other areas of theoretical computer science as well. They provide a good source of lower bounds in the studies of threshold functions, threshold circuits, oracle computations and in other fields [3, 8, 13, 34].

A lot of effort has been put into the studies of effective PTF representations of Boolean functions, and for decision lists in particular. Klivans and Servedio [22] have shown that for every $h < n$ any decision list can be computed by a PTF of degree $O(\sqrt{n}\log n)$ and weight $2^{O(\frac{n}{h}+\sqrt{h}\log^2 n)}$; in terms of degree d it results in $2^{O(\frac{n\log^2 d}{d^2}+d\log d)}$ weight bound. They used it to create the first online learning algorithm for decision lists that is subexponential in both running time and sample complexity; namely, their algorithm runs in time $n^{O(n^{1/3}\log^{1/3} n)}$ and mistake bound $2^{O(n^{1/3}\log^{4/3} n)}$. Servedio, Tan and Thaler proved in [29, Theorem 6] that for any $n^{1/4} \le d \le n$ any decision list $L\colon \{-1,1\}^n \to \{-1,1\}$[1] on n variables has a degree-d PTF of weight $2^{\tilde{O}_n((n/d)^{2/3})}$, where $\tilde{O}_n$ suppresses the polylog factor of n. This bound is weaker for small d, but gives an upper bound for $d \ge \Omega(\sqrt{n})$. As for the lower bounds, Beigel [3] provided a decision list that requires weight of $2^{\Omega(n/d^2)}$ to be computed by a degree-d PTF (we provide the details in Section 3). Servedio, Tan and Thaler [29] also proved a lower bound of $2^{\Omega(\sqrt{n/d})}$ for $d = o\left(\frac{n}{\log^2 n}\right)$, which is stronger than the bound of Beigel for $d = \Omega(n^{1/3})$.

Above we have discussed PTF representations of decision lists over the Boolean cube $\{0,1\}^n$. However, representations over its subsets are also relevant. The case of the Hamming ball $\{0,1\}^n_{\le k}$ consisting of all vectors with at most k 1s received some attention. Long and Servedio [25] gave bounds for the weights of PTFs for degree $d = 1$. Their main motivation to study this setting comes from learning theory: in scenarios involving learning categorical data the common representation for examples is the one-hot encoded vector, which might have an extremely large amount of features, but only a small fraction of them can be active at the same time. The Winnow algorithm used in [22] can be applied to learn functions not only on the Boolean cube but also on its subsets, including $\{0,1\}^n_{\le k}$, so it makes sense to study not only linear representations but also polynomial ones. Boolean functions on Hamming balls are also important outside of learning theory. For example, approximation of such functions by polynomials of low weight arises in the problem of indistinguishability [6, 19], and recently the approximation on Hamming Balls in general received a lot of attention as a stepping stone for the approximation of some important Boolean functions [9, 31], including constant-depth circuits (AC^0) [10, 11, 32, 33].

Our results

First, we address the PTF representations of decision lists over $\{0,1\}^n$. In our first result, we show that for every $h < n$ any decision list can be computed by a PTF of degree $O(\sqrt{n})$ and weight $2^{O(\frac{n}{h}+\sqrt{h}\log n)}$. This improves the result of [22] by a logarithmic factor both in degree and exponent of weight. As a corollary, we slightly improve upon the upper bound for the attribute efficient learning of decision lists: we prove that there is an algorithm that learns decision lists in time $n^{O(n^{1/3})}$ and mistake bound $2^{O(n^{1/3}\log n)}$.

[1] In this result, input Boolean variables range over $\{-1,1\}$.

In terms of degree vs. weight tradeoff, our results imply that for all $d = O\left(\left(\frac{n}{\log n}\right)^{1/3}\right)$ any decision list over $\{0,1\}^n$ can be computed by a PTF of degree d and weight $2^{O(n/d^2)}$. The analogous result that follows the construction in [22] gave the weight upper bound of $2^{O((n\log^2 d)/d^2)}$. Our bound is tight for all $d = O\left(\left(\frac{n}{\log n}\right)^{1/3}\right)$ due to the matching upper bound by Beigel [3] mentioned above.

Clearly, our upper bound for the Boolean cube works also for decision lists over the Hamming ball $\{0,1\}^n_{\leq k}$ as they are just restrictions to a smaller domain. However, the tightness of this bound for the case of Hamming balls has to be investigated. We actually show that the upper bound can be improved to a polynomial in n for d roughly equal to $\sqrt{k}$. More precisely, we show that for $d = \Theta(\sqrt{k})$ any decision list over $\{0,1\}^n_{\leq k}$ can be represented with weight $n^{O(\sqrt{k})}$. However, we show that such an improvement is impossible for smaller degrees. In order to do so, we extend Beigel's lower bound to this setting, showing that there is a decision list that requires weight $2^{\Omega(n/d^2)}$ when computed by a PTF of degree $d = O(\sqrt{k})$.

Our techniques

To prove our upper bounds we adopt the same strategy as in [22]. We first represent a decision list as a sum of sublists, then we pointwise approximate each of the sublists. The improvement comes from the better approximation technique originated by Sherstov [31], which we adapt for decision lists.

For the lower bound, we extend the proof of Beigel [3] to the setting of low-weight inputs. We use the same proof strategy of inductively constructing a sequence of inputs on which the value of the polynomial in PTF grows exponentially. The new ingredient in the proof is to keep the number of 1's in the input low by reusing them from the previous blocks.

Organization

In Section 2 we provide the necessary definitions and theorems. In Section 3 we give our PTF construction for decision lists on $\{0,1\}^n$, and in Section 4 we prove the upper and the lower bounds for representing decision lists as a PTF on $\{0,1\}^n_{\leq k}$.

2 Preliminaries

We use the following notation for the *arithmetization* of a literal ℓ:

$$\tilde{\ell} = \begin{cases} x, & \ell \text{ is an unnegated variable } x, \\ 1-x, & \ell \text{ is a negated variable } \overline{x}, \end{cases}$$

We also use the notations $[n] = \{1,2,\dots,n\}$ and $|||p|||$ for the sum of absolute values of coefficients in p (therefore, the weight of a PTF p is equals to $|||p|||$ if p has integer coefficients).

For technical reasons, we need a modified definition of a decision list, where the last b_i is always equal to 0 instead of ± 1. We call those decision lists *modified*.

As mentioned in the introduction, one of the technical steps involves the pointwise approximation of Boolean functions.

▶ **Definition 3.** *Given a Boolean function $f\colon X \to \{0,1\}$, we say that a polynomial p is ε-approximates the function f iff $\max_{x\in X} |f(x) - p(x)| \leq \varepsilon$. The ε-approximate degree $\deg_\varepsilon(f)$ is the minimum degree such that there exists a polynomial of degree d that ε-approximates for the function f.*

Polynomial threshold functions may be viewed as a generalization of ε-approximators: if p ε-approximates the function f on $X \subseteq \{0,1\}^n$, then $p(x) - \frac{1}{2}$ is a PTF that computes f on X.

One of the commonly used families of polynomials for the Boolean function approximation is the Chebyshev polynomials of the first kind.

▶ **Definition 4.** *The Chebyshev polynomials of the first kind are defined by the recurrence relation* $T_d(x) = 2xT_{d-1}(x) - T_{d-2}(x)$ *with* $T_0(x) = 1$ *and* $T_1(x) = x$.

The solution of the recurrence above gives the following equation.

$$T_d(x) = \frac{1}{2}\left(\left(x - \sqrt{x^2-1}\right)^d + \left(x + \sqrt{x^2-1}\right)^d\right). \tag{1}$$

It is obvious from the definition that $\deg(T_d) = d$ and T_d has integer coefficients. We also need the following claims about the Chebyshev polynomials.

▷ Claim 5 ([31, Proposition 2.6]). For every $\delta > 0$, $T_d(1+\delta) \geq 1 + d^2\delta$.

▷ Claim 6. For every $\delta \in [0;2]$, $T_d(1+\delta) \leq (1+\sqrt{\delta})^{2d}$.

Proof. Note that $\sqrt{(1+\delta)^2-1} = \sqrt{\delta(\delta+2)} \leq 2\sqrt{\delta}$ for $0 \leq \delta \leq 2$. By substituting $x = 1+\delta$ into (1), we get

$$T_d(1+\delta) = \frac{1}{2}\left(\left(1+\delta-\sqrt{(1+\delta)^2-1}\right)^d + \left(1+\delta+\sqrt{(1+\delta)^2-1}\right)^d\right) \leq$$

$$\leq \left(1+\delta+\sqrt{(1+\delta)^2-1}\right)^d \leq (1+\delta+2\sqrt{\delta})^d = (1+\sqrt{\delta})^{2d}.$$

◁

▷ Claim 7 ([6, Section 2.2, Property 5]). $|||T_d||| \leq 2^{2d}$.

Klivans and Servedio used the Expanded-Winnow algorithm [22, Theorem 2] to derive the bounds on learning the decision lists class in the attribute-efficient learning model. It essentially runs the Winnow algorithm [24, Algorithm 4] on a set of all possible monomials of degree up to d. We note that the Winnow algorithm can learn Boolean functions on any domain $X \subseteq \{0,1\}^n$, and therefore we can apply the same bounds from the Expanded-Winnow to learn the decision lists on $\{0,1\}^n_{\leq k}$. Thus, the next theorem implicitly follows from the Expanded-Winnow algorithm.

▶ **Theorem 8.** *Let C be a class of Boolean functions over $S \subseteq \{0,1\}^n$ with the property that each $f \in C$ has a PTF of degree at most d and weight at most W. Then there is an online learning algorithm for C which runs in n^d time per example and has mistake bound $O(W \cdot d^2 \cdot \log n)$.*

3 Decision lists on the Boolean cube

In this section, we prove the upper bound for representing decision lists as a PTF over the Boolean cube. Our proof can be viewed as an improvement of the proof of the following theorem by Klivans and Servedio.

▶ **Theorem 9** ([22, Theorem 7])**.** *Let L be a decision list of length n on $\{0,1\}^n$. Then for any $h < n$, L is computed by a PTF of degree $O(\sqrt{h}\log h)$ and weight $2^{O(n/h+\sqrt{h}\log^2 h)}$.*

We start with the outline of their construction and then proceed to prove the missing pieces to tighten up the upper bound. This construction consists of two parts. The "outer" part is the decomposition of the original decision list into the sum of modified sublists, i.e. we represent the $L(x)$ as the $L(x) = \sum_i^{n/h} a_i L_i(x)$, where each sublist is responsible for the independent block of size roughly h. The "inner" part is the pointwise approximation of the sublist. To achieve a close enough approximation, Klivans and Servedio represent each of the sublists as a sum of conjunctions and approximate each of the conjunctions separately. The degree-weight traidoff is adjusted by the parameter h.

The problem with this approach is that in order to approximate the sublist, one has to very closely approximate the inner conjunctions, i.e. with a precision of at least $O(\frac{1}{h})$. But it was proved in [14] that such an approximation requires the degree of $\Theta(\sqrt{h \log h})$, and while we can improve on the logarithmic factor, we can not get rid of it completely. However, we can notice that the set of conjunctions we are dealing with is not entirely random; in fact, every two successive conjunctions share most of their literals. That means we can adjust our approximator so it would increase it's precision as more terms in conjunction are set to zero. Luckily for us, it was proved in [31] that such an adjustment can be done with the same degree as in the regular approximator. In the next theorem, we adapt it for our needs and also prove the weight bound for it.

▶ **Theorem 10.** *For every $d = \Theta(1)$ and every $\varepsilon = \Theta(1)$ there exists an univariate polynomial $P_{n,d,\varepsilon}(x)$ such that $\deg(P_{n,d,\varepsilon}) = O(\sqrt{n})$, $|||P_{n,d,\varepsilon}||| = 2^{O(\sqrt{n})}$ and*

$$\forall t \in [0;1] \quad |P_{n,d,\varepsilon}(t) - 1| \le \varepsilon,$$

$$\forall t \in (2;n] \quad |P_{n,d,\varepsilon}(t)| \le \frac{\varepsilon}{t^d}.$$

Moreover, there exists a constant $C = n^{O(\sqrt{n})}$ such that $C \cdot P_{n,d,\varepsilon}(x)$ has integer coefficients.

Proof. We can get such a polynomial from [31, Theorem 3.7] (in fact, our statement is much more limited than the original one) with a proven degree bound but with no weight estimate. So it suffices for us to prove that the claimed weight bound holds as well.

The polynomial $P_{n,d,\varepsilon}(x)$ from [31, Theorem 3.7] is given as $P_{n,d,\varepsilon}(x) = p_1(x)^d p_2(p_1(x)) p_3(x)$, where

$$p_1(t) = \frac{T_{d_1+1}\left(1 - 2\frac{t-1}{n-1}\right)}{T_{d_1+1}\left(\frac{n+1}{n-1}\right)}$$

$$p_2(t) = \sum_{i=0}^{D} \binom{i+d-1}{i} (1-t)^i$$

$$p_3(t) = B_{d_3}\left(\frac{1}{e^7} T_{\sqrt{n}}\left(1 + \frac{2-t}{n}\right)\right), \quad B_{d_3}(t) = \sum_{i=\lceil 2.5e^{-7} d_3 \rceil}^{d_3} \binom{d_3}{i} t^i (1-t)^i$$

with $d_1 = \lfloor \sqrt{2(n-1)} \rfloor$, $D = O\left(d + \log \frac{1}{\varepsilon}\right)$ and $d_3 = O\left(\log \frac{1}{\varepsilon}\right)$.

To start with,

$$T_{d_1+1}\left(\frac{n+1}{n-1}\right) \le \left(1 + \sqrt{\frac{2}{n-1}}\right)^{2\sqrt{2(n-1)}} = \left(1 + \frac{2}{\sqrt{2(n-1)}}\right)^{2\sqrt{2(n-1)}} \le e^4,$$

where the first inequality is Claim 5 and the last one is $(1+\frac{a}{x})^x < e^a$ for $a > 0$ and $x > 1$. On the other hand, by Claim 6

$$T_{d_1+1}\left(\frac{n+1}{n-1}\right) \geq 1 + 2(n-1)\cdot\frac{2}{n-1} = 5,$$

so the scaling factor in $p_1(t)$ is bounded by constants, and by Claim 7 $|||p_1||| = 2^{O(\sqrt{n})}$. By multiplying p_1 by $(n-1)^{d_1+1}$, we clear the denominators from $1-2\frac{t-1}{n-1}$, and thus it would have integer coefficients. The same holds for $T_{\sqrt{n}}\left(1+\frac{2-t}{n}\right)$ and $n^{\sqrt{n}}$. As for the $T_{d_1+1}\left(\frac{n+1}{n-1}\right)$ in p_1, we know that it is a rational number $\frac{a}{b}$ with $b \leq (n-1)^{d_1+1}$, which is bounded by constants. Therefore, $a = O((n-1)^{d_1+1})$, and to clear all the denominators from $P_{n,d,\varepsilon}(x)$ it suffices to multiply it by $C = n^{O(\sqrt{n})}$.

Next, both $p_2(t)$ and $B_{d_3}(t)$ have integer and independent of n coefficients, as well as independent of n degrees, so their degrees and weights are constant. Finally, the product of a constant number of terms of degree $O(\sqrt{n})$ and weight $n^{O(\sqrt{n})}$ has the same degree and weight bounds, and the overall weight bound holds. The bound for the constant C holds for the same reasons. ◀

With the new approximator, we are ready to proceed to the main proof of this section.

▶ **Theorem 11.** *Let L be a modified decision list of length h on $\{0,1\}^n$. Then for every $\varepsilon = \Theta(1)$ L can be ε-approximated by a polynomial of degree $O(\sqrt{h})$ and weight $n^{O(\sqrt{h})}$. Moreover, $p(0^n) = 0$.*

Proof. Consider the decision list $L = (\ell_1, b_1), \ldots (\ell_h, b_h), 0$. It is straightforward to check that L can be expressed as $L(x) = \sum_{i=1}^{h} b_i \ell_i \bigwedge_{j=1}^{h-1} \overline{\ell_j}$, where the term corresponding to b_i is non-zero iff ℓ_i is the first condition satisfied in L.

For every i put $T_i(x) = \bigwedge_{j=1}^{h-1} \overline{\ell_j(x)}$ and $A_i(x) = 3(i-1) - 3\tilde{\ell_1} - \ldots - 3\tilde{\ell}_{i-1}$. Notice that $A_i(x) = 0 \Leftrightarrow T_i(x) = 1$ and $A_i(x) \geq 3$ otherwise. Moreover, for every $j < i$ such that $\ell_j = 1$ the value of $A_i(x)$ increases by 3. By corollary, for every $i < j$ we have $\ell_i = 1 \Rightarrow A_i(x) + 3 \leq A_j(x)$ because A_j consists of every zero literal of T_i and $\overline{\ell_i} = 0$.

Now consider $p_i(x) = P_{3h,2,\varepsilon/2}(A_i(x))$ where $P_{3h,2,\varepsilon/2}$ is from Theorem 10. We will prove that $p(x) = \sum_{i=1}^{h} b_i \tilde{\ell_i} p_i(x)$ is the desired polynomial. To start with, if $\ell_i = 0$ then the corresponding term is also equals to zero; in particular, $p(0^n) = 0$. Next, let $j_1 < \ldots < j_m$ be the set of all indexes such that $\ell_{j_i} = 1$. For j_1 we have $A_{j_1}(x) = 0$, and so $\operatorname{sign}(b_{j_1}\tilde{\ell_{j_1}} p_{j_1}(x)) = b_{j_1}$ and $|b_{j_1}\tilde{\ell_{j_1}} p_{j_1}(x)| \geq 1 - \varepsilon/2$. Finally, for the remaining indexes we have $3 \leq A_{j_2}(x) < A_{j_3}(x) < \ldots < A_{j_m}(x) \leq 3h$, and so

$$\left|\sum_{i=2}^{m} b_{j_i}\ell_{j_i} p_{j_i}(x)\right| \leq \sum_{i=2}^{m} |p_{j_i}(x)| \leq \frac{\varepsilon}{2}\sum_{i=3}^{\infty}\frac{1}{i^2} = \frac{\varepsilon}{2}\left(\sum_{i=1}^{\infty}\frac{1}{i^2} - \frac{5}{4}\right) \leq \frac{\varepsilon}{2}\left(\frac{\pi^2}{6} - \frac{5}{4}\right) \leq \frac{\varepsilon}{2}.$$

Therefore, for every $x \in \{0,1\}^n$ we have $|L(x) - p(x)| \leq \varepsilon$ and $p(0^n) = 0$. The bounds on degree and weight follows from the Theorem 10 and the following observation: if $p = a_d x^d + \ldots + a_1 x_1 + a_0$ then

$$|||p(q(x))||| = |||a_d q(x)^d + \ldots + a_1 q(x) + a_0||| \leq |||a_d q(x)^d + \ldots + a_1 q(x)^d + a_0||| =$$

$$= |||q(x)^d(a_d + \ldots + a_1 + a_0)||| \leq |||q|||^d \cdot |||p|||$$

By composing A_i of weight $O(n)$ and degree 1 with $P_{3h,2,\varepsilon/2}$ of weight $2^{O(\sqrt{h})}$ and degree $\sqrt{h}$, we get a polynomial of degree $\sqrt{h}$ and weight at most $n^{O(\sqrt{h})}$. ◀

▶ **Corollary 12.** *Let L be a modified decision list of length h on $\{0,1\}^n$. Then L is computed by a PTF p of degree $O(\sqrt{h})$ and weight $h^{O(\sqrt{h})}$. Moreover, the $p(0^n) = 0$.*

Proof. Multiply the polynomial from the previous theorem by a constant $C = n^{O(\sqrt{n})}$. By Theorem 10, every p_i from the proof of Theorem 11 now has integer coefficients. The resulting polynomial does not pointwise approximate the decision list anymore, but for every $x \in \{0,1\}^n$ we have $|CL(x) - p(x)| \leq C/\varepsilon$ and $p(0^n) = 0$. Therefore, for any sufficiently large constant ε, say $\frac{1}{100}$, Cp is the desired PTF for L. ◀

Using Corollary 12 as the inner approximator, we can use the outer construction by Klivans and Servedio to achieve the final PTF for the decision lists.

▶ **Theorem 13.** *Let L be a decision list of length n on $\{0,1\}^n$. Then for any $h < n$, L is computed by a PTF of degree $O(\sqrt{h})$ and weight $2^{O(n/h+\sqrt{h}\log h)}$.*

The proof goes exactly the same as in [22, Theorem 7], except the inner approximator has a constant precision, which does not affect the overall correctness. For the sake of completeness, we provide the proof in Appendix A.

▶ **Corollary 14.** *Let L be a decision list of length n on $\{0,1\}^n$. Then for any $d < \sqrt{n}$, L is computed by a PTF of degree d and weight $2^{O(n/d^2+d\log d)}$.*

In [3], Beigel proved the lower bound for the weight of PTF for the specific decision list called ODD-MAX-BIT.

▶ **Definition 15.** *The ODD-MAX-BIT$_n$ function on input $x \in \{0,1\}^n$ is equal to $(-1)^i$ where i is the position of the rightmost 1 in x. If $x = 0^n$ then ODD-MAX-BIT$_n(x) = 1$.*

▶ **Theorem 16** ([3])**.** *Let p be a degree d PTF with integer coefficients which computes ODD-MAX-BIT$_n$ on $\{0,1\}^n$. Then $|||p||| = 2^{\Omega(n/d^2)}$.*

Note that Corollary 14 gives a PTF of weight $2^{O(n/d^2)}$ for any decision list and $d = O\left(\left(\frac{n}{\log n}\right)^{1/3}\right)$. Thus, in this range our bound is asymptotically optimal.

Using Theorem 13, we can also provide a slightly more efficient online learning algorithm for decision lists.

▶ **Corollary 17.** *Let L be a decision list of length n on $\{0,1\}^n$. Then L is computed by a PTF of degree $O(n^{1/3})$ and weight $2^{O(n^{1/3}\log n)}$.*

Proof. Apply Theorem 13 with $h = n^{1/3}$. ◀

▶ **Corollary 18.** *There is an algorithm that learns decision lists on $\{0,1\}^n$ in time $n^{O(n^{1/3})}$ and mistake bound $2^{O(n^{1/3}\log n)}$.*

Proof. Follow immediately from the previous corollary and Theorem 8. ◀

4 Decision lists on Hamming balls

In this section, we shift our focus from $\{0,1\}^n$ to $\{0,1\}^n_{\leq k}$. We show that both upper and lower bounds can be generalized on the new domain, and the degree-weight dependency is much more significant than it is on the Boolean cube. To be more precise, we show that if the degree parameter is low, then the Hamming ball scenario is not much different from the Boolean cube one, and we still need an exponential in terms of n weight for a PTF. However, after a certain threshold it can be drastically improved.

4.1 Upper bound

We first start with the upper bound; to achieve it, we can straightforwardly modify the proof of Theorem 11 for the $\{0,1\}^n_{\leq k}$ domain.

▶ **Theorem 19.** *Let L be a modified decision list of length h on $\{0,1\}^n_{\leq k}$. Then for every $\varepsilon = \Theta(1)$ L can be ε-approximated by a polynomial of degree $O(\sqrt{k})$ and weight $n^{O(\sqrt{k})}$. Moreover, $p(0^n) = 0$.*

Proof. Recall the following definitions from the proof of Theorem 11

$$L(x) = \sum_{i=1}^{h} b_i \ell_i \bigwedge_{j=1}^{h-1} \overline{\ell_j} = \sum_{i=1}^{h} b_i \ell_i T_i(x) \qquad A_i(x) = 3(i-1) - \tilde{\ell_1} - \ldots - \tilde{\ell}_{i-1}$$

We need to reexamine $A_i(x)$. If for every $x \in \{0,1\}^n_{\leq k}$ we have $A_i(x) > 0$, then $T_i(x) \equiv 0$ on $\{0,1\}^n_{\leq k}$ and we can safely remove the corresponding term from the sum without affecting it's value on any input. Otherwise, let $x' \in \{0,1\}^n_{\leq k}$ be such an input that $A_i(x') = 0$. Flipping the value of one bit in x' can affect at most one literal in T_i, so if $|x \oplus x'| = 1$ then $A_i(x) \leq 3 + A_i(x')$.

Notice that because we are only interested in $x \in \{0,1\}^n_{\leq k}$, for any x we have $|x \oplus x'| \leq |x \vee x'| \leq 2k$ and $A_i(x) \leq 3 \cdot 2k + A_i(x') = 6k$. Thus, we can approximate $A_i(x)$ with $p_i(x) = P_{6k,2,2\varepsilon}(A_i(x))$ instead of $P_{3n,2,\varepsilon/2}$ (where both polynomials are from Theorem 10), and this results in the desired bounds on degree and weight. ◀

▶ **Corollary 20.** *Let L be a decision list of length n on $\{0,1\}^n_{\leq k}$. Then L is computed by a PTF p of degree $O(\sqrt{k})$ and weight $n^{O(\sqrt{k})}$.*

Proof. The proof is absolutely the same as in Theorem 13. ◀

We note that the same ideas can be applied to [22, Theorem 6], but it would result in a bound of $\deg(p) = O(\sqrt{k}\log n)$ and $|||p||| = 2^{O(\sqrt{k}\log^2 n)}$, which is much worse if $k \ll n$.

As a side result, we also get an online-learning algorithm for decision lists on $\{0,1\}^n_{\leq k}$.

▶ **Corollary 21.** *There is an algorithm that learns decision lists on $\{0,1\}^n_{\leq k}$ in time $n^{O(\sqrt{k})}$ and with mistake bound $n^{O(\sqrt{k})}$.*

4.2 Lower bound

While the previous proof is straightforward, it requires $d = \Theta(\sqrt{k})$. It turns out that it is not a coincidence: if we want to lower the degree even further, we would need to drastically increase the weight parameter. In fact, Corollary 14 is tight in the case of Hamming Balls as well as in the case of the Boolean Cube.

▶ **Theorem 22.** *Let p be a degree d PTF with integer coefficients which computes ODD-MAX-BIT$_n$ on $\{0,1\}^n_{\leq k}$ and $d = O(\sqrt{k})$. Then $|||p||| = 2^{\Omega(n/d^2)}$.*

The new proof is heavily based on the proof of Theorem 16; the main difference is the way we construct a polynomial that achieves a contradiction on its approximate degree.

Proof. The proof goes as follows. We first partition $[n]$ into blocks of even size about $t = O(d^2)$. Note that we got $r = \Omega(n/d^2)$ blocks in total. Then we prove that for every block i we can find an input y_i such that it may have 1's only in the first i blocks and $|p(y_i)| \geq 2^i$. If we succeed, then we get an input y_r such that $|p(y_r)| \geq 2^r = 2^{\Omega(n/d^2)}$ and, as a corollary, $|||p||| = 2^{\Omega(n/d^2)}$. We will prove this claim by induction.

We adjust the hidden constant in $d = O(\sqrt{k})$ so we can assume $k \geq t$. With that in mind, let $y_0 = 1^t 0^{n-t}$, i.e. the first t bits are 1s and the remaining bits are 0s. Because p has integer coefficients, we have $|p(y_0)| \geq 1$ and we are done with the base case. Now suppose without loss of generality that we have $p(y_i) \geq 2^i$ (the case of negative $p(y_i)$ is completely analogous), we will prove that we can fill the $(i+1)$'s block in y_i and get y_{i+1} such that $p(y_{i+1}) \leq -2^{i+1}$.

Let $P(z)$: $\{0,1\}^{t/2} \to \mathbb{R}$ be constructed by the following constraints on the input of p.

1. Every block up to i's is filled as in y_i.
2. In the $(i+1)$'s block every odd bit becomes a new variable, and every other bit is set to 0.
3. Let $x_1, \ldots, x_t$ be the positive bits in the current input and let $z = (z_1, \ldots, z_{t/2})$ be the new set of variables. We change the value of every x_i to $1 - z_i$.
4. All the remaining bits are also set to 0.

We first prove that for any z the achieved input is indeed in $\{0,1\}^n_{\leq k}$. Note that because we started with t 1s in y_0, every time we set z_i to 1 we change the corresponding x_i to 0. Thus, we never increase the number of 1s in the input, and because $k \geq t$ it is in $\{0,1\}^n_{\leq k}$.

Now suppose by contradiction that the desired y_{i+1} does not exist, i.e. for every x achieved by filling the $(i+1)$'s block of y_i, we have $|p(x)| \leq 2^{i+1}$. By definition, $P(0^{t/2}) = p(y_i) > 2^i$ but for every $z \neq 0^{t/2}$ we have $-2^{i+1} < p(z) < 0$ where the first inequality follows from $|P(x)| \leq 2^{i+1}$ and the second one follows from the definition of the ODD-MAX-BIT$_n$.

We claim that the polynomial $\tilde{P}(z) = -\frac{1}{3 \cdot 2^i} P(z)$ $\frac{1}{3}$-approximates the $OR_{t/2}$ function. Indeed, $\tilde{P}(0) = \frac{1}{3}$ and for every $z \in \{0,1\}^{t/2} \setminus \{0^{t/2}\}$ we have $\frac{2}{3} \leq \tilde{P}(z) \leq \frac{4}{3}$. It is well known (see [26]) that $\deg_{1/3}(OR_n) = \Omega(\sqrt{n})$. For a suitable choice of the hidden constant in t we get that $\deg(\tilde{P}) > \sqrt{t} = d$, but at the same time $\deg(\tilde{P}) = \deg(P) \leq \deg(p) = d$, which contradicts the non-existence of the desired y_{i+1}. ◀

5 Discussion

Despite having optimal bounds on decision lists for almost all $d = O(n^{1/3})$, the situation is less clear for other values of d. As we mentioned in the introduction, Servedio, Tan and Thaler [29, Theorem 6] proved that for any $n^{1/4} \leq d \leq n$ any decision list L: $\{-1,1\}^n \to \{-1,1\}$ on n variables has a degree-d PTF of weight $2^{\tilde{O}_n((n/d)^{2/3})}$. This gives an upper bound for $d = \Omega(n^{1/2})$, but it uses $\{-1,1\}^n$ domain rather than $\{0,1\}^n$. They also proved a lower bound of $2^{\Omega(\sqrt{n/d})}$ for both domains and $d = o\left(\frac{n}{\log^2 n}\right)$, which is stronger than Theorem 16 for $d = \Omega(n^{1/3})$. It uses the same approach, but a different technique for bounding the degree on a block, which suggests that a matching upper bound (if there is one) should have different construction than the one achieved by Corollary 12. We also note that because of the block approach this lower bound can be also adapted for Hamming Balls.

As for the approximation on $\{0,1\}^n_{\leq k}$, an obvious open question is to generalize Theorem 22 for $k = o(d^2)$. Another open question is to find optimal degree-weight tradeoffs for pointwise approximation of Boolean functions on both $\{0,1\}^n$ and $\{0,1\}^n_{\leq k}$, as suggested in [19]. We suspect that by including ε into analysis of Theorem 10 it can be generalized for any $\varepsilon = o(1)$ as well to achieve a similar to Theorem 11 result. After that, the similar ideas used in [6] and [19] can be applied to get degree-weight traidoff results for a class of decision lists.

References

1 Martin Anthony, Yves Crama, and Peter L. Hammer. Decision lists and related classes of boolean functions. In Yves Crama and Peter L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 577–596. Cambridge University Press, 2010. doi:10.1017/cbo9780511780448.017.

2 Richard Beigel. The polynomial method in circuit complexity. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 82–95. IEEE Computer Society Press, 1993.

3 Richard Beigel. Perceptrons, PP, and the polynomial hierarchy. *Comput. Complex.*, 4:339–349, 1994. doi:10.1007/BF01263422.

4 Arnab Bhattacharyya, Suprovat Ghoshal, and Rishi Saket. Hardness of learning noisy halfspaces using polynomial thresholds. In Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet, editors, *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018*, volume 75 of *Proceedings of Machine Learning Research*, pages 876–917. PMLR, 2018. URL: http://proceedings.mlr.press/v75/bhattacharyya18a.html.

5 Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Inf. Process. Lett.*, 42(4):183–185, 1992. doi:10.1016/0020-0190(92)90237-P.

6 Andrej Bogdanov and Christopher Williamson. Approximate bounded indistinguishability. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 53:1–53:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.53.

7 Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions, and spectral norms. *SIAM Journal on Computing*, 21(1):33–42, 1992.

8 Harry Buhrman, Nikolai K. Vereshchagin, and Ronald de Wolf. On computation and communication with small bias. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 24–32. IEEE Computer Society, 2007. doi:10.1109/CCC.2007.18.

9 Mark Bun, Robin Kothari, and Justin Thaler. The polynomial method strikes back: Tight quantum query bounds via dual polynomials. *Theory Comput.*, 16:1–71, 2020. doi:10.4086/toc.2020.v016a010.

10 Mark Bun and Justin Thaler. A nearly optimal lower bound on the approximate degree of AC^0. *SIAM J. Comput.*, 49(4), 2020. doi:10.1137/17M1161737.

11 Mark Bun and Justin Thaler. The large-error approximate degree of AC^0. *Theory Comput.*, 17:1–46, 2021. URL: https://theoryofcomputing.org/articles/v017a007/.

12 Arkadev Chattopadhyay, Meena Mahajan, Nikhil S. Mande, and Nitin Saurabh. Lower bounds for linear decision lists. *Chic. J. Theor. Comput. Sci.*, 2020, 2020. URL: http://cjtcs.cs.uchicago.edu/articles/2020/1/contents.html.

13 Arkadev Chattopadhyay and Nikhil S. Mande. A short list of equalities induces large sign rank. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 47–58. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00014.

14 Ronald de Wolf. A note on quantum algorithms and the minimal degree of ϵ-error polynomials for symmetric functions. *Quantum Inf. Comput.*, 8(10):943–950, 2008. doi:10.26421/QIC8.10-4.

15 Ilias Diakonikolas, Daniel M. Kane, and Alistair Stewart. Learning geometric concepts with nasty noise. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1061–1073. ACM, 2018. doi:10.1145/3188745.3188754.

16 Ilias Diakonikolas, Ryan O'Donnell, Rocco A. Servedio, and Yi Wu. Hardness results for agnostically learning low-degree polynomial threshold functions. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1590–1606. SIAM, 2011. doi:10.1137/1.9781611973082.123.

17 Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2(4):277–300, 1992.

18 Lisa Hellerstein and Rocco A. Servedio. On PAC learning algorithms for rich boolean function classes. *Theor. Comput. Sci.*, 384(1):66–76, 2007. doi:10.1016/j.tcs.2007.05.018.

19 Xuangui Huang and Emanuele Viola. Approximate degree, weight, and indistinguishability. *ACM Trans. Comput. Theory*, 14(1):3:1–3:26, 2022. doi:10.1145/3492338.

20 Adam R. Klivans, Ryan O'Donnell, and Rocco A. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68(4):808–840, 2004.

21 Adam R. Klivans and Rocco A. Servedio. Learning DNF in time $2^{\widetilde{O}(n^{1/3})}$. *Journal of Computer and System Sciences*, 68(2):303–318, 2004.

22 Adam R. Klivans and Rocco A. Servedio. Toward attribute efficient learning of decision lists and parities. *J. Mach. Learn. Res.*, 7:587–602, 2006. URL: http://jmlr.org/papers/v7/klivans06a.html.

23 Matthias Krause and Pavel Pudlák. Computing boolean functions by polynomials and threshold circuits. *Computational Complexity*, 7(4):346–370, 1998.

24 Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, 1987. doi:10.1007/BF00116827.

25 Philip M. Long and Rocco A. Servedio. On the weight of halfspaces over hamming balls. *SIAM J. Discret. Math.*, 28(3):1035–1061, 2014. doi:10.1137/120868402.

26 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Comput. Complex.*, 4:301–313, 1994. doi:10.1007/BF01263419.

27 Ryan O'Donnell and Rocco A. Servedio. New degree bounds for polynomial threshold functions. *Comb.*, 30(3):327–358, 2010. doi:10.1007/s00493-010-2173-3.

28 Micheal E. Saks. Slicing the hypercube. In Keith Walker, editor, *Surveys in Combinatorics*, number 187 in London Mathematical Society Lecture Note Series, pages 211–256. Cambridge University Press, 1993.

29 Rocco A. Servedio, Li-Yang Tan, and Justin Thaler. Attribute-efficient learning and weight-degree tradeoffs for polynomial threshold functions. In Shie Mannor, Nathan Srebro, and Robert C. Williamson, editors, *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, volume 23 of *JMLR Proceedings*, pages 14.1–14.19. JMLR.org, 2012. URL: http://proceedings.mlr.press/v23/servedio12/servedio12.pdf.

30 Alexander A. Sherstov. Communication lower bounds using dual polynomials. *Bulletin of the EATCS*, 95:59–93, 2008.

31 Alexander A. Sherstov. Algorithmic polynomials. *SIAM J. Comput.*, 49(6):1173–1231, 2020. doi:10.1137/19M1278831.

32 Alexander A. Sherstov. The approximate degree of DNF and CNF formulas. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1194–1207. ACM, 2022. doi:10.1145/3519935.3520000.

33 Alexander A. Sherstov and Pei Wu. Near-optimal lower bounds on the threshold degree and sign-rank of AC^0. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 401–412. ACM, 2019. doi:10.1145/3313276.3316408.

34 Kei Uchizawa and Eiji Takimoto. Lower bounds for linear decision trees with bounded weights. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater, and Roger Wattenhofer, editors, *SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings*, volume 8939 of *Lecture Notes in Computer Science*, pages 412–422. Springer, 2015. doi:10.1007/978-3-662-46078-8_34.

A Omitted proofs

Proof of Theorem 13. We can represent L as follows:

$$L(x) = \operatorname{sign} C\left(\sum_{i=1}^{n/h} 3^{n/h-i+1} f_i(x) + b_{n+1}\right).$$

with C from Theorem 10, where each f_i is a sublist responsible for the bits on the interval $x_{(i-1)h+1}, \ldots, x_{ih}$, which returns zero if no literals on this interval is satisfied. It is straightforward to check the correctness of the above decomposition: if f_i contains the first satisfied literal, then it determines the sign of the sum because $3^{n/h-i+1} > \sum_{j<i} 3^{n/h-j+1}$ and $f_j(x) = 0$ for every $j < i$.

Every sublist f_i is a modified decision list, so we can apply Corollary 12 and replace every $Cf_i(x)$ with $p_i(x)$, resulting in

$$H(x) = \sum_{i=1}^{n/h} 3^{n/h-i+1} p_i(x) + Cb_{n+1}.$$

Our goal is to show that $H(x)$ is the desired PTF. First of all, if $x = 0^n$ then every $p_i(x) = 0$ and $\operatorname{sign} H(x) = b_{n+1}$. Otherwise, let $r = (i-1)h + c$ be the first bit such that ℓ_r is satisfied. We have several cases:

1. If $j < i$ then $3^{n/h-i+1} p_j(x) = 0$;
2. $3^{n/h-i+1} p_i(x)$ differs from $3^{n/h-i+1} Cb_r$ by at most $C3^{n/h-i+1}\frac{1}{100}$;
3. The magnitude of each of the remaining values is at most $C3^{n/h-j+1}\left(1 + \frac{1}{100}\right)$.

Combining these bounds, the value of $H(x)$ differs from $3^{n/h-i+1} Cb_r$ by at most

$$C\left(1 + \frac{3^{n/h-i+1}}{100} + \left(1 + \frac{1}{100}\right)\sum_{j=1}^{n/h-i} 3^j\right).$$

The value of the sum is less than $\frac{3^{n/h-i+1}}{2}$, so the overall value is less than $C3^{n/h-i+1}$ and $\operatorname{sign} H(x) = b_r$. The bounds on degree and weight follows from Corollary 12. ◀

Pop & Push: Ordered Tree Iteration in $\mathcal{O}(1)$-Time

Paul Lapey
Department of Computer Science, Williams College, Williamstown, MA, USA

Aaron Williams
Department of Computer Science, Williams College, Williamstown, MA, USA

Abstract

The number of ordered trees (also known as plane trees) with n nodes is the $(n-1)$st Catalan number C_{n-1}. An ordered tree can be stored directly using nodes and pointers, or represented indirectly by a Dyck word. This paper presents a loopless algorithm for generating ordered trees with n nodes using pointer-based representations. In other words, we spend $\mathcal{O}(C_{n-1})$-time to generate all of the trees, and moreover, the delay between consecutive trees is worst-case $\mathcal{O}(1)$-time.

To achieve this run-time, each tree must differ from the previous by a constant amount. In other words, the algorithm must create a type of Gray code order. Our algorithm operates on the children of a node like a stack, by popping the first child off of one node's stack and pushing the result onto another node's stack. We refer to this pop-push operation as a pull, and consecutive trees in our order differ by one or two pulls. There is a simple two-case successor rule that determines the pulls to apply directly from the current tree. When converted to Dyck words, our rule corresponds to a left-shift, and these shift generate a cool-lex variant of lexicographic order.

Our results represent the first pull Gray code for ordered trees, and the first fully published loopless algorithm for ordered trees using pointer representations. More importantly, our algorithm is incredibly simple: A full implementation in C, including initialization and output, uses only three loops and three if-else blocks. Our work also establishes a simultaneous Gray code for Dyck words, ordered trees, and also binary trees, using cool-lex order.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Combinatorics; Mathematics of computing → Combinatorial algorithms

Keywords and phrases combinatorial generation, Gray code, simultaneous Gray code, ordered trees, plane trees, Dyck words, binary trees, Catalan objects, loopless algorithm, cool-lex order

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.53

Supplementary Material *Software (Source Code)*: `https://gitlab.com/combinatronics/ordered-tree-iteration` [12]

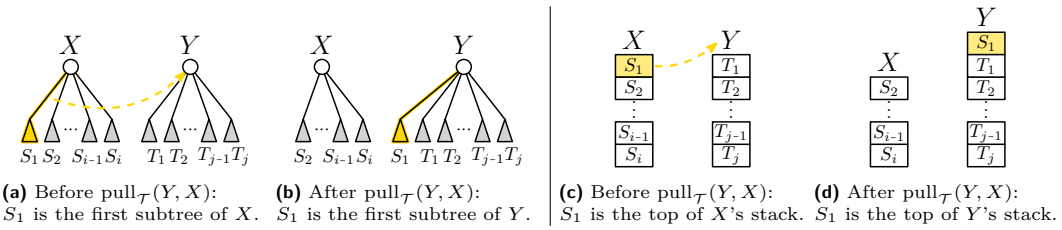
1 Introduction

This article is focused on iterating through every ordered tree with n nodes as quickly and simply as possible. By quickly we mean *loopless*, which is worst-case $\mathcal{O}(1)$-time per tree. This means that we store a single ordered tree in memory, and in constant time we change it into the next tree. In other words, there is always a *constant delay* between successive trees.

To achieve such an algorithm we need to first develop a suitable order. That is, we need to create an ordering in which each tree differs from the previous tree by a constant amount. When measuring the difference between two trees, it is necessary to know how they are represented, or stored in memory. We focus on link-based representations, in which the parent-child relationships are specified using pointers or references.

The way in which we operate on the ordered trees is novel. We treat the children of each node like a stack, and we modify a tree by popping and pushing these stacks. In particular, a *pull* involves popping one stack, and pushing the result onto another stack. In other words, we remove the first subtree of one node, and insert it as the new first subtree of another node. The pull operation is illustrated in Figure 1.

Remarkably, we are able to generate all ordered trees by only pulling subtrees that are paths. In fact, we create each successive tree using one or two of these more restrictive *path-pull* operations. Moreover, the location of the pulled paths is very predictable, which allows us to locate them in constant time. This point is illustrated in Figure 2. A sample listing of all ordered trees with $n = 6$ nodes appears in Figure 6.



(a) Before $\text{pull}_{\mathcal{T}}(Y, X)$: S_1 is the first subtree of X.
(b) After $\text{pull}_{\mathcal{T}}(Y, X)$: S_1 is the first subtree of Y.
(c) Before $\text{pull}_{\mathcal{T}}(Y, X)$: S_1 is the top of X's stack.
(d) After $\text{pull}_{\mathcal{T}}(Y, X)$: S_1 is the top of Y's stack.

Figure 1 The pull operation relocates first subtrees in an ordered tree $\mathcal{T}$. More specifically, if X and Y are nodes of $\mathcal{T}$, then $\text{pull}_{\mathcal{T}}(Y, X)$ removes the first subtree of X and adds it as a new first subtree of Y; see (a)–(b). In other words, Y pulls X's first subtree so that it becomes its own first subtree. The operation is particularly natural when each node stores its subtrees using a stack, since the pull simply "pops" X's stack, and "pushes" the result onto Y's stack; see (c)–(d).

One of the most notable aspects of our algorithm is its simplicity. Indeed, our C implementation uses only three loops and three if-else blocks, half of which are used for initialization and output. Our main result is summarized by the following theorem. Figure 7 provides the main loop of Algorithm O in both pseudocode and C.

▶ **Theorem 1.** *Algorithm O iterates through the ordered trees with n nodes looplessly using only one or two path-pulls between successive trees.*

We refer to the order in which the trees are generated as a *Gray code* – in reference to the well-known binary reflected Gray code [8] – meaning that the objects are ordered so that the next object always differs from the previous object in a constant amount according to some simple metric. More descriptively, our order is a *2-pull Gray code*, since two pulls are used in the worst-case [1] . Theorem 1 represents the first 2-pull Gray code for ordered trees.

[1] The pulled subtrees are always paths, so we could also describe our result as a *2-path-pull Gray code*.

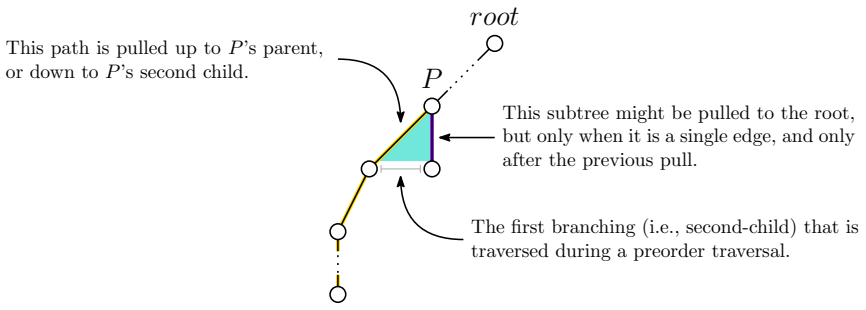


Figure 2 Our Gray code for ordered trees is based on pulls. More specifically, the next ordered tree is always created by one or two pulls. Furthermore, the pulled subtrees are always paths, and they are always located on either side of the tree's first branching in a preorder traversal (in turquoise) as shown in (b). A successor rule with two cases provides the details in (6) and Figure 5. Figure 7 has loopless (i.e., worst-case $\mathcal{O}(1)$-time per tree) algorithm implementations in pseudocode and C, with full C implementations (including command-line argument processing) in the Appendices.

Algorithm O is also the first loopless algorithm for generating ordered trees with a linked-based representation in which all of the details are provided. The secret to our algorithm is ordering Dyck words using a variant of co-lexicographic order known as cool-lex order.

1.1 Relationship to Previous Results

Many previous papers have focused on efficiently generating ordered trees, and other closely related objects, using various representations. For broad overviews of this research area, we recommend Knuth's coverage of generating combinatorial objects in Volume 4A of *The Art of Computer Programming* [10], and Mütze's recent update [16] of Savage's well-known survey [27]. Many practical programs for generating combinatorial objects can also be found in Arndt [2], and on the *Combinatorial Object Server* [17]. Readers are also directed to a new preprint by Nakano [18] that provides a nice alternative Gray code for ordered trees: successive trees are obtained by removing a leaf and appending it to another node.

1.1.1 Catalan Objects

Ordered trees are a *Catalan object* since they are enumerated by the Catalan numbers. More specifically, the number of ordered trees with n nodes is the $(n-1)$st Catalan number C_{n-1}. Other well-known Catalan objects include Dyck words and binary trees. A *Dyck word of order* n is a binary string with n copies of 1 and n copies of 0, where each prefix has at least as many 1s as 0s. These strings have a trivial correspondence with balanced parentheses of length $2n$: map 1s to (s and 0s to)s. These sets of strings are shown in (1) for $n = 3$.

$$\{101010, 101100, 110010, 110100, 111000\} \qquad \{()()(), ()(()), (())(), (()()), ((()))\} \tag{1}$$

Dyck words of order n are counted by the nth Catalan number C_n, as are binary trees with n nodes. Many previous results have focused on ordering and generating these objects using different operations, with a very well-known example being binary trees by rotations [14].

1.1.2 Loopless vs Constant Amortized Time

Skarbek [30] provided the first algorithm for generating link-based representations of ordered trees in *constant amortized time (CAT)*, meaning that the delay is $\mathcal{O}(1)$-time in an amortized sense, rather than in the worst-case as in a loopless algorithm. Korsh and Lafolette [11] crafted a loopless algorithm for generating ordered trees with a fixed branching sequence. In other words, they generated subsets of ordered trees in which the number of children at each node form the same multiset. Their algorithm used a string-based representation, and answered an open problem posed by Roelants van Baronaigien [34]. They also stated, in one sentence, that their results could be adapted to link-based representations. By "layering" the output of that proposed algorithm, it may be possible to create a loopless link-based algorithm for generating all ordered trees with n nodes.

Theoretically speaking, loopless algorithms are more significant achievements than CAT algorithms. In particular, standard lexicographic orders can often be generated in CAT [21], and this has been accomplished for ordered trees on multiple occasions [7, 19]. However, loopless algorithms are often less practical than their CAT counterparts for several reasons.

- Efficiency. Loopless algorithms typically involve more instructions than CAT algorithms, and their overall run-times are often longer.
- Ease of implementation. Generation algorithms are often ported from one programming language (or pseudocode) to another, so longer implementations are again less desirable.

- Novelty. Most loopless algorithms involve the creation of a novel ordering of the underlying objects, which can hinder debugging and efficient ranking/unranking (see Section 1.1.3).

For example, [11] creates a novel order, and its provided C code includes over 100 instances of the `if` and `else` keywords. In contrast, we generate a familiar variant of lexicographic order and our C implementation requires only 4 instances of `if` and `else`; see [12] or the Appendix.

1.1.3 Simultaneous Gray Codes

Suppose that X and Y are sets of combinatorial objects that are in one-to-one correspondence according to bijection $f : X \to Y$. If $x_1, x_2, \ldots, x_m$ is a Gray code for X, then we may ask whether the corresponding order for the objects in Y, namely $f(x_1), f(x_2), \ldots, f(x_m)$, is also a Gray code. If this is true, then we refer to the order as a *simultaneous Gray code* for X and Y according to f. As a trivial example, let X_n be the set of n-bit binary strings, and Y_n be the set of subsets of $[n] = \{1, 2, \ldots, n\}$, with $f : X_n \to Y_n$ mapping each incidence vector $b_1 b_2 \cdots b_n$ to its subset. In this case, the binary reflected Gray code provides an ordering of X_n in which successive strings differ in a single bit, and the corresponding subsets in Y_n differ by adding or removing one element, as seen for $n = 3$ in (2) and (3), respectively.

000	001	011	010	110	111	101	100	(2)
$\emptyset$	$\{3\}$	$\{2,3\}$	$\{2\}$	$\{1,2\}$	$\{1,2,3\}$	$\{1,3\}$	$\{1\}$	(3)

The literature features relatively few non-trivial simultaneous Gray codes[2]. This scarcity is in part due to the difficulty in finding such orders. To illustrate this important point, let us consider the goal of constructing a simultaneous Gray code for Dyck words and ordered trees. During this discussion, we'll use the standard mapping of an ordered tree $\mathcal{T}$ with $n-1$ nodes to a Dyck word of order n found in Stanley's *Catalan Addendum* [31]:

> *Conduct a preorder traversal of the ordered tree $\mathcal{T}$, and record* 1 *when going down an edge, and* 0 *when going up an edge.*

Figure 3a illustrates this mapping, where Dyck($\mathcal{T}$) is the Dyck word corresponding to $\mathcal{T}$.

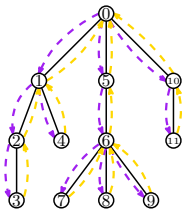
One of the simplest operations on strings, including Dyck words, is an *adjacent-transposition*, which replaces a substring xy with yx. An adjacent-transposition is also known as a *swap*. Figure 3b illustrates how a swap in a Dyck word affects the corresponding ordered tree.

To understand Figure 3b, first note that a 10 substring in a Dyck word corresponds to a leaf in an ordered tree. Swapping 10 to 01 causes the leaf (shown in gold) to become the next sibling of its parent (purple) with its rightward siblings (gold) becoming its children. As a result, this small change in a Dyck word does not correspond to a constant amount of work in the ordered trees, so long as the nodes have `parent` pointers. Hence, a swap Gray code for Dyck words won't provide a simultaneous Gray codes for ordered trees, unless (a) a non-standard bijection is used, (b) `parent` pointers are omitted, or (c) the swaps used can be further constrained to change fewer `parent` pointers. There is another complication with this proposed approach: Swap Gray codes don't always exist for Dyck words[3] [20, 22].

---

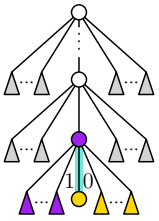
[2] A recent exception involves Baxter permutations and mosaic floorplans that arise in VLSI design [38]. A Gray code for Baxter permutations was found by generalizing the greedy interpretation of plain change order [36], and it provides a simultaneous Gray code for mosaic floorplans according to the bijection in [1]. More broadly, Algorithm J [9] generates a jump Gray code for various types of pattern-avoiding permutations and simultaneously orders various types of rectangulations by flips and wall-slides [15].

[3] A 2-swap Gray code of Dyck words does exist by Vajnovszki and Walsh [33]. Their order uses swaps

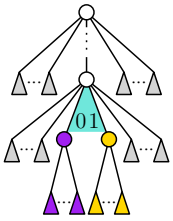


Dyck($\mathcal{T}$) = 11100100111010100011100.

(a) During a preorder traversal of $\mathcal{T}$, record 1 when going down an edge (purple) and 0 when going up an edge (gold) to obtain its corresponding Dyck word.



Before swapping 10.



After swapped to 01.

(b) A swap in a Dyck word can change an arbitrary number of `parent` pointers. Specifically, the root of each gold subtree has a new parent.

Figure 3 With the standard bijection between Dyck words and ordered trees in (a), a swap in a Dyck word can cause a non-constant amount of change in the corresponding ordered tree as shown in (b). This illustrates the difficulty in constructing simultaneous Gray codes.

Cool-lex order is a variation of co-lexicographic order that was first introduced for (s, t)-combinations by Ruskey and Williams [25] in the latter's PhD thesis [37]. Since then it has been used to create Gray codes and efficient algorithms for a variety of combinatorial objects [35, 23, 28, 24, 5, 3], with recent examples including multiset necklaces [29] and fixed-content Łukasiewicz words [13]. In particular, the application of cool-lex order to Dyck words [25] is relevant later in the article. Besides its familiarity, cool-lex order also provides efficient *ranking* and *unranking* of Dyck words [25] (and k-ary [6] and $\frac{1}{k}$-ary Dyck words [5]), which means that the position of any word can be determined, as can the word in any position, using only $\mathcal{O}(n)$ arithmetic operations. Hence, our Gray code of ordered trees can be ranked and unranked just as efficiently, since the standard bijection runs in linear-time.

1.2 Outline

Section 2 discusses ordered trees in more detail. Section 3 provides our Gray code, and Section 4 generates it efficiently. Correctness is proven in Section 5 and Section 6 has final remarks. C implementations with or without `parent` pointers are in the appendices and [12].

2 Ordered Trees

In this section, we provide background information on ordered trees, starting with terminology in Section 2.1 and link-based representations in Section 2.2. Then in Section 2.3 we introduce the stack-based pull operation, and a further specialization in Section 2.4.

2.1 Terminology and Conventions

Throughout the article, we typeset ordered trees as $\mathcal{T}$, and we use n to denote the number of nodes in a tree. We typeset nodes of a tree as X, and we frequently refer to a node and the subtree rooted at that node interchangeably. Code is written with a `monospaced` font.

and *two-close transpositions*, which replace a substring 100 with 001 (or vice versa) and which can be realized using two swaps. This would be a good candidate for simultaneous Gray code, so long as parent pointers could be avoided. Its loopless pseudocode is relatively compact, containing only 17 `if` and `else` keywords, but this is still significantly more complicated than our result.

When discussing ordered tree $\mathcal{T}$, we use the term *first branching*, and this term could be misinterpreted. We intend it to mean the following: The first branching occurs when the preorder traversal goes up an edge, and then down an edge, for the first time. The node that is incident to both of these edges is the parent of the first branching. For example, these first branches are highlighted in turquoise in Figure 6. In particular, note that the subtree to the left of the first branch must be a path (see Lemma 2).

Another way of describing the first branching is as follows. Let O be the first second-child node that is visited during a preorder traversal. If P is the parent of O, then P is the parent of the first branching.

2.2 Link-Based Representations

We focus on *link-based* representations of ordered trees, meaning that children are linked by pointers or references. There are at least two natural variations, as discussed below.

- Each node has a first child reference and a right sibling reference. The children of a given node `o` form a linked list `[o.first, o.first.right, o.first.right.right...]`. In particular, a leaf `o` has a `o.first == null`, while a rightmost child of a node has `o.right == null`.
- Each node has a child array and a number of children. In particular, a leaf `o` has `o.num == 0` and `o.child[0] == null` (using 0-based indexing).

We refer to the first as a *linked list representation* and the second as a *link array* representation.

Throughout this paper, we treat the children of a node as a stack. Since stacks can be implemented with constant-time operations using a linked list or an array, this allows us to ignore the specific type of link-based representation. However, our focus is on the linked list representation.

2.3 Stack Operations: Pop, Push, and Pull

Given an ordered tree $\mathcal{T}$ and a non-leaf node A, let $\text{pop}_{\mathcal{T}}(A)$ remove the first subtree of A and return it. In other words, if the child-list of A is viewed as a stack, then this operation pops the stack. In particular, if A has only one child in $\mathcal{T}$, then A will be a leaf in $\text{pop}_{\mathcal{T}}(A)$.

If $\mathcal{S}$ is a non-empty ordered tree, then let $\text{push}_{\mathcal{T}}(A, \mathcal{S})$ be the result of adding $\mathcal{S}$ as a new first subtree to node A. In other words, if the child-list of A is viewed as a stack, then this operation pushes a new subtree $\mathcal{S}$ onto the stack. In particular, if A is a leaf in $\mathcal{T}$, then A will have one child in $\text{push}_{\mathcal{T}}(A, \mathcal{S})$.

We combine these two operations by popping a node, and then pushing the removed subtree into the tree that was created by the pop. Note that the tree that results from this combined operation will have the same number of nodes as the original tree. Furthermore, the resulting ordered tree differs from the original ordered tree, if and only if, the nodes that are popped and pushed are not the same.

We refer to the combined pop-push operation as a *pull*, and we define it as follows

$$\text{pull}_{\mathcal{T}}(A, B) \text{ is } \text{push}_{\mathcal{T}'}(A, \text{pop}_{\mathcal{T}}(B)) \text{ where } \mathcal{T}' \text{ is the modification of } \mathcal{T} \text{ after popping.} \quad (4)$$

We think of $\text{pull}_{\mathcal{T}}(A, B)$ as node A pulling the first subtree off of B and adding it as its own first child. For this reason, we'll illustrate the operation with an arrow from B to A that shows how the subtree is pulled toward A.

Finally, we also use a *double pull*, which pulls from a given node twice in a row. In other words, the first subtree is pulled from the node, and then the new first subtree is pulled from the node. We define the operation and its notation as follows.

$$\text{pull}_{\mathcal{T}}(A; C, B) \text{ is } \text{pull}_{\mathcal{T}}(A, B) \text{ then } \text{pull}_{\mathcal{T}'}(C, B), \text{ where } \mathcal{T}' \text{ is created by the first pull.} \quad (5)$$

▷ A pulls B's first child
function PULL(A, B)
    $temp \leftarrow B.first$
    $B.first \leftarrow temp.right$
    $temp.right \leftarrow A.first$
    $A.first \leftarrow temp$
    $temp.parent \leftarrow A$

```
// A pulls B's first child.
node *pull(node *A, node *B){
  node *temp = B.first;
  B.first = temp.right;
  temp.right = A.first;
  A.first = temp;
  temp.parent = A;
}
```

Figure 4 The `pull` operation can be implemented as a `pop` followed by a `push`, namely, `pull(A,B)` is `push(A, pop(B))`. It can also be implemented directly, as shown above in pseudocode and C. The pulled node must not be a leaf.

In other words, A and C both pull B: B's first subtree becomes A's new first subtree, then B's new first subtree becomes C's new first subtree. When illustrating the double pull in Figure 5, we use gold for the first pull (from G), and purple for the second (from *root*).

2.4 Path-Pulls

We refer to a pull operation as a *path-pull* if the pulled subtree is a path. All of the pull operations used in our Gray code are path-pulls. This distinction does not change the efficiency of the operation, but it will become relevant in Section 5. The following lemma states that pulling on the left side of the first branching is always a path-pull.

▶ **Lemma 2.** *If $\mathcal{T}$ is an ordered tree, and P be the parent of its first branching, then the operation* $\text{pull}_{\mathcal{T}}(X, P)$ *is a path-pull for any node $X \neq P$.*

Proof. Since P is the parent of the first branching, all of the nodes in the first subtree of P must have only one child. Hence, the first subtree of P is a path. ◀

3 Gray Code order using Pulls

This section defines the order in which we generate ordered trees with n nodes. The order is built one tree at a time by a successor rule whose origin is based on Theorem 5 in Section 5.

3.1 First and Last Trees

Our order starts and ends with two path-like trees.

- The first tree $\mathcal{T}_1$ has a root with two children, and the subtree rooted at the second-child is a path of length $n-2$.
- The last tree $\mathcal{T}_0$ is the unique path on n nodes.

For examples of these trees, refer to the beginning and end of Figure 6.

The rest of the order is generated by a successor rule, which transform any ordered tree – other than $\mathcal{T}_0$ – into the next ordered tree.

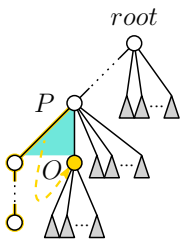
3.2 Successor Rule

Given an ordered tree $\mathcal{T}$ that is not a path (i.e., $\mathcal{T} \neq \mathcal{T}_0$), we distinguish several nodes that will be referenced by the successor rule. Let O be the first second-child that is visited during a preorder traversal of $\mathcal{T}$, and P be its parent, and G be its grandparent. In other words, P is the parent of the first branching that is traversed during a preorder traversal of $\mathcal{T}$,

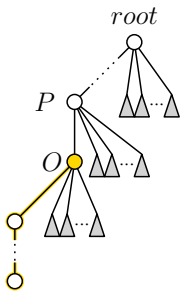
meaning that the traversal goes up into P and back down into its second-child O. Note that P and O are well-defined so long as $\mathcal{T} \neq \mathcal{T}_0$, while G is only well-defined when the parent P is not the *root* of the tree $\mathcal{T}$. Given these nodes, the successor rule can now be stated.

$$\text{next}(\mathcal{T}) = \begin{cases} \text{pull}_{\mathcal{T}}(O, P) & \text{if } P = root \text{ or } O \text{ has a child} \quad (6c) \\ \text{pull}_{\mathcal{T}}(G; root, P) & \text{otherwise (i.e. } P \neq root \text{ and } O \text{ is a leaf)} \quad (6d) \end{cases}$$

Figure 5 illustrates the pulls used in (6), while Figure 6 illustrates the full order for $n = 6$.

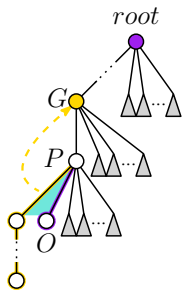


Current tree $\mathcal{T}$ where $P = root$ or O is not a leaf.

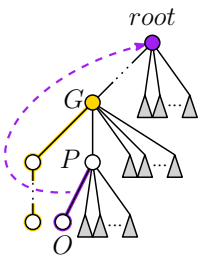


New tree $\mathcal{T}'$ is obtained by $\text{pull}_{\mathcal{T}}(O, P)$.

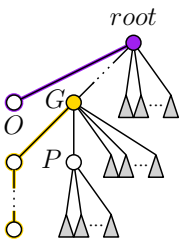
(a) Case 6c. After the pull, the new value of node O is updated to be the new second-child of O (if non-null) or the new second-child of P (if non-null), or *null*. The pull is equivalent to $\text{push}(O, \text{pop}(P))$.



Current tree $\mathcal{T}$ where $P \neq root$ and O is a leaf.



Intermediate tree $\mathcal{I}$ is obtained by $\text{pull}_{\mathcal{T}}(G, P)$.



New tree $\mathcal{T}'$ is obtained by $\text{pull}_{\mathcal{I}}(root, P)$.

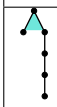
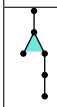
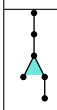
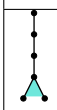
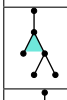
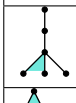
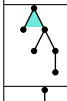
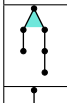
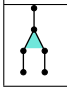
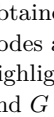
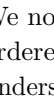
(b) Case 6d. After the pull, the new value of node O is updated to be the new second-child of *root*. The pull operations are equivalent to $\text{push}(G, \text{pop}(P))$ followed by $\text{push}(root, \text{pop}(P))$.

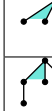
Figure 5 Our pull Gray code is generated by the two case successor rule in (6), which transforms the current ordered tree $\mathcal{T}$ into the new next ordered tree $\mathcal{T}'$ using one or two pulls. In these figures, O is the first node in a preorder traversal that is not on the path from the root to the leftmost descendent, and P is its parent, and G is its grandparent (if applicable). White circles denote non-null nodes, and grey triangles denote an unspecified number of children and subtrees. The pull operations, which are always path-pulls, are highlighted. The first pulling node and pulled path are in gold, while the second are in purple. The captions also explain how to update the value of O.

4 Loopless implementation

In this section, we show how to generate the order proposed in Section 3.2 by a loopless algorithm that we refer to as *Algorithm O*.

Algorithm O first creates an initial tree $\mathcal{T}_1$. Then it repeatedly applies the successor rule in (6). To apply the rule, the algorithm tracks node O, with Figure 5 illustrating how its location changes after each application of (6). Given O's location, the pointer updates can be easily inferred from Figure 5. The rule is applied, and the next tree is visited, until the final tree $\mathcal{T}_0$ is created. This termination condition occurs exactly when O is set to null.

Tree	Dyck word	Next
	10111110000	(6c)
	11011110000	(6c)
	11101110000	(6c)
	11110100000	(6d)
	10111010000	(6c)
	11011010000	(6c)
	11101010000	(6d)
	10110110000	(6c)
	11011100000	(6c)
	11101100000	(6d)
	10111001000	(6c)
	11011001000	(6d)
	10110101000	(6c)
	11010101000	(6d)
	10101101000	(6c)
	11001101000	(6c)
	11100101000	(6d)
	10111000100	(6c)
	11011000100	(6d)
	10101000100	(6c)
	11001000100	(6d)
	11100000100	(6d)

Tree	Dyck word	Next
	1111001000	(6d)
	1011100100	(6c)
	1101100100	(6c)
	1110100100	(6d)
	1011010100	(6c)
	1101010100	(6d)
	1010110100	(6c)
	1100110100	(6c)
	1110010100	(6d)
	1011001100	(6c)
	1101001100	(6d)
	1010101100	(6c)
	1100101100	(6c)
	1110010100	(6c)
	1111000100	(6d)

Tree	Dyck word	Next
	1011100010	(6c)
	1101100010	(6c)
	1110100010	(6d)
	1011010010	(6c)
	1101010010	(6d)
	1010110010	(6c)
	1100110010	(6c)
	1110010010	(6d)
	1011001010	(6c)
	1101001010	(6d)
	1010101010	(6c)
	1100101010	(6c)
	1110001010	(6c)
	1111000010	(6c)
	1111100000	

Figure 6 The ordered trees with $n = 6$ nodes as generated by our algorithm. The next tree is obtained by pull(O, P) when (6c) is specified, or by pull$(G; root, P)$ when (6d) is specified. These nodes are situated around the first branching that is traversed during a preorder traversal, which is highlighted in turquoise. More specifically, O is the second-child of this branching, P is its parent, and G is its grandparent. The last tree is the unique tree which has no such branching. Each tree is displayed beside its corresponding Dyck word. Each Dyck word can be transformed into its successor by left-shifting the highlighted bit into the second position, as discussed in Theorem 5.

Figure 7 gives the algorithm in pseudocode and C. A complete C implementation (with arguments parsing, initialization, and visit routines) can be found in the appendix or [12].

5 Connection to Cool-lex Order for Dyck Words

We now prove that the order from Section 3 is correct, in the sense that it generates every ordered tree with n nodes, starting from $\mathcal{T}_1$ and ending at the path $\mathcal{T}_0$. Our approach is to understand the successor rule from Section 3.2 in terms of Dyck words. We start by showing

```
function COOL-ORDERED-TREES(n)
    ▷ Generate initial tree
    O ← root.first.right
    visit(root)
    while O ≠ NULL do
        P ← O.parent
        if O.first ≠ NULL then
            pull(O, P)
            O ← O.first.right
        else
            if O.parent == root then
                pull(O, P)
            else
                pull(P.parent, P)
                pull(root, P)
            O ← O.right
        visit(root)
```

(a) Generate all ordered trees with $n+1$ nodes.

```
void coolOtree(int n){
  node* root = get_initial_tree(n);
  node* o=root->first->right;
  visit(root);
  while(o){
    p=o->parent;
    if (o->first) { // if o has a child
      pull(o,p);
      o = o->first->right;
    } else {
      if (p == root) { // if o's parent
                       // is root
        pull(o,p);
      } else {
        pull(p->parent,p);
        pull(root,p);
      }
      o = o->right;
    }
    visit(root);
  }
}
```

(b) C implementation of `coolOtree`.

Figure 7 Algorithm O is presented in pseudocode in (a), and its main function in C is in (b).

how certain pull operations in ordered trees translate to changes in the associated Dyck words in Section 5.1. In Section 5.2, we recall the successor rule for generating Dyck words in cool-lex order from [25]. Finally, in Section 5.3, we prove that the successor rule for ordered trees proceeds in the same way as the cool-lex rule for Dyck words. In other words, our successor rule for ordered trees is guaranteed to generate all C_{n-1} ordered trees because the corresponding Dyck words are generated in cool-lex order.

5.1 Pulls in Ordered Trees and Shifts in Dyck Words

Now we attempt to understand how the path-pulls in (6) translate to changes in the associated Dyck words. We'll find that the changes are *shifts*, which moves one bit elsewhere in the Dyck word. In particular, a *left-shift* moves a bit to the left, and a *right-shift* moves a bit to the right. We present two lemmas that mirror the cases in (6) and their depictions in Figure 8. Our first lemma is associated with the single pull used in (6c).

▶ **Lemma 3.** *Let $\mathcal{T}$ be an ordered tree that is not a path and let O be the first node in a preorder traversal that is not a first-child, and P be its parent. Let $\mathcal{T}'$ be the result of* $\text{pull}_{\mathcal{T}}(O, P)$. *Then there exists $p \geq q \geq 1$ and an $\alpha \in \{0,1\}^*$ in which*

$$\text{Dyck}(\mathcal{T}) = 1^p 0^q 1\alpha \text{ and } \text{Dyck}(\mathcal{T}') = 11^p 0^q \alpha. \tag{7}$$

In other words, the 1 at index $p+q+1$ is shifted to index 1.

Proof. From Figure 8a, we can see that $\text{pull}_{\mathcal{T}}(O, P)$ transforms $\mathcal{T}$ into $\mathcal{T}'$ with

$$\text{Dyck}(\mathcal{T}) = 1^a 11^b 0^b 01\alpha \text{ and } \text{Dyck}(\mathcal{T}') = 1^a 111^b 0^b 0\alpha.$$

Therefore, the result follows by setting $p = a+b+1$ and $q = b+1$. This is because $\text{Dyck}(\mathcal{T}) = 1^a 11^b 0^b 01\alpha = 1^p 0^q 1\alpha$ and $\text{Dyck}(\mathcal{T}') = 1^a 111^b 0^b 0\alpha = 11^p 0^q \alpha$. ◀

The second lemma considers the double pull used in (6d).

▶ **Lemma 4.** *Suppose that* (6d) *transforms* $\mathcal{T}$ *into* $\mathcal{T}'$*. That is,* $\mathcal{T}$ *is an ordered tree that is not a path, where* O *is a non-leaf node that is the first node in a preorder traversal that is not a first-child,* $P \neq root$ *is its parent,* G *is its grandparent,* $\mathcal{I}$ *is the result of* $\text{pull}_{\mathcal{T}}(G, P)$*, and* $\mathcal{T}'$ *is the result of* $\text{pull}_{\mathcal{I}}(root, P)$*. Then there exists* $p < q$ *and an* $\alpha \in \{0,1\}^*$ *in which*

$$\text{Dyck}(\mathcal{T}) = 1^p0^q10\alpha \text{ and } \text{Dyck}(\mathcal{T}') = 101^{p-1}0^q1\alpha. \tag{8}$$

In other words, the 0 *at index* $p+q+2$ *is shifted to index* 2*.*

Proof. From Figure 8b, we can see that $\text{pull}_{\mathcal{T}}(G, P)$ transforms $\mathcal{T}$ into $\mathcal{I}$ with

$$\text{Dyck}(\mathcal{T}) = 1^a111^b0^b010\alpha \text{ and } \text{Dyck}(\mathcal{I}) = 1^a11^b0^b0110\alpha.$$

Then Figure 8c, shows that $\text{pull}_{\mathcal{I}}(root, P)$ transforms $\mathcal{I}$ into $\mathcal{T}'$ with

$$\text{Dyck}(\mathcal{I}) = 1^a11^b0^b0110\alpha \text{ and } \text{Dyck}(\mathcal{T}') = 101^a11^b0^b01\alpha.$$

Therefore, the result follows by setting $p = a + b + 2$ and $q = b + 1$. This is because $\text{Dyck}(\mathcal{T}) = 1^a111^b0^b010\alpha = 1^p0^q10\alpha$ and $\text{Dyck}(\mathcal{T}') = 101^a11^b0^b01\alpha = 101^{p-1}0^q1\alpha$. ◀

5.2 Cool-lex Order for Dyck Words

Dyck words of order n can be generated by a simple successor rule [25]. The resulting order can be understood as a sublist of the cool-lex order of (n, n)-combinations [26] or binary strings of length $2n$ [32], and as a special case of the order for bubble languages [23] or fixed-content Łukasiewicz words [13] both of which can be generated efficiently [28, 13]. However, these string results have no immediate implications for generating ordered trees.

▶ **Theorem 5** ([25])**.** *Dyck words of order* n *are generated in cool-lex order starting from* $101^{n-1}0^{n-1}$ *by the following successor rule, where* $\alpha \in \{0,1\}^*$ *denotes an unchanged suffix.*

(a) *If the current string equals* $1^p0^q11\alpha$ *or* $1^p0^p10\alpha$*, then the next string is* $11^p0^q1\alpha$ *or* $11^p0^p0\alpha$*, respectively. In other words, the* 1 *at index* $p+q+1$ *is shifted to index* 1*.*

(b) *If the current string equals* $1^p0^q10\alpha$ *with* $p > q$*, then the next string is* $101^{p-1}0^q1\alpha$*. In other words, the* 0 *at index* $p+q+2$ *is shifted to index* 2*.*

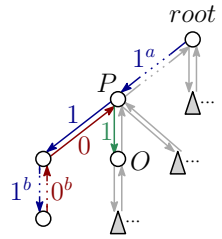
The only string that is not covered by one of these cases is the last Dyck word, 1^n0^n*.*

Notice that in each case of Theorem 5, the next string is obtained by a left-shift. In other words, a single symbol is deleted, and then reinserted somewhere to the left

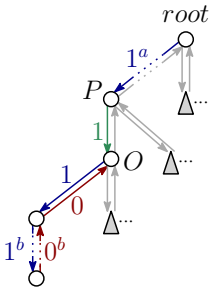
5.3 Proof of Correctness

Proof of Theorem 1. It is clear that Algorithm O in Figure 7 is loopless and correctly implements the order defined in Section 3. What remains to be proven is that the algorithm actually generates all of the ordered trees with n nodes. The proof equates the successor (6) for ordered trees into the cool-lex rule for Dyck words of order $n-1$.

Observe that the first ordered tree $\mathcal{T}_1$ has $\text{Dyck}(\mathcal{T}_1) = 101^{n-2}0^{n-2}$, which matches the first string in the cool-lex order of Dyck words. Similarly, the last ordered tree $\mathcal{T}_0$ has $\text{Dyck}(\mathcal{T}_0) = 1^{n-1}0^{n-1}$, which matches the last string. It remains to be proven that the successor rule for ordered trees in (6) matches the cool-lex successor rule for Dyck words. Fortunately, this follows immediately from Lemmas 3–4 and Theorem 5. ◀

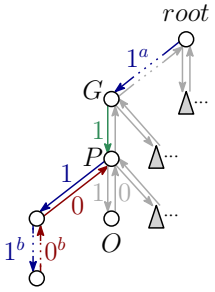


Ordered tree $\mathcal{T}$ with Dyck($\mathcal{T}$) = $1^a11^b0^b01\alpha$.

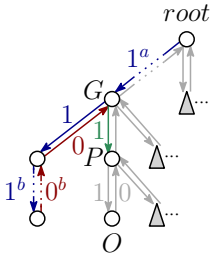


Ordered tree $\mathcal{T}'$ with Dyck($\mathcal{T}'$) = $1^a111^b0^b0\alpha$.

(a) The Dyck word is modified by shifting a single 1 to the left during the operation $\text{pull}_{\mathcal{T}}(O, P)$.

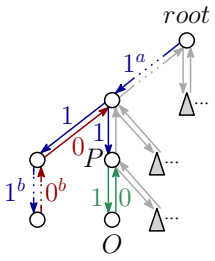


Ordered tree $\mathcal{T}$ with Dyck($\mathcal{T}$) = $1^a111^b0^b0\alpha$.

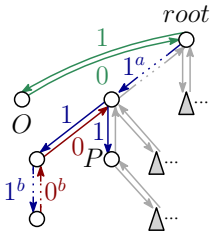


Ordered tree $\mathcal{I}$ with Dyck($\mathcal{I}$) = $1^a11^b0^b01\alpha$.

(b) The Dyck word is modified by shifting a single 1 to the right during the operation $\text{pull}_{\mathcal{T}}(G, P)$.



Ordered tree $\mathcal{I}$ with Dyck($\mathcal{I}$) = $1^a11^b0^b0110\alpha$.



Ordered tree $\mathcal{T}'$ = $101^a11^b0^b01\alpha$.

(c) The Dyck word is modified by shifting 10 to the left during the operation $\text{pull}_{\mathcal{I}}(root, P)$.

Figure 8 Illustrating how several path-pull operations change the corresponding Dyck words. Lemmas 3–4 are illustrated in (a)–(c), with the first mirroring the pull in (6c) and Figure 5a, and the latter two mirroring the double pulls in (6d) and Figure 5b, respectively. Blue and red arrows indicate 1 and 0 bits during a preorder traversal, respectively, with green reserved for changes, and gray for suffixes that do not change.

6 Final Remarks

In this article, we provided a 2-pull Gray code for ordered trees and a loopless algorithm for generating it. Our work also completes a simultaneous Gray code for Dyck words, ordered trees, and binary trees [25], which are three of the foremost Catalan structures.

6.1 Additional Results

Our presentation of pseudocode and C code is focused on the linked list implementation of the link-based representation of an ordered tree, however, the same results hold when using link arrays. This is because the children of a node can be stored in an array of sufficient size, and each pop and push can be performed at the end of the array.

The cool-lex order of Dyck words was also used to create a loopless algorithm for binary trees in [25]. By storing an additional pointer to a node's left sibling, it is possible to augment our algorithms so that they simultaneously output a Gray code of ordered trees and binary trees. We are looking forward to publishing this in an extended version of the paper.

6.2 Open Problems

We showed that two pulls are sufficient for listing ordered trees, even when the pulled subtrees are always paths. This raises two follow-up questions.

- Is there a 1-pull Gray code for ordered trees?
- Is there a 1-path-pull Gray code for ordered trees?

Ideally, a positive answer would also be accompanied by an efficient algorithm.

Finally, we mention that generalizations of Catalan objects were recently ordered and generated by Cardinal, Merino and Mütze [4] using Algorithm J. This opens the door to considering further generalizations of the results found in this article.

References

1 Eyal Ackerman, Gill Barequet, and Ron Y Pinter. A bijection between permutations and floorplans, and its applications. *Discrete Applied Mathematics*, 154(12):1674–1684, 2006.

2 Jörg Arndt. *Matters Computational: ideas, algorithms, source code*. Springer Science & Business Media, 2010.

3 Péter Burcsi, Gabriele Fici, Zsuzsanna Lipták, Frank Ruskey, and Joe Sawada. On combinatorial generation of prefix normal words. In *Symposium on Combinatorial Pattern Matching*, pages 60–69. Springer, 2014.

4 Jean Cardinal, Arturo Merino, and Torsten Mütze. Efficient generation of elimination trees and graph associahedra. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2128–2140. SIAM, 2022.

5 Stephane Durocher, Pak Ching Li, Debajyoti Mondal, Frank Ruskey, and Aaron Williams. Cool-lex order and k-ary Catalan structures. *Journal of Discrete Algorithms*, 16:287–307, 2012.

6 Stephane Durocher, Pak Ching Li, Debajyoti Mondal, and Aaron Williams. Ranking and loopless generation of k-ary Dyck words in cool-lex order. In *International Workshop on Combinatorial Algorithms*, pages 182–194. Springer, 2011.

7 MC Er. Enumerating ordered trees lexicographically. *The Computer Journal*, 28(5):538–542, 1985.

8 Frank Gray. Pulse code communication. *United States Patent Number 2632058*, 1953.

9 Elizabeth Hartung, Hung Hoang, Torsten Mütze, and Aaron Williams. Combinatorial generation via permutation languages. i. fundamentals. *Transactions of the American Mathematical Society*, 375(04):2255–2291, 2022.

10 Donald E Knuth. *Art of Computer Programming, Volume 4, Fascicle 4, The: Generating All Trees–History of Combinatorial Generation*. Addison-Wesley, 2013.

11 James F Korsh and Paul LaFollette. Multiset permutations and loopless generation of ordered trees with specified degree sequence. *Journal of Algorithms*, 34(2):309–336, 2000.

12 Paul W Lapey and Aaron Williams. Ordered tree iteration, 2022. URL: `https://gitlab.com/combinatronics/ordered-tree-iteration`.

13 Paul W Lapey and Aaron Williams. A shift Gray code for fixed-content Łukasiewicz words. In *International Workshop on Combinatorial Algorithms*, pages 383–397. Springer, 2022.

14 Joan M Lucas, Dominique Roelants van Baronaigien, and Frank Ruskey. On rotations and the generation of binary trees. *Journal of Algorithms*, 15(3):343–366, 1993.

15 Arturo Merino and Torsten Mütze. Efficient generation of rectangulations via permutation languages. In *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

16 Torsten Mütze. Combinatorial Gray codes-an updated survey. *arXiv preprint*, 2022. `arXiv:2202.01280`.

17 Torsten Mütze. Cos++. the combinatorial object server, 2022. URL: `http://combos.org`.

18 Shin-ichi Nakano. A gray code of ordered trees. *arXiv preprint*, 2022. `arXiv:2207.01129`.

19 Victor Parque and Tomoyuki Miyashita. An efficient scheme for the generation of ordered trees in constant amortized time. In *2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, pages 1–8. IEEE, 2021.

20 Andrzej Proskurowski and Frank Ruskey. Binary tree Gray codes. *Journal of Algorithms*, 6(2):225–238, 1985.

21 Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.

22 Frank Ruskey and Andrzej Proskurowski. Generating binary trees by transpositions. *Journal of Algorithms*, 11(1):68–84, 1990.

23 Frank Ruskey, Joe Sawada, and Aaron Williams. Binary bubble languages and cool-lex order. *Journal of Combinatorial Theory, Series A*, 119(1):155–169, 2012.

24 Frank Ruskey, Joe Sawada, and Aaron Williams. De Bruijn sequences for fixed-weight binary strings. *SIAM Journal on Discrete Mathematics*, 26(2):605–617, 2012.

25 Frank Ruskey and Aaron Williams. Generating balanced parentheses and binary trees by prefix shifts. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 107–115, 2008.

26 Frank Ruskey and Aaron Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, 2009.

27 Carla Savage. A survey of combinatorial Gray codes. *SIAM review*, 39(4):605–629, 1997.

28 Joe Sawada and Aaron Williams. Efficient oracles for generating binary bubble languages. *The electronic journal of combinatorics*, pages P42–P42, 2012.

29 Joe Sawada and Aaron Williams. A universal cycle for strings with fixed-content (which are also known as multiset permutations). In *Workshop on Algorithms and Data Structures*, pages 599–612. Springer, 2021.

30 Wladyslaw Skarbek. Generating ordered trees. *Theoretical Computer Science*, 57(1):153–159, 1988.

31 Richard P Stanley. *Catalan numbers*. Cambridge University Press, 2015.

32 Brett Stevens and Aaron Williams. The coolest way to generate binary strings. *Theory of Computing Systems*, 54(4):551–577, 2014.

33 Vincent Vajnovszki and Timothy Walsh. A loop-free two-close Gray-code algorithm for listing k-ary Dyck words. *Journal of Discrete Algorithms*, 4(4):633–648, 2006.

34 D Roelants Van Baronaigien. A loopless algorithm for generating binary tree sequences. *Information Processing Letters*, 39(4):189–194, 1991.

35 Aaron Williams. Loopless generation of multiset permutations by prefix shifts. In *SODA 2009, Symposium on Discrete Algorithms*, 2009.

36 Aaron Williams. The greedy gray code algorithm. In *Workshop on Algorithms and Data Structures*, pages 525–536. Springer, 2013.

37 Aaron Michael Williams. *Shift Gray codes*. PhD thesis, University of Victoria, 2009.

38 Bo Yao, Hongyu Chen, Chung-Kuan Cheng, and Ronald Graham. Floorplan representations: Complexity and connections. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 8(1):55–80, 2003.

A C Implementation – With Parent Pointer

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node { struct node *parent, *first, *right; } node;

void pull(node *A, node *B){
  node *child = B->first;
  B->first = child->right;
  child->right = A->first;
  A->first = child;
  child->parent = A;
}

void visit(node* root){ // prints an ordered tree as a Dyck word
  node* child = root->first;
  while(child){
    putchar('1');
    visit(child);
    child = child->right;
    putchar('0');
  }
  if (!root->parent) putchar('\n');
}

void coolOtree(int n) {
  node* root = (node*)calloc(sizeof(node),1);
  node* curr = root;
  for(int i = 0; i < n; i++){
    curr->first = (node*)calloc(sizeof(node),1);
    curr->first->parent=curr;
    curr=curr->first;
  }
  pull(root,curr->parent);
  node *p, *o = root->first->right;
  visit(root);
  while (o) {
    p=o->parent;
    if (o->first) { // if o has a child, o pulls p
      pull(o,p);
      o = o->first->right;
    } else {
      if (p == root) { // if p is the root, o pulls p
        pull(o,p);
      } else { // otherwise, g pulls p and root pulls p
        pull(p->parent,p);
        pull(root,p);
      }
      o = o->right;
    }
    visit(root);
  }
}

int main(int argc, char **argv) { coolOtree(atoi(argv[1])); }
```

B C Implementation – Without Parent Pointer

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node { struct node *first; struct node *right; } node;

void pull(node *A, node *B){
  node *pulled = B->first;
  B->first = pulled->right;
  pulled->right = A->first;
  A->first = pulled;
}

void visitrec(node* n){ // prints an ordered tree as a Dyck word;
  node* child = n->first;
  while(child){
    putchar('1');
    visitrec(child);
    child = child->right;
    putchar('0');
  }
}
void visit(node* n){ visitrec(n); putchar('\n');}

void coolOtree(int n) {
  node *root = (node*)calloc(sizeof(node),1);
  node *prev, *curr = root;
  for(int i = 0; i < n; i++){
    curr->first = (node*)calloc(sizeof(node),1);
    prev=curr;
    curr=curr->first;
  }
  pull(root,prev);
  node *p=root, *g=NULL, *o = root->first->right;
  visit(root);
  while (o) {
    if (o->first) { // if o has a child, o pulls p
      pull(o,p);
      g=p;
      p=o;
      o = o->first->right;
    } else {
      if (p == root) { // if p is the root, o pulls p
        pull(o,p);
      } else { // otherwise, g pulls p and root pulls p
        pull(g,p);
        pull(root,p);
        p=root; //g could be set to null
      }
      o = o->right;
    }
    visit(root);
  }
}
int main(int argc, char **argv) { coolOtree(atoi(argv[1])); }
```

Popular Edges with Critical Nodes

Kushagra Chatterjee[1]
National University of Singapore, Singapore

Prajakta Nimbhorkar
Chennai Mathematical Institute, India

Abstract

In the *popular edge problem*, the input is a bipartite graph $G = (A \cup B, E)$ where A and B denote a set of men and a set of women respectively, and each vertex in $A \cup B$ has a strict preference ordering over its neighbours. A matching M in G is said to be *popular* if there is no other matching M' such that the number of vertices that prefer M' to M is more than the number of vertices that prefer M to M'. The goal is to determine, whether a given edge e belongs to some popular matching in G. A polynomial-time algorithm for this problem appears in [5].

We consider the popular edge problem when some men or women are prioritized or critical. A matching that matches all the critical nodes is termed as a feasible matching. It follows from [15, 20, 25, 24] that, when G admits a feasible matching, there always exists a matching that is popular among all feasible matchings.

We give a polynomial-time algorithm for the popular edge problem in the presence of critical men or women. We also show that an analogous result does not hold in the many-to-one setting, which is known as the Hospital-Residents Problem in literature, even when there are no critical nodes.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Matching, Stable Matching, Popular feasible Matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.54

Related Version *Full Version*: https://arxiv.org/abs/2209.10805v1

Funding *Kushagra Chatterjee*: Supported in part by an MoE AcRF Tier 2 grant (WBS No. A-8000416-00-00) and an ODPRT grant (WBS No. A-0008078-00-00).
Prajakta Nimbhorkar: Supported in part by SERB Award CRG/2019/004757.

1 Introduction

The stable marriage problem is well-studied in literature. The input instance is a bipartite graph $G = (A \cup B, E)$ where A and B denote the sets of men and women respectively, and each vertex has a strict preference ordering on its neighbors. The preference ordering is referred to as the *preference list* of the vertex. Such an instance is referred to as a *marriage instance*. A matching M in G is said to be *stable* if there is no pair $(a, b) \in E$ such that both a and b prefer each other over their respective partners in M, denoted as $M(a)$ and $M(b)$ respectively. A matching is called *unstable* if such a pair (a, b) exists, and such a pair (a, b) is called a *blocking pair*. In their seminal paper, Gale and Shapley showed that stable matchings always exist and can be computed in linear time [8]. However, all the stable matchings match the same set of vertices [9] and they can be as small as half the size of a maximum matching [13]. Hence popularity has been considered as an alternative to stability.

[1] Work was done when the author was a Master's student in Chennai Mathematical Institute.

▶ **Definition 1** (Popular Matching). *A matching M is said to be* popular *in a marriage instance G if, for all matchings N in G, the number of vertices that prefer N over M is no more than the number of vertices that prefer M over N. We denote the number of vertices preferring the matching M over N using the symbol $\phi(M, N)$.*

In other words, M is popular if it does not lose a head-to-head election with any other matching N where votes are cast by vertices. This notion was introduced by Gärdenfors [10] and has been well-studied since then (see Section 1.3). Popular matchings always exist since stable matchings are popular, in fact, stable matchings are minimum size popular matchings [13]. A subclass of maximum size popular matchings called *dominant matchings* was identified in [5].

▶ **Definition 2** (Dominant Matching). *A matching M in a marriage instance G is called a* dominant matching *if M is popular, and for each N such that $|N| > |M|$, the number of vertices that prefer M to N is more than the number of vertices that prefer N to M.*

Informally, a matching M is a dominant matching if M is popular and M wins against any other matching N which is larger than M. Note that a dominant matching is clearly a maximum size popular matching but a maximum size popular matching need not be a dominant matching.

Cseh and Kavitha [5] addressed the problem of determining whether there is a popular matching containing a given edge e, referred to as the *popular edge problem*. They gave a polynomial-time algorithm for this problem. This is surprising since, in [7], it was shown that stable matchings and dominant matchings are the only two tractable subclasses of popular matchings, and it is NP-hard to find a popular matching which is neither stable nor dominant.

Popular matchings find applications in situations where certain nodes are prioritized or critical and they are required to be matched. A real-life example of this scenario is assignment of sailors to billets in the US Navy [26, 29, 20] where certain billets are required to be matched. Rural hospitals often face the problem of understaffing in the National Resident Matching Program in the USA [27, 28]. Thus marking some positions in these hospitals as critical and finding a critical matching provides a way to address this issue. While matching students to mentors, it may be required to assign mentors to all the students whose past performance is below a certain threshold. In several other applications, a subset of people needs to be prioritized based on their economic, ethnic, geographic, or medical backgrounds. A matching that matches all the prioritized or *critical* nodes is termed as a *feasible* matching. Such a scenario has been considered in [24] and [25] in the many-to-one setting, and it is shown that there always exists a matching that is popular within the set of feasible matchings. In [20], a matching that matches as many critical nodes as possible has been referred to as a *critical matching*. It is shown in [20] that a matching that is popular in the set of critical matchings, called a *popular critical matching*, always exists and a polynomial time algorithm is given for the same. A special case of this is addressed in [15], where all the nodes are critical, and hence a critical matching is a matching that is popular amongst all maximum size matchings. A polynomial-time algorithm is given in [15] for this problem. In the presence of critical men or women, popular edge problem for feasible matchings is a natural question that arises in this context. Thus, given a marriage instance $G = (A \cup B, E)$, a set of critical nodes $C \subseteq A$, and an edge e, the problem is to determine whether there is a feasible matching containing e that is popular within the set of feasible matchings. We call this the *popular feasible edge problem*. In our work we only look for the case when $C \subseteq A$ we keep the case when $C \subseteq (A \cup B)$ as an interesting open problem.

▶ **Definition 3** (Popular feasible matching). *Given a marriage instance $G = (A \cup B, E)$, and a set of critical nodes C, a feasible matching that is popular among all the feasible matchings is called a* popular feasible matching.

We also define dominant feasible matchings below.

▶ **Definition 4** (Dominant feasible matching). *Given a marriage instance $G = (A \cup B, E)$ and a set of critical nodes C, a matching M is called a* dominant feasible matching *if M is a popular feasible matching, and for all the feasible matchings N such that $|N| > |M|$, M gets strictly more votes than N.*

1.1 Our contributions

We show the following main result in this paper:

▶ **Theorem 5.** *Given a marriage instance $G = (A \cup B, E)$ along with a set of critical nodes $C \subseteq A$, an edge $e \in E$ belongs to a popular feasible matching in G if and only if e belongs to a minimum size popular feasible matching or a dominant feasible matching in G.*

Theorem 5, along with the following results, leads to a polynomial-time algorithm for the popular critical edge problem.

▶ **Theorem 6.** *There are polynomial-time reductions from a given instance G with a set of critical nodes C to marriage instances G' and G'' such that there is a surjective map from stable matchings in G' to minimum size popular feasible matchings in G and there is a surjective map from stable matchings in G'' to dominant feasible matchings in G.*

The reductions are similar to those in [24, 25, 20], however, the surjectivity of the maps is not shown there. In [21], a similar reduction is given and the surjectivity of the map is shown using dual certificates, whereas our proofs of surjectivity are combinatorial.

Counter-example for the many-to-one setting. We show that a result analogous to Theorem 5 does not generalize to the many-to-one setting referred to as the *Hospital-Residents problem* in literature, even when there are no critical nodes. Figure 1 shows such an example. Informally, popularity in the many-to-one setting is defined as follows. To compare two matchings M and N, a hospital casts as many votes as its upper quota. It compares the sets of residents $M(h)$ and $N(h)$ that it gets in the matchings M and N respectively by fixing any correspondence function between $M(h) \setminus N(h)$ and $N(h) \setminus M(h)$. For the formal definition of popularity in the many-to-one setting in the presence of critical nodes, we refer the reader to [25, 24], where it is shown that the respective algorithms output a matching that is popular under any choice of the correspondence function.

1.2 Overview of our algorithm

We give a brief outline of our algorithm. After proving Theorem 5, the algorithm to determine whether an edge e belongs to a popular feasible matching goes as follows:

- **(i)** Check whether e belongs to a minimum size popular feasible matching. If so, output yes and stop, otherwise go to the next step.
- **(ii)** Check whether e belongs to a dominant feasible matching. If so, then output yes and stop. If not, then conclude that e does not belong to a popular feasible matching in that instance by Theorem 5.

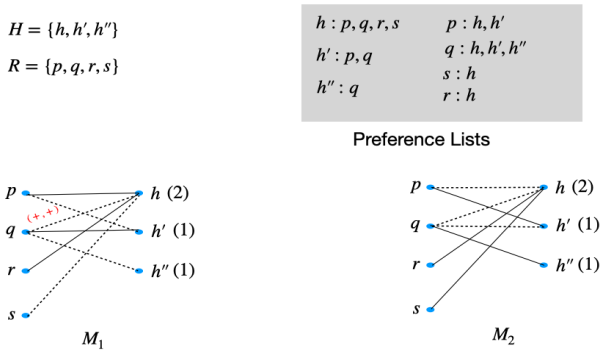


Figure 1 Here H and R are the sets of hospitals and residents respectively, h has upper quota or capacity 2, other hospitals have upper quota 1. The only stable matching is $M = \{(p, h), (q, h)\}$ of size 2 whereas the only dominant matching is M_2, of size 4. The edge (q, h') belongs to a popular matching M_1 of size 3, but does not belong to the stable matching M or to the dominant matching M_2. Thus Theorem 5 does not hold for this instance.

For steps (i) and (ii) above, we use the reductions mentioned in Theorem 6. For an edge e in G, there are multiple edges in G' and G'' corresponding to e. The stable edge algorithm of [23] can be used to determine whether any of the edges that correspond to e in G' or G'' is contained in some stable matching in G' or G''. The details are given in Section 3. To prove Theorem 5, we assume that e is contained in a popular feasible matching M which is neither a minimum size popular feasible matching nor a dominant feasible matching. We give a *Partition Method* in Section 3.1 which partitions the given instance into three parts. We call the restrictions of M on the three parts as M_d, M_m and M_r. Since e is contained in M, e must belong to one of the three parts viz. M_d, M_m and M_r. If $e \in (M_d \cup M_r)$, we convert the matching M_m to another matching M'_d which is a dominant feasible matching in that part, and show that $M^*_d = (M_d \cup M'_d \cup M_r)$ is a dominant feasible matching in the whole instance. Thus e is contained in a dominant feasible matching, namely M^*_d. Similarly, if $e \in (M_m \cup M_r)$ then we convert the matching M_d to another matching M'_m which is a minimum size popular feasible matching in the respective part, and moreover, $M^*_m = (M'_m \cup M_m \cup M_r)$ is a minimum size popular feasible matching in the whole instance. Thus e belongs to the minimum size popular feasible matching M^*_m.

1.3 Related Results

Gale and Shapley proposed an algorithm to find a stable matching in a marriage instance in their seminal paper [8]. The notion of popular matching was introduced by Gärdenfors [10]. Popular matchings in the marriage instance have been considered first in [1, 13, 15]. An $O(|E|)$-time algorithm to find a dominant matching in a marriage instance is given in [15]. In [15], a size-popularity tradeoff has been considered, and a polynomial-time algorithm for finding a maximum matching that is popular among all maximum matchings is given. The popular edge problem is inspired by the *stable edge problem*. The stable edge problem involves deciding whether a given edge e belongs to a stable matching in a Stable Marriage instance. A polynomial-time algorithm for the stable edge problem is given in the book by Knuth[23]. Cseh and Kavitha [5] addressed the popular edge problem and gave an $O(|E|)$ time algorithm for the same. Later, Faenza et. al [7] show that the problem of deciding whether an instance admits a popular matching containing a set of two or more edges is NP-Hard. In that paper, the authors also show that finding a popular matching in a stable marriage instance, which is neither stable nor dominant is NP-Hard.

In [24], the authors showed that a popular feasible matching always exists in an HRLQ instance. This has been further generalized by Nasre et al. [25] to the HRLQ case with critical residents. While our work and [24, 25] deal with instances that admit a feasible matching, the work of Kavitha [20] contains an algorithm to find a popular critical matching i.e., a matching that matches maximum possible number of critical nodes and is popular among all such matchings. Problems related to HRLQ have also been considered in [12] and [2] in different settings. Besides this, there has been a lot of recent work on various aspects of popular matchings and their generalizations e.g. weighted popular matchings, quasi-popular matchings, extended formulations, popular matchings with one-sided bias, dual certificates to popularity, popular matchings polytope and its extension complexity, hardness and algorithms for popular matchings in case of ties in preferences etc. [21, 18, 19, 22, 6, 17, 14, 16, 4, 11].

A comparison with [5]. Cseh and Kavitha in their paper [5] presented an $O(|E|)$-time algorithm for the popular edge problem. Our result follows a similar template as theirs, although unlike that in [5] where nodes are divided into two levels, we have nodes divided into a number of levels proportional to the number of critical nodes. Also, we need to partition the given instance into three parts, all of which can have blocking pairs, whereas in [5], all the blocking pairs can be put into only one of the two parts straight away.

1.4 Organization of the paper

In Section 2, we give the reductions from a marriage instance with critical men to marriage instances without critical nodes. In Section 3, we prove Theorem 5 and discuss the popular edge algorithm.

2 The Reductions

We describe the reductions from a marriage instance $G = (A \cup B, E)$ with a critical node set $C \subseteq A$ to marriage instances G' and G'' such that there is a surjective map from the set of stable matchings in G' to the set of minimum size popular feasible matchings in G, and a surjective map from the set of stable matchings in G'' to the set of dominant feasible matchings in G, thereby proving Theorem 6. We recall some notation below, that is standard in popular matchings literature (e.g. [13, 15, 5] etc.)

▶ **Definition 7** (Edge labels). *Given a matching M in G, a vertex u assigns a label $+1$ (respectively -1) to an edge (u, v) incident on it if $(u, v) \notin M$ and u prefers v over its partner in M denoted by $M(u)$ (respectively $M(u)$ over v). Thus each edge (u, v) gets two labels, one from u and the other from v.*

By above definition, an edge not present in a given matching M can get one of the four labels $(+1, +1), (+1, -1), (-1, +1), (-1, -1)$. We use the convention that the first label in the pair is from a vertex in A and the second label is from a vertex in B. Any vertex prefers to be matched to one of its neighbors over remaining unmatched.

2.1 Reduction from G to G'

Given an instance G, the instance G' is constructed as follows. Let $C \subseteq A$ be the set of critical nodes and $\ell = |C|$.

- **The set A':** For each $m \in C$, A' has $(\ell+1)$ copies of m, denoted by the set $A'_m = \{m^0, m^1, ..., m^\ell\}$. We refer to $m^i \in A'$ as the *level* i copy of $m \in A$. For each $m \in A \setminus C$, A' has only one copy of m, denoted $A'_m = \{m^0\}$. Now, $A' = \bigcup_{m \in A} A'_m$.
- **The set B':** All the women in B are present in B'. Additionally, corresponding to each $m \in C$, B' contains ℓ dummy women denoted by the set $D_m = \{d_m^1, d_m^2, d_m^3, ..., d_m^\ell\}$. We call d_m^i as the *level* i *dummy woman for* m. For $m \in A \setminus C$, $D_m = \emptyset$. Now, $B' = B \cup \bigcup_{m \in A} D_m$.

We denote by Pref(m) and Pref(w) the preference lists of $m \in A$ and $w \in B$ respectively. Let Pref$(w)^i$ be the list of level i copies of men present in Pref(w), if these copies exist. We now describe the preference lists in G'. Here $\circ$ denotes the concatenation of two lists.

m^0 s.t. $m \in A \setminus C$	:	Pref(m)
$m \in C$, $i \in \{0, \ldots, \ell\}$:		
m^0	:	Pref(m), d_m^1
m^i	:	d_m^i, Pref(m), d_m^{i+1}, $i \in \{1, \ell-1\}$)
m^ℓ	:	d_m^ℓ, Pref(m)
w s.t. $w \in B$	:	Pref$(w)^\ell \circ$ Pref$(w)^{\ell-1} \circ \ldots \circ$Pref$(w)^0$
d_m^i,$i \in \{1, \ldots, \ell\}$	:	m^{i-1}, m^i

2.2 Correctness of the reduction

After constructing G', the mapping of a stable matching M' in G' to a minimum size popular feasible matching M in G is a simple and natural one: For $m \in A$, define the set $M(m) = B \cap \bigcup_{m \in A} M'(m^i)$, which is the set of non-dummy women matched to any copy of m in A'. In the rest of this section, the term *image* always refers to the image under this map. It remains to prove that M is a minimum size popular feasible matching i.e., M is a matching in G, it is feasible, popular, and no matching smaller than M is popular. We define some terminology first. A man $m \in A$ and his matched partner $w \in B$ in M are said to be at level i if $(m^i, w) \in M'$. A man $m \in A$ which is unmatched in M is said to be at level i if m^i in A' is unmatched in M'. All unmatched women are said to be at level 0. Now we give a sufficient condition for a minimum size popular feasible matching in G, the proof appears in [3].

▶ **Theorem 8.** *The image M of a stable matching M' in G' is a minimum size popular feasible matching in G and it satisfies the following conditions. Moreover, any matching M that satisfies the following conditions for some assignment of levels to vertices of G is a minimum size popular feasible matching.*

1. *All $(+1, +1)$ edges are present in between a man at level i and a woman w at level j where $j > i$.*
2. *All edges between a man at level i and a woman at level $(i-1)$ are $(-1, -1)$ edges.*
3. *No edge is present between a man at level i and a woman at level j where $j \leq (i-2)$, and all the edges of M are between vertices at the same level.*
4. *All unmatched men are at level 0.*

2.3 Surjectivity of the map

In this section, the goal is to prove the following theorem:

▶ **Theorem 9.** *For every minimum size popular feasible matching M in G, there exists a stable matching M' in G' such that M is the image of M'.*

To show the surjectivity i.e. the fact that every minimum size popular feasible matching M in G has a stable matching M' in G' as its pre-image, we first assign levels to nodes in G with respect to M. From the assignment of levels to nodes in G, the pre-image M' is then immediate. The assignment of levels is described in Algorithm 1. In the pseudocode for Algorithm 1, we denote the level of a vertex v by $level(v)$, and the matched partner of v in M as $M(v)$. The proof of Theorem 9 is immediate from the correctness of Algorithm 1, proved below.

Algorithm 1 Leveling Algorithm for minimum size popular feasible matching.

Input: A marriage instance G, set of critical nodes $C \subseteq A$, a minimum size popular feasible matching M in G

Output: Assignment of levels to the vertices in G based on the matching M.

Initially all the men and the women are assigned level 0
flag = true
while flag = true **do**
    check1 = 0, check2 = 0, check3 = 0
    while $\exists\, m \in A, w \in B$ s.t. $level(m) = i$, $level(w) = j$, $j \leq i$, and (m, w) is a $(+1, +1)$ edge **do**
        Set $level(w) = level(M(w)) = i + 1$ ▷ Note that w cannot be unmatched in M because then $M \setminus (m, M(m)) \cup (m, w)$ is more popular than M and hence M would not be a popular feasible matching.
        check1 = 1
    while $\exists\, m \in A, w \in B$, s.t. $level(m) = i$, $level(w) = j$, $j < i$ and (m, w) is a $(+1, -1)$ or a $(-1, +1)$ edge **do**
        Set $level(w) = level(M(w)) = i$ ▷ Note that w cannot be unmatched in M because then M would not be a popular feasible matching.
        check2 = 1
    while $\exists m \in A, w \in B$ s.t. $level(m) = i$, $level(w) = j$, $j \leq (i - 2)$ and (m, w) is a $(-1, -1)$ edge **do**
        Set $level(w) = level(M(w)) = i - 1$ ▷ Note that w cannot be unmatched in M because then M would not be a PFM.
        check3 = 1
    if check1 = 0 and check2 = 0 and check3 = 0 **then**
        flag = false

In Algorithm 1, the Boolean variables check1, check2 and check3 are used to check whether the assignment of levels at any point violates one of the conditions of Theorem 8. If not, then we set flag to false and the algorithm terminates. In Theorem 10 below, we show that no level is empty. Since level of a vertex never reduces during the execution of Algorithm 1, it implies that the algorithm terminates.

▶ **Theorem 10.** *For a man m at level i there exists (i) either a woman w at each level j, where $j < i$, or (ii) an unmatched man m_0, if $j = 0$ such that there is an alternating path from w to m or from m_0 to m which consists of $(i - j)$ more $(+1, +1)$ edges than $(-1, -1)$ edges.*

The following corollary is a straight forward consequence of Theorem 10 and the fact that M is a minimum size popular feasible matching.

▶ **Corollary 11.** *All non-critical men and unmatched women are assigned level zero and the critical men are assigned level less than or equal to $|C|$ by Algorithm 1.*

Proof of Corollary 11. Suppose there is a non-critical man m_i at level $i, i > 0$. Now, from Theorem 10, there is an alternating path, say ρ, from a woman w_0 at level 0 or from an unmatched man m_0 to m_i which has i more $(+1, +1)$ edges than $(-1, -1)$ edges. Let $N = M \oplus \rho$. Observe that Algorithm 1 assigns level 0 to all the unmatched men in M, so m_i is matched. Now, it is easy to see that N is also a popular feasible matching. But $|N| < |M|$, so this contradicts the assumption that M is a minimum size popular feasible matching. ◀

After assigning levels to the vertices in G, the pre-image of M i.e. a stable matching M' in G' is constructed as follows. If a man m in G gets assigned level i then $M'(m_i) = M(m)$. If m is unmatched in M, then $m \notin C$ by feasibility of M, and m gets level 0 by Corollary 11. Then we leave m unmatched in M' as well. For $j < i$, m^j gets matched to the dummy woman d_m^{j+1} and for $j > i$, m^j gets matched to the dummy woman d_m^j. We give the proof of Theorem 12 in [3], where we prove that M' is stable in G'.

▶ **Theorem 12.** *The matching M' is a stable matching in G'.*

Similar reduction and proofs for dominant feasible matching appear in [3].

3 The Popular Edge Algorithm

Now we are ready to prove Theorem 5, from which, the popular edge algorithm is as follows. For a given edge $e = (m, w)$, we construct G', G'' using reductions from Section 2 and check if any of the edges (m^i, w) in G' or G'' belong to a stable matching in the respective instance using Knuth's algorithm for stable edges [23] If there is a minimum size popular feasible matching or a dominant feasible matching containing e then there is nothing to prove. So we need to prove the theorem for an edge e that is contained in a popular feasible matching M that is neither a minimum size popular feasible matching nor a dominant feasible matching, and show that there is also a minimum size popular feasible matching or a dominant feasible matching containing e. The proof of Theorem 5 involves the following two results:

▶ **Theorem 13.** *If M is neither a minimum size popular feasible matching or a dominant feasible matching, then $A \cup B$ can be partitioned into three parts $A_d \cup B_d$, $A_m \cup B_m$ and $A_r \cup B_r$ such that no edge of M is present in $A_i \times B_j$, $i \neq j$, where $i, j \in \{d, m, r\}$.*

We prove Theorem 13 in Section 3.1. Because of Theorem 13, it follows that the partition of $A \cup B$ also induces a partition of M into three parts, say M_d, M_m, M_r respectively. The following theorem shows that either M_d or M_m can be transformed into another matching so that the resulting matching is a minimum size popular feasible matching or a dominant feasible matching in G.

▶ **Theorem 14.** *There exist algorithms to transform:*

1. *the matching M_d to another matching M'_m in $A_d \cup B_d$ such that $M^*_m = M'_m \dot\cup M_m \dot\cup M_r$ is a minimum size popular feasible matching in G*
2. *the matching M_m to another matching M'_d in $A_m \cup B_m$ such that $M^*_d = M_d \dot\cup M'_d \dot\cup M_r$ is a dominant feasible matching in G.*

A proof of Theorem 14 is given in Section 3.2. From Theorems 13 and 14, Theorem 5 follows:

Proof of Theorem 5. Depending on the part that contains the given edge e, one of the two transformations mentioned in Theorem 14 can be applied: If $e \in M_m$ (respectively $e \in M_d$), apply the first (respectively, second) transformation from Theorem 14 i.e. convert the

matching M_d to M'_m (M_m to M'_d). Then, by Theorem 14, the resulting matching M^*_m (M^*_d) is a minimum size popular feasible matching (dominant feasible matching) in G containing e. If $e \in M_r$, we can apply any one of the two transformations mentioned in Theorem 14. Thus, in all the three cases, we get a minimum size popular feasible matching or a dominant feasible matching containing e. This completes the proof of Theorem 5. ◀

3.1 Partition Method

We prove Theorem 13 now. For partitioning $A \cup B$ and M, we first assign levels to the vertices of $A \cup B$ using Algorithm 1 described in Section 2.1. We refer to the level of a vertex $u \in A \cup B$ as $level(u)$. Since M is a popular feasible matching but not a minimum size popular feasible matching by assumption, all the non-critical men may not be at level 0. However, the following holds:

▶ **Lemma 15.** *After applying Algorithm 1 on a popular feasible matching M all non-critical men are assigned levels 0 or 1.*

Proof. If there is a non-critical man m who is assigned level $i \geq 2$, then according to Theorem 10 there exists a woman at level 0 such that m has an alternating path ρ from w with i more $(+1,+1)$ edges than $(-1,-1)$ edges. Since $i \geq 2$, the matching $M \oplus \rho$ is feasible and is more popular than M, contradicting the assumption that M is a popular feasible matching . Hence, all non-critical men are assigned levels 0 or 1. ◀

The following notions will be used in the partitioning procedure.

▶ **Definition 16** (Size Reducing Alternating Path (SRAP))**.** *An alternating path ρ with respect to a matching M is called as* SRAP *if the following conditions are satisfied:*
1. *The number of $(+1,+1)$ edges in ρ is one more than the number of $(-1,-1)$ edges in ρ,*
2. *It starts in a matched woman at level 0.*
3. *It ends in a matched non-critical man at level 1.*

▶ **Definition 17** (Size Increasing Alternating Path (SIAP))**.** *An alternating path ρ with respect to a matching M is called an* SIAP *if the following conditions are satisfied.*
1. *There are an equal number of $(+1,+1)$ and $(-1,-1)$ edges in ρ.*
2. *Its end-points are an unmatched man and an unmatched woman.*

Intuitively, if ρ is an SIAP (respectively an SRAP), then $M \oplus \rho$ gives a larger (respectively, smaller) popular feasible matching. Theorem 18 below shows that an SRAP and an SIAP must exist if M is not a minimum size popular feasible matching or a dominant feasible matching. We give the detailed proof in [3].

▶ **Theorem 18.** *If a popular feasible matching M is neither a minimum size popular feasible matching nor a dominant feasible matching, then G must contain an* SRAP *and an* SIAP *with respect to M.*

Proof. Suppose M_{min} is a minimum size popular feasible matching. Consider $M \oplus M_{min}$ which is a disjoint union of alternating paths and cycles. Since $|M_{min}| < |M|$, there exists an alternating path ρ in $M \oplus M_{min}$ whose both end-points are matched in M. By popularity of M and M_{min}, ρ must have one more $(+1,+1)$ edge than $(-1,-1)$ edges so that $\phi(M \oplus \rho, M) = \phi(M, M \oplus \rho)$. By feasibility of M_{min}, ρ must have a non-critical man m as one of its end-points, since m is unmatched in M_{min}. The level assigned to m has to be 1 because non-critical men can only be assigned levels 0 or 1 by Lemma 15. Moreover, since ρ has 1 more $(+1,+1)$ edge than $(-1,-1)$ edges, the level assigned to the other end-point w is 0. Recall that w is matched in M and unmatched in M_{min}. Hence ρ is an SRAP.

Now suppose M_d is a dominant feasible matching in G. The graph $M \oplus M_d$ is a disjoint union of alternating paths and cycles. Since $|M| < |M_d|$, there must exist an alternating path ρ in $M \oplus M_d$ whose end-points are unmatched in M and matched in M_d. Here, ρ must have an equal number of $(+1, +1)$ and $(-1, -1)$ edges, otherwise $(M \oplus \rho)$ becomes a more popular matching than M. Hence M has an SIAP. ◀

The partitioning is based on SIAP and SRAP, so the following theorem is essential for the partitioning to be well-defined. Theorem 19 below shows that no vertex belongs to both an SRAP and an SIAP:

▶ **Theorem 19.** *Given a popular feasible matching M, no vertex in G belongs to both an* SIAP *and an* SRAP*.*

Proof. Note that if a man m belongs to both an SIAP ρ and an SRAP σ, then his matched partner $M(m)$ must belong to both ρ and σ too. Also note that no man or woman unmatched in M can belong to both ρ and σ because all the men and women in an SRAP are matched in M. Suppose a matched pair (m, w) in M belongs to both ρ and σ. Let m_I and w_I be the end-points of ρ and m_R and w_R be the end-points of σ. Let the level assigned to the pair (m, w) be $i_{(m,w)}$. Since m_I and w_I are unmatched in M, both of them are assigned level 0 by Corollary 11. Now, the alternating subpath ρ_I of ρ from w_I to m must contain $i_{(m,w)}$ more $(-1, -1)$ edges than $(+1, +1)$ edges. This is because all the adjacent vertices of the women present in level i must be present at a level j where $j \leq (i + 1)$. So, ρ_I can go up by only one level that is from a level i woman it can only go to a level $i + 1$ man and, from the properties of Algorithm 1, we also know that all the edges between level i women and level $(i + 1)$ men are $(-1, -1)$ edges. Now, since an SIAP has an equal number of $(+1, +1)$ and $(-1, -1)$ edges, we have that the alternating subpath $\rho_{I'}$ of ρ from m_I to m must contain $i_{(m,w)}$ more $(+1, +1)$ edges than $(-1, -1)$ edges.

By a similar argument, the alternating subpath σ_R of σ starting from m to m_R consists of $(i_{(m,w)} - 1)$ more $(-1, -1)$ edges than $(+1, +1)$ edges. Thus the path $\beta = \rho_{I'} \circ \sigma_R$, where $\circ$ denotes concatenation, contains more $(+1, +1)$ edges than $(-1, -1)$ edges. This is because $\rho_{I'}$ contains $i_{(m,w)}$ more $(+1, +1)$ edges than $(-1, -1)$ edges, and then ρ_R has $(i_{(m,w)} - 1)$ more $(-1, -1)$ edges than $(+1, +1)$ edges. The matching $M \oplus \beta$ is more popular than M because β has one more $(+1, +1)$ edge than the number of $(-1, -1)$ edges, and $M \oplus \beta$ has the same size as that of M. This contradicts the fact that M is a popular feasible matching. Hence, no vertex in G can belong to both an SIAP and an SRAP for a given matching M. ◀

Now we give the method to partition $A \cup B$ below, as required by Theorem 13.

3.1.1 Partitioning $A \cup B$

(a) Initialize A_d, A_m, A_r, B_d, B_m, B_r to empty sets.
(b) For all unmatched men (m_u) and unmatched women (w_u) in M we do: $A_m = A_m \cup \{m_u\}$ and $B_m = B_m \cup \{w_u\}$
(c) From Theorem 18, we know that M must have an SRAP and an SIAP. For all men m_d and women w_d in each SRAP do: $A_d = A_d \cup \{m_d\}$ and $B_d = B_d \cup \{w_d\}$
(d) For all men m and women w in each SIAP do: $A_m = A_m \cup \{m\}$ and $B_m = B_m \cup \{w\}$
(e) While there exists a $(+1, +1)$ edge (m, w) such that $m \in A \backslash (A_d \cup A_m)$, $level(m) = i$, $level(w) = j$, $j \leq i + 1$ do: $A_d = A_d \cup \{m\}$ and $B_d = B_d \cup \{M(m)\}$
(f) While there exists a $(+1, +1)$ edge (m, w) such that $m \in A_m$, $level(m) = i$, $w \in B \backslash (B_d \cup B_m)$, $level(w) = j$, $j \leq i + 1$, do: $B_m = B_m \cup \{w\}$ and $A_m = A_m \cup \{M(w)\}$
(g) $A_r = A \backslash (A_d \cup A_m)$ and $B_r = B \backslash (B_d \cup B_m)$. Let M_d, M_m, M_r be the parts of M present in the induced subgraph on $A_d \cup B_d$, $A_m \cup B_m$, and $A_r \cup B_r$ respectively.

To complete the proof of Theorem 13, we need to show that the above procedure partitions $A \cup B$ i.e., the three sets $A_d \cup B_d, A_m \cup B_m, A_r \cup B_r$ are disjoint. The partition procedure always puts a vertex and its matched partner in the same partition. So it is immediate that M_d, M_m, M_r partition M. Lemmas 20, 21, 22,23 are useful in showing that the partitioning is well-defined, and in proving correctness of the transformations mentioned in Theorem 14. We retain the same assignment of levels to all the vertices as was done before partitioning. We state Lemmas here. The proofs of the lemmas are given in [3].

▶ **Lemma 20.** *For a man $m \in A_m$ at level i, there is an alternating path with i more $(+1,+1)$ edges than $(-1,-1)$ edges which starts at m_I and ends in m where m_I is an unmatched man and also an endpoint of an* SIAP *with respect to M. Analogously, for a woman $w \in B_d$ at level i there is an alternating path with $(i-1)$ more $(-1,-1)$ edges than $(+1,+1)$ edges which starts at m_R and ends in w where m_R is a non-critical man and also an endpoint of an* SRAP *with respect to M.*

▶ **Lemma 21.** *For an edge $(m,w) \in A_m \times B_d$ in G we have the following*
- **(i)** *If m is at level i and w is at level $(i+1)$ then the edge (m,w) is not a $(+1,+1)$ edge.*
- **(ii)** *If m is at level i then w cannot be at level $(i-1)$ or below.*
- **(iii)** *If m is at level i and w is at level i then (m,w) is a $(-1,-1)$ edge.*

▶ **Lemma 22.** *For an edge $(m,w) \in A_r \times B_d$ in G we have the following*
- **(i)** *If m is at level i and w is at level $(i+1)$ then the edge (m,w) is not a $(+1,+1)$ edge.*
- **(ii)** *If m is at level i then w cannot be at level $(i-1)$ or below.*
- **(iii)** *If m is at level i and w is at level i then (m,w) is a $(-1,-1)$ edge.*

▶ **Lemma 23.** *For a pair $(m,w) \in A_m \times B_r$ we have the following*
- **(i)** *If m is at level i and w is at level $(i+1)$ then the edge (m,w) is not a $(+1,+1)$ edge.*
- **(ii)** *If m is at level i then w cannot be at level $(i-1)$ or below.*
- **(iii)** *If m is at level i and w is at level i then (m,w) is a $(-1,-1)$ edge.*

3.2 Transformation Procedures

We prove Theorem 14 now. Recall that, before partitioning $A \cup B$, we have assigned levels, denoted by $level(u)$ to all the vertices $u \in A \cup B$ according to M using Algorithm 1, and that $level(u) = level(M(u))$. Our transformation procedures use these levels.

3.2.1 Transformation of M_d to M'_m

Following are the steps involved in the transformation, we refer to this as *Transformation* 1.
- **(a)** For $m \in A_d, M(m) \in B_d$, if $level(m) = level(M(m)) = i$, $i \geq 1$, then set $level(m) = level(M(m)) = i-1$
- **(b)** Mark the matched edges present among level 0 vertices as unmatched edges. So all the level 0 men in A_d are not assigned to any partner now.
- **(c)** Execute a proposal algorithm now. The men at level 0 start proposing from the beginning of their preference lists. A woman prefers a man at level j more than a man at level i where $j > i$. If a man m proposes to a woman w then w will accept m's proposal iff w is unmatched or if w prefers m more than her matched partner.
- **(d)** If a critical man m at level i where $i < |C|$ exhausts his preference list while proposing and remains unmatched then we assign level $(i+1)$ to m and m starts proposing again from the beginning of his preference list.

Let M'_m be the matching obtained after applying the above steps on the induced subgraph on $A_d \cup B_d$, and let $M^*_m = M'_m \dot{\cup} M_m \dot{\cup} M_r$ be the resulting matching in G.

3.2.2 Transformation of M_m to M'_d

This is referred to as Transformation 2 here onwards, and involves promoting all the unmatched men to level 1 and executing a similar proposal algorithm as above. Men that get unmatched during the course of the proposal algorithm continue proposing to women further down in their preference list. If they exhaust their preference list without getting matched, then they are promoted to the next higher level and continue proposing, however, non-critical men are not promoted beyond level 1. We give the formal transformation in [3]. Let the resulting matching in G be $M^*_d = M'_m \dot\cup M_d \dot\cup M_r$.

The following property is crucially used in proving that M^*_m is a minimum size popular feasible matching in G whereas M^*_d is a dominant feasible matching in G.

▶ **Lemma 24.** *For a man $m \in A_d$, if $level(m) = i, i > 0$ before applying Transformation 1 on $A_d \cup B_d$ then, after applying Transformation 1, $level(m) \in \{i-1, i\}$. The same holds for a woman $w \in B_d$. Similarly, for a man $m \in A_m$, if $level(m) = i$ before applying Transformation 2 on $A_m \cup B_m$ then, after applying Transformation 2, $level(m) \in \{i, i+1\}$. The same holds for a woman $w \in B_m$.*

Proof. We prove the property for Transformation 2. The proof for Transformation 1 is analogous.

Suppose there exists a man $m \in A_m$ who was assigned level i before applying Transformation 2 on $A_m \cup B_m$ but after applying Transformation 2 suppose m is assigned level $(i+2)$ or more. In Transformation 2 we convert the matching M_m to M^*_d. If m is unmatched in M_m then the level of m in M_m was 0 and it can be at most at level 1 in M^*_d because m is a non-critical man. So, m cannot be an unmatched man because it contradicts our assumption that the level of m in M^*_d is $(i+2)$ or more. Suppose m is matched to a woman w in M_m and the level of m is i. While applying Transformation 2 w rejected m because w got a proposal from some man m' who is better than m and is at level i. Note that w must have rejected m while applying Transformation 2 because after applying Transformation 2 level of m changes to $(i+2)$ or more from i. Now, since m is assigned level $(i+2)$ or more so m exhausts his preference list while proposing and remains unmatched at level i. So, m gets promoted to level $(i+1)$ and he starts proposing from the beginning of his preference list. Again since m is assigned level $(i+2)$ or more so m exhausts his preference list while proposing and remains unmatched at level $i+1$ but this is not possible because in the worst case m can propose to w and get matched to her. This is because w would reject m' which is at level i and m is at level $(i+1)$. Hence w would accept m's proposal and the level of w changes to $(i+1)$. Note that no man m'' can get promoted from level i to level $(i+1)$ and breaks the engagement of m and w because if this happens then in M_m the edge (m'', w) is a $(+1, +1)$ edge but since both m'' and w are in the same level i in M_m hence (m'', w) cannot be a $(+1, +1)$ edge. Hence, m does not exhaust his preference list while proposing at level $i+1$. So, after applying Transformation 2 the level of m can be either i or $(i+1)$. ◀

Theorems 25 and 26 show that the matchings output by the transformations are a minimum size popular feasible matching and a dominant feasible matching in G respectively. We prove Theorem 26 in [3].

▶ **Theorem 25.** *$M^*_m = (M'_m \cup M_m \cup M_r)$ is a minimum size popular feasible matching.*

Proof. The four conditions given in Theorem 8 are sufficient to show that a matching is a minimum size popular feasible matching. We show that M' satisfies all of them.

Before applying the Transformation 1, M satisfied conditions 1 to 3 of Theorem 8 because of the way Algorithm 1 assigns levels.After applying Transformation 1, the matching M_d changes to M'_m and the levels of the vertices in $A_d \cup B_d$ decrease by at most 1 (Lemma 24).

So, if M' does not satisfy conditions $1-3$ of Theorem 8 then it has to be because of the pairs present in $A_m \times B_d$ and $A_r \times B_d$. Now we show that the conditions are still satisfied. Below the proofs are given only for the pairs in $A_m \times B_d$. Proofs for the pairs in $A_r \times B_d$ are similar.

Let (m, w) be an edge in $A_m \times B_d$. From Lemma 21 (ii), if the level of w is i with respect to M, then m has level $j \leq i$. Now, after applying the Transformation 1, the level of w either remains i or becomes $(i-1)$. In the former case, the first condition of Theorem 8 is satisfied.In the later case, we have three possibilities: (a) either $j < (i-1)$ or (b) $j = (i-1)$, or (c) $j = i$. In case (b), (m, w) is not a $(+1, +1)$ edge (Lemma 21 (i)), in case (c), (m, w) is a $(-1, -1)$ edge (Lemma 21 (iii)). Hence, there is no $(+1, +1)$ edge in between a pair $(m, w) \in A_m \times B_d$ in the matching M' where m is at level j and w is at level i such that $j \leq i$. Hence, condition 1 is satisfied.

From Lemma 21 (ii), if the level of w is i in M, then m has level $j \leq i$. If level of w changes to $(i-1)$ after applying the Transformation 1, and if level of m is i, then due to Lemma 21(iii), (m, w) is a $(-1, -1)$ edge. Thus the condition 2 of Theorem 8 is satisfied.

From Lemma 21 (i), if w is at level i with respect to M, then level of m is $j \leq i$. If level of w changes to $(i-1)$, the conditions of Theorem 8 are still satisfied because no man in A_m adjacent to w is present at level $(i+1)$ or above.

We know that all the unmatched men are non-critical men. In the first step of Transformation 1, we decrease the level of each vertex by 1. Since the level of a non-critical man is at most 1 to begin with, and they are never promoted to a higher level in the Transformation 1, all the vertices unmatched in M^*_m remain at level 0. Since all the conditions of Theorem 8 are satisfied, M^*_m is a minimum size popular feasible matching. ◀

The following is an analogous result for Transformation 2. We refer the proof to [3].

▶ **Theorem 26.** *$M^*_d = (M_d \cup M'_d \cup M_r)$ is a dominant feasible matching in G.*

References

1. David J Abraham, Robert W Irving, Telikepalli Kavitha, and Kurt Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.
2. Péter Biró, Tamás Fleiner, Robert W Irving, and David F Manlove. The college admissions problem with lower and common quotas. *Theoretical Computer Science*, 411(34-36):3136–3153, 2010.
3. Kushagra Chatterjee and Prajakta Nimbhorkar. Popular edges with critical nodes. *Arxiv Version*, 2022.
4. Ágnes Cseh, Chien-Chung Huang, and Telikepalli Kavitha. Popular matchings with two-sided preferences and one-sided ties. *SIAM J. Discret. Math.*, 31(4):2348–2377, 2017.
5. Ágnes Cseh and Telikepalli Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.
6. Yuri Faenza and Telikepalli Kavitha. Quasi-popular matchings, optimality, and extended formulations. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 325–344. SIAM, 2020.
7. Yuri Faenza, Telikepalli Kavitha, Vladlena Powers, and Xingyu Zhang. Popular matchings and limits to tractability. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2790–2809. SIAM, 2019.
8. David Gale and Lloyd Stowell Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
9. David Gale and Marilda Sotomayor. Some remarks on the stable matching problem. *Discret. Appl. Math.*, 11(3):223–232, 1985.

10 Peter Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioral Science*, 20(3):166–173, 1975.

11 Kavitha Gopal, Meghana Nasre, Prajakta Nimbhorkar, and T. Pradeep Reddy. Many-to-one popular matchings with two-sided preferences and one-sided ties. In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Computing and Combinatorics - 25th International Conference, COCOON 2019*, volume 11653 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2019.

12 Koki Hamada, Kazuo Iwama, and Shuichi Miyazaki. The hospitals/residents problem with lower quotas. *Algorithmica*, 74(1):440–465, 2016.

13 Chien-Chung Huang and Telikepalli Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.

14 Chien-Chung Huang and Telikepalli Kavitha. Popularity, mixed matchings, and self-duality. *Math. Oper. Res.*, 46(2):405–427, 2021.

15 Telikepalli Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM J. Comput.*, 43(1):52–71, 2014.

16 Telikepalli Kavitha. Popular half-integral matchings. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 22:1–22:13, 2016.

17 Telikepalli Kavitha. Popular matchings of desired size. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018*, volume 11159 of *Lecture Notes in Computer Science*, pages 306–317. Springer, 2018.

18 Telikepalli Kavitha. Min-cost popular matchings. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, volume 182 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

19 Telikepalli Kavitha. Popular matchings with one-sided bias. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 70:1–70:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

20 Telikepalli Kavitha. Matchings, Critical Nodes, and Popular Solutions. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213, pages 25:1–25:19, 2021.

21 Telikepalli Kavitha. Maximum matchings and popularity. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 85:1–85:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

22 Telikepalli Kavitha, Tamás Király, Jannik Matuschke, Ildikó Schlotter, and Ulrike Schmidt-Kraepelin. Popular branchings and their dual certificates. In *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020*, volume 12125 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2020.

23 Donald Ervin Knuth. Marriages stables. *Technical report*, 1976.

24 Meghana Nasre and Prajakta Nimbhorkar. Popular matchings with lower quotas. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 44:1–44:15, 2017.

25 Meghana Nasre, Prajakta Nimbhorkar, Keshav Ranjan, and Ankita Sarkar. Popular Matchings in the Hospital-Residents Problem with Two-Sided Lower Quotas. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213, pages 30:1–30:21, 2021.

26 A. Robards, Paul. Applying two-sided matching processes to the united states navy enlisted assignment process, 2001.

27 Alvin E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984. doi:10.1086/261272.

28 Alvin E. Roth. On the allocation of residents to rural hospitals: A general property of two-sided matching markets. *Econometrica*, 54(2):425–427, 1986.

29 Mallory Soldner. Optimization and measurement in humanitarian operations, 2014.

Proportional Allocation of Indivisible Goods up to the Least Valued Good on Average

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Ryoga Mahara
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

We study the problem of fairly allocating a set of indivisible goods to multiple agents and focus on the proportionality, which is one of the classical fairness notions. Since proportional allocations do not always exist when goods are indivisible, approximate notions of proportionality have been considered in the previous work. Among them, proportionality up to the maximin good (PROPm) has been the best approximate notion of proportionality that can be achieved for all instances. In this paper, we introduce the notion of *proportionality up to the least valued good on average* (PROPavg), which is a stronger notion than PROPm, and show that a PROPavg allocation always exists. Our results establish PROPavg as a notable non-trivial fairness notion that can be achieved for all instances. Our proof is constructive, and based on a new technique that generalizes the cut-and-choose protocol.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Discrete Fair Division, Indivisible Goods, Proportionality

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.55

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society", and by JSPS, KAKENHI grant numbers JP19H05485, 20H05795, and 22H05001, Japan.

Acknowledgements The authors would like to thank anonymous reviewers for their comments.

1 Introduction

1.1 Proportional Allocation of Indivisible Goods

We study the problem of fairly allocating a set of indivisible goods to multiple agents under additive valuations. Fair division of indivisible goods is a fundamental and well-studied problem in Economics and Computer Science. We are given a set M of m indivisible goods and a set N of n agents with individual valuations. Under additive valuations, each agent $i \in N$ has value $v_i(\{g\}) \geq 0$ for each good g and her value for a bundle S of goods is equal to the sum of the value of each good $g \in S$, i.e., $v_i(S) = \sum_{g \in S} v_i(\{g\})$. An indivisible good can not be split among multiple agents and this causes finding a fair division to be a difficult task.

One of the standard notions of fairness is *proportionality*. Let $X = (X_1, X_2, \ldots, X_n)$ be an allocation, i.e., a partition of M into n bundles such that X_i is allocated to agent i. An allocation X is said to be *proportional (PROP)* if $v_i(X_i) \geq \frac{1}{n} v_i(M)$ holds for each agent i. In other words, in a proportional allocation, every agent receives a set of goods whose value is at least $1/n$ fraction of the value of the entire set. Unfortunately, proportional allocations do not always exist when goods are indivisible. For instance, when allocating a single indivisible good to more than one agents it is impossible to achieve any proportional allocation. Thus, several relaxations of proportionality such as PROP1, PROPx, and PROPm have been considered in the previous work.

Each of these notions requires that each agent $i \in N$ receives value at least $\frac{1}{n}v_i(M) - d_i(X)$, where $d_i(X)$ is appropriately defined for each notion. *Proportionality up to the largest valued good (PROP1)* is a relaxation of proportionality that was introduced by Conitzer et al. [17]. PROP1 requires $d_i(X)$ to be the largest value that agent i has for any good allocated to other agents, i.e., $d_i(X) = \max_{k \in N \setminus \{i\}} \max_{g \in X_k} v_i(\{g\})$. It is shown in [17] that there always exists a Pareto optimal[1] allocation that satisfies PROP1. Moreover, Aziz et al. [4] presented a polynomial-time algorithm that finds a PROP1 and Pareto optimal allocation even in the presence of chores, i.e., some items can have negative value.

Another relaxation is *proportionality up to the least valued good (PROPx)*, which is much stronger than PROP1. PROPx requires $d_i(X)$ to be the least value that agent i has for any good allocated to other agents, i.e., $d_i(X) = \min_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$. Moulin [26] gave an example for which no PROPx allocation exists, and Aziz et al. [4] gave a simpler example.

Recently, Baklanov et al. [5] introduced *proportionality up to the maximin good (PROPm)*. PROPm requires $d_i(X) = \max_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$, which shows that PROPm is the notion between PROP1 and PROPx. It is shown in [5] that a PROPm allocation always exists for instances with at most five agents, and later Baklanov et al. [6] showed that there always exists a PROPm allocation for any instance and it can be computed in polynomial time. To the best of our knowledge, PROPm has been the best approximate notion of proportionality that is shown to be achieved for all instances.

However, in some cases, PROPm is not a good enough relaxation of proportionality. Suppose that there exists a good $g \in M$ for which every agent has value at least $1/n$ fraction of the value of M. Then allocating g to some agent i and allocating all the goods in $M \setminus \{g\}$ to another agent achieves a PROPm allocation, whereas it will be better to allocate $M \setminus \{g\}$ to $N \setminus \{i\}$ in a fair manner (see Example 1). This motivates the study of better relaxations of proportionality than PROPm.

1.2 Our Contribution

In this paper, we introduce *proportionality up to the least valued good on average (PROPavg)*, a new relaxation of proportionality, and show that there always exists a PROPavg allocation for all instances. PROPavg requires $d_i(X)$ to be the average of minimum value that agent i has for any good allocated to other agents, i.e., $d_i(X) = \frac{1}{n-1} \sum_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$. It is easy to see that PROPavg implies PROPm. Note that a similar and slightly stronger notion was introduced by Baklanov et al. [5] with the name of Average-EFX (Avg-EFX), where $d_i(X) = \frac{1}{n} \sum_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$. Note that Avg-EFX is also an approximate notion of proportionality. It remains open whether an Avg-EFX allocation always exists. The following example demonstrates that PROPavg is a reasonable relaxation of proportionality compared to PROPm.

▶ **Example 1.** Suppose that $N = \{1, 2, 3\}$, $M = \{g_1, g_2, g_3, g_4\}$, and each agent has an identical additive valuation v such that $v(\{g_1\}) = 10, v(\{g_2\}) = v(\{g_3\}) = 7$, and $v(\{g_4\}) = 6$. As $v(\{g_1\}) \geq 10$, the allocation $(\{g_1, g_2, g_3, g_4\}, \emptyset, \emptyset)$ satisfies PROP1 even though agents 2 and 3 receive no good. Similarly, the allocation $(\{g_1\}, \{g_2, g_3, g_4\}, \emptyset)$ satisfies PROPm even though agent 3 receives no good. In contrast, every agent has to receive at least one good in any PROPavg allocation. Table 1 shows a comparison among some fairness notions (see Section 1.4 for the definition of EFX).

[1] An allocation $X = (X_1, \ldots, X_n)$ is Pareto optimal if there is no allocation $Y = (Y_1, \ldots, Y_n)$ such that $v_i(Y_i) \geq v_i(X_i)$ for any agent i, and there exists an agent j such that $v_j(Y_j) > v_j(X_j)$.

Table 1 Comparison among fairness notions in Example 1. The symbol "✓" (resp. "✗") indicates that the allocation satisfies (resp. does not satisfy) the corresponding fairness.

	EFX	PROPavg	PROPm	PROP1
$(\{g_1\}, \{g_2, g_4\}, \{g_3\})$	✓	✓	✓	✓
$(\{g_1\}, \{g_2, g_3\}, \{g_4\})$	✗	✓	✓	✓
$(\{g_1\}, \{g_2, g_3, g_4\}, \emptyset)$	✗	✗	✓	✓
$(\{g_1, g_2, g_3, g_4\}, \emptyset, \emptyset)$	✗	✗	✗	✓

Table 2 Relaxations of Proportionality.

	$d_i(X)$	Does it always exist?
PROPx	$\min_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$	No [4, 26]
Avg-EFX	$\frac{1}{n} \sum_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$	Open
PROPavg	$\frac{1}{n-1} \sum_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$	Yes **(our result)**
PROPm	$\max_{k \in N \setminus \{i\}} \min_{g \in X_k} v_i(\{g\})$	Yes [6]
PROP1	$\max_{k \in N \setminus \{i\}} \max_{g \in X_k} v_i(\{g\})$	Yes [17]

The main contribution of this paper is to show the existence of PROPavg allocations for all instances, which extends the existence of PROPm allocations shown by Baklanov et al. [6].

▶ **Theorem 2.** *There always exists a PROPavg allocation when each agent has a non-negative additive valuation.*

Known results on relaxations of proportionality are summarized in Table 2.

In order to prove Theorem 2, we provide an algorithm to find a PROPavg allocation. The running time of our algorithm is pseudo-polynomial, while Baklanov et al. [6] showed that a PROPm allocation can be computed in polynomial time. We discuss the time complexity in Section 5 in detail.

1.3 Our Techniques

Our algorithm can be seen as a generalization of *cut-and-choose* protocol, which is a well-known procedure to fairly allocate resources between two agents. In the cut-and-choose protocol, one agent partitions resources equally into two bundles for her valuation, and then the other agent chooses the best bundle of the two for her valuation. We generalize this protocol from two agents to n agents in the following way: some $n-1$ agents partition the goods into n bundles, and then the remaining agent chooses the best bundle among them for her valuation. To apply this protocol, it suffices to show that there exists a partition of the goods into n bundles such that no matter which bundle the remaining agent chooses, the remaining $n-1$ bundles can be allocated to the first $n-1$ agents fairly.

In our algorithm, we find such a partition by using an auxiliary graph called PROPavg-graph. A formal definition of the PROPavg-graph is given in Section 3, and our algorithm and its correctness proof are shown in Section 4. Let us emphasize that introducing the PROPavg-graph is a key technical ingredient in this paper. It is also worth noting that Hall's marriage theorem [21], a classical and famous theorem in discrete mathematics, plays an important role in our argument.

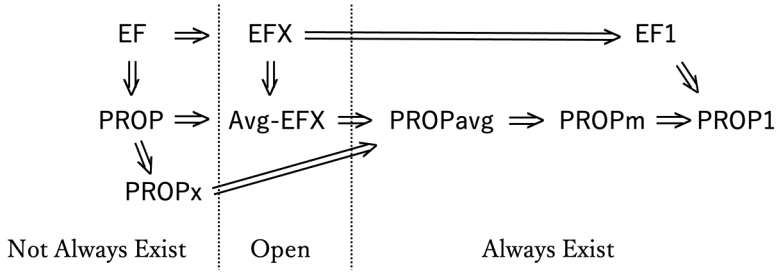


Figure 1 Relationship among some fairness notions. EF, PROP, or PROPx allocations do not always exist, while PROPavg, PROPm, EF1, and PROP1 can be achieved for all instances. It is not known whether EFX or Avg-EFX allocations always exist or not.

1.4 Related Work

Fair division of divisible resources is a classical topic starting from the 1940's [29] and has a long history in multiple fields such as Economics, Social Choice Theory, and Computer Science [9, 10, 25, 28]. In contrast, fair division of indivisible goods has actively studied in recent years (see, e.g., [2, 3]).

In the context of fair division, besides proportionality, *envy-freeness* is another well-studied notion of fairness. An allocation is called *envy-free (EF)* if for each agent, she receives a set of goods for which she has value at least value of the set of goods any other agent receives. As in the proportionality case, envy-free allocations do not always exist when goods are indivisible, and several relaxations of envy-freeness have been considered. Among them, a notable one is *envy-freeness up to one good (EF1)* [11]. It is known that there always exists an EF1 allocation, and it can be computed in polynomial time [22]. Another notable relaxation is *envy-freeness up to any good (EFX)* [13]. An allocation $X = (X_1, \ldots, X_n)$ is called EFX if for any pair of agents $i, j \in N$, $v_i(X_i) \geq v_i(X_j) - m_i(X_j)$, where $m_i(X_j)$ is the value of the least valuable good for agent i in X_j. It is one of the major open problems in fair division whether EFX allocations always exist or not. As mentioned in [5], it is easy to see that EFX implies Avg-EFX. As with EFX, it is not known whether Avg-EFX allocations always exist for instances with four or more agents. The relationship among notions mentioned above and the existence results are summarized in Figure 1.

There have been several studies on the existence of an EFX allocation for restricted cases. Plaut and Roughgarden [27] showed that an EFX allocation always exists for instances with two agents even when each agent can have more general valuations than additive valuations. Chaudhury et al. [14] showed that an EFX allocation always exists for instances with three agents. It is not known whether EFX allocations always exist even for instances with four agents having additive valuations. We can also consider the cases with restricted valuations. For example, there always exists an EFX allocation when valuations are identical [27], two types [23, 24], binary [7, 18], or bi-valued [1].

Another direction of research related to EFX is *EFX-with-charity*, in which unallocated goods are allowed. Obviously, without any constraints, the problem is trivial: leaving all goods unallocated results in an envy-free allocation. Thus, the goal here is to find allocations with better guarantees. For additive valuations, Caragiannis et al. [12] showed that there exists an EFX allocation with some unallocated goods where every agent receives at least

half the value of her bundle in a maximum *Nash social welfare* allocation[2]. For normalized and monotone valuations, Chaudhury et al. [16] showed that there exist an EFX allocation and a set of unallocated goods U such that every agent has value for her own bundle at least value for U, and $|U| < n$. Berger et al. [8] showed that the number of the unallocated goods can be decreased to $n-2$, and to just one for the case of four agents having nice cancelable valuations, which are more general than additive valuations. Mahara [24] showed that the number of the unallocated goods can be decreased to $n-2$ for normalized and monotone valuations, which are more general than nice cancelable valuations. For additive valuations, Chaudhury et al. [15] presented a polynomial-time algorithm for finding an approximate EFX allocation with at most a sublinear number of unallocated goods and high Nash social welfare.

2 Preliminaries

Let $N = \{1, \dots, n\}$ be a set of n agents and M be a set of m goods. We assume that goods are indivisible: a good can not be split among multiple agents. Each agent $i \in N$ has a non-negative valuation $v_i : 2^M \to \mathbb{R}_{\geq 0}$, where 2^M is the power set of M. We assume that each valuation v_i is *additive*, i.e., $v_i(S) = \sum_{g \in S} v_i(\{g\})$ for any $S \subseteq M$. Note that since valuations are non-negative and additive, they have to be *normalized*: $v_i(\emptyset) = 0$ and *monotone*: $S \subseteq T$ implies $v_i(S) \leq v_i(T)$ for any $S, T \subseteq M$. For ease of explanation, we normalize the valuations so that $v_i(M) = 1$ for all $i \in N$.

To simplify notation, we denote $\{1, \dots, k\}$ by $[k]$ for any positive integer k, write $v_i(g)$ instead of $v_i(\{g\})$ for $g \in M$, and use $S \setminus g$ and $S \cup g$ instead of $S \setminus \{g\}$ and $S \cup \{g\}$, respectively.

We say that $X = (X_1, X_2, \dots, X_n)$ is an *allocation of M to N* if it is a partition of M into n disjoint subsets such that each set is indexed by $i \in N$. Each X_i is the set of goods given to agent i, which we call a *bundle*. It is simply called an *allocation to N* if M is clear from context. For $i \in N$ and $S \subseteq M$, let $m_i(S)$ denote the value of the least valuable good for agent i in S, that is, $m_i(S) = \min_{g \in S}\{v_i(g)\}$ if $S \neq \emptyset$ and $m_i(\emptyset) = 0$. For an allocation $X = (X_1, X_2, \dots, X_n)$ to N, we say that an agent i is *PROPavg-satisfied* by X if

$$v_i(X_i) + \frac{1}{n-1} \sum_{k \in [n] \setminus i} m_i(X_k) \geq \frac{1}{n},$$

where we recall that $v_i(M) = 1$. In other words, agent i receives a set of goods for which she has value at least $1/n$ fraction of her total value minus the average of minimum value of the set of goods any other agent receives. An allocation X is called *PROPavg* if every agent $i \in N$ is PROPavg-satisfied by X.

Let $G = (V, E)$ be a graph. For $S \subseteq V$, let $\Gamma_G(S) = \{v \in V \setminus S \mid (s, v) \in E \text{ for some } s \in S\}$ denote the set of neighbors of S in G. For $v \in V$, let $G - v$ denote the graph obtained from G by deleting v. A *perfect matching* in G is a set of pairwise disjoint edges of G covering all the vertices of G.

3 Key Ingredient: PROPavg-Graph

In order to prove Theorem 2, we give an algorithm for finding a PROPavg allocation. As described in Section 1.3, our algorithm is a generalization of the cut-and-choose protocol that consists of the following three steps.

[2] This is an allocation that maximizes $\Pi_{i=1}^{n} v_i(X_i)$.

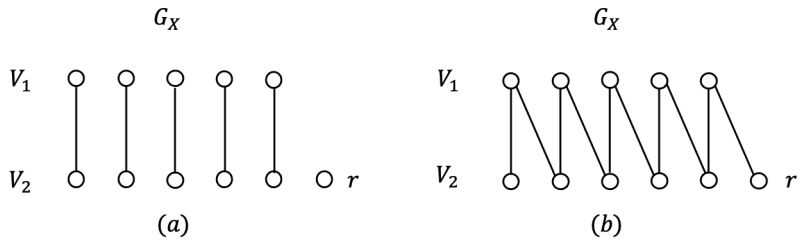


Figure 2 Examples of a PROPavg-graph G_X. If G_X is as in (a), then X satisfies (P1) but does not satisfy (P2). If G_X is as in (b), then X satisfies (P2).

1. We partition the goods into n bundles without assigning them to agents.
2. A specified agent, say n, chooses the best bundle for her valuation.
3. We determine an assignment of the remaining bundles to the agents in $N \setminus n$.

The partition given in the first step is represented by an allocation of M to a newly introduced set of size n, say V_2, and the assignment in the third step is represented by a matching in an auxiliary bipartite graph, which we call *PROPavg-graph*. In this section, we define the PROPavg-graph and its desired properties.

Let V_2 be a set of n elements and fix a specified element $r \in V_2$. We say that $X = (X_u)_{u \in V_2}$ is an *allocation to* V_2 if it is a partition of M into n disjoint subsets such that each set is indexed by an element in V_2, that is, $\bigcup_{u \in V_2} X_u = M$ and $X_u \cap X_{u'} = \emptyset$ for distinct $u, u' \in V_2$. For an allocation $X = (X_u)_{u \in V_2}$ to V_2, we define a bipartite graph $G_X = (V_1, V_2; E)$ called *PROPavg-graph* as follows. The vertex set consists of $V_1 = N \setminus n$ and V_2, and the edge set E is defined by

$$(i, u) \in E \iff v_i(X_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X_{u'}) \geq \frac{1}{n}$$

for $i \in V_1$ and $u \in V_2$. It should be emphasized that the summation is taken over $V_2 \setminus \{r, u\}$, i.e., $m_i(X_r)$ is not counted, in the above definition, which is crucial in our argument. The following lemma shows that the PROPavg-graph is closely related to the definition of PROPavg-satisfaction.

▶ **Lemma 3.** *Suppose that $G_X = (V_1, V_2; E)$ is the PROPavg-graph for an allocation $X = (X_u)_{u \in V_2}$ to V_2. Let σ be a bijection from N to V_2 and define an allocation $Y = (Y_1, \ldots, Y_n)$ to N by $Y_i = X_{\sigma(i)}$ for $i \in N$. For $i^* \in V_1$, if $(i^*, \sigma(i^*)) \in E$, then i^* is PROPavg-satisfied by Y.*

Proof. Let $u^* = \sigma(i^*)$ and suppose that $(i^*, u^*) \in E$. We directly obtain

$$v_{i^*}(Y_{i^*}) + \frac{1}{n-1} \sum_{j \in [n] \setminus i^*} m_{i^*}(Y_j) \geq v_{i^*}(X_{u^*}) + \frac{1}{n-1} \sum_{u \in V_2 \setminus \{u^*, r\}} m_{i^*}(X_u) \geq \frac{1}{n},$$

where the first inequality follows from the definition of Y and $m_{i^*}(X_r) \geq 0$, and the second inequality follows from $(i^*, u^*) \in E$. ◀

As we will see in Section 4, throughout our algorithm, we always keep an allocation $X = (X_u)_{u \in V_2}$ to V_2 that satisfies the following property.

(P1) $G_X - r$ has a perfect matching.

By updating allocation X repeatedly while keeping (P1), we construct an allocation that satisfies the following stronger property.

(P2) For any $u \in V_2$, $G_X - u$ has a perfect matching.

Examples of a PROPavg-graph G_X are shown in Figure 2. We can rephrase these conditions by using the following classical theorem known as Hall's marriage theorem in discrete mathematics.

▶ **Theorem 4** (Hall's marriage theorem [21]). *Suppose that $G = (A, B; E)$ is a bipartite graph with $|A| = |B|$. Then, G has a perfect matching if and only if $|S| \leq |\Gamma_G(S)|$ for any $S \subseteq A$.*

The property (P1) is equivalent to $|S| \leq |\Gamma_{G_X - r}(S)|$ for any $S \subseteq V_1$ by this theorem. The property (P2) is equivalent to $|S| \leq |\Gamma_{G_X - u}(S)|$ for any $u \in V_2$ and $S \subseteq V_1$ by Hall's marriage theorem. By simple observation, we can obtain another characterization of property (P2).

▶ **Lemma 5.** *Let $X = (X_u)_{u \in V_2}$ be an allocation to V_2. Then, X satisfies (P2) if and only if $|S| + 1 \leq |\Gamma_{G_X}(S)|$ for any non-empty subset $S \subseteq V_1$.*

Proof. By Hall's marriage theorem, it is sufficient to show that the following two conditions are equivalent:

(i) $|S| \leq |\Gamma_{G_X - u}(S)|$ for any $u \in V_2$ and $S \subseteq V_1$, and
(ii) $|S| + 1 \leq |\Gamma_{G_X}(S)|$ for any non-empty subset $S \subseteq V_1$.

Suppose that (i) holds. Let S be a nonempty subset of V_1. Since (i) implies that $|\Gamma_{G_X}(S)| \geq |S| \geq 1$, we obtain $\Gamma_{G_X}(S) \neq \emptyset$. Let $u \in \Gamma_{G_X}(S)$. By (i) again, we obtain $|\Gamma_{G_X}(S)| = |\Gamma_{G_X - u}(S)| + 1 \geq |S| + 1$. This shows (ii).

Conversely, suppose that (ii) holds. Let $u \in V_2$ and let $S \subseteq V_1$. If $S = \emptyset$, then it clearly holds that $|S| \leq |\Gamma_{G_X - u}(S)|$. If $S \neq \emptyset$, then we have $|S| + 1 \leq |\Gamma_{G_X}(S)| \leq |\Gamma_{G_X - u}(S)| + 1$, which implies that $|S| \leq |\Gamma_{G_X - u}(S)|$. This shows (i). ◀

4 Existence of a PROPavg Allocation

We prove our main result, Theorem 2, in this section. Our algorithm begins with obtaining an initial allocation $X = (X_u)_{u \in V_2}$ to V_2 satisfying (P1). Unless X satisfies (P2), we appropriately choose a good in $\bigcup_{u \in V_2 \setminus r} X_u$ and move it to X_r while keeping (P1). Finally, we get an allocation $X^* = (X^*_u)_{u \in V_2}$ to V_2 satisfying (P2). As we will see later, we can obtain a PROPavg allocation to N by using this allocation.

4.1 Our Algorithm

In order to obtain an initial allocation $X = (X_u)_{u \in V_2}$ to V_2 satisfying (P1), we use the following previous result about EFX-with-charity.

▶ **Theorem 6** (Chaudhury et al. [16]). *For normalized and monotone valuations, there always exists an allocation $X = (X_1, \ldots, X_n)$ of $M \setminus U$ to N, where U is a set of unallocated goods, such that*

- *X is EFX, that is, $v_i(X_i) + m_i(X_j) \geq v_i(X_j)$ for any pair of agents $i, j \in N$,*
- *$v_i(X_i) \geq v_i(U)$ for any agent $i \in N$, and*
- *$|U| < n$.*

The following lemma shows that by applying Theorem 6 to agents $N \setminus n$, we can obtain an initial allocation $X = (X_u)_{u \in V_2}$ to V_2 satisfying (P1).

▶ **Lemma 7.** *There exists an allocation $X = (X_u)_{u \in V_2}$ to V_2 satisfying (P1).*

Proof. By applying Theorem 6 to agents $N \setminus n$, we can obtain an allocation $Y = (Y_1, \dots, Y_{n-1})$ of $M \setminus U$ to $N \setminus n$, where U is a set of unallocated goods, satisfying the conditions in Theorem 6. Let $V_2 = \{r, u_1, \dots, u_{n-1}\}$ and define an allocation $X = (X_u)_{u \in V_2}$ to V_2 as $X_{u_j} = Y_j$ for $j \in [n-1]$ and $X_r = U$. Let $G_X = (V_1, V_2; E)$ be the PROPavg-graph for X. We show that X satisfies (P1).

Fix any agent $i \in V_1$. We have $v_i(X_{u_i}) + m_i(X_{u_j}) \geq v_i(X_{u_j})$ for any $j \in [n-1] \setminus i$ since Y is EFX and $X_{u_j} = Y_j$. We also have $v_i(X_{u_i}) = v_i(Y_i) \geq v_i(U) = v_i(X_r)$ and a trivial inequality $v_i(X_{u_i}) \geq v_i(X_{u_i})$. By summing up these inequalities, we obtain $n \cdot v_i(X_{u_i}) + \sum_{j \in [n-1] \setminus i} m_i(X_{u_j}) \geq \sum_{u \in V_2} v_i(X_u) = 1$. This shows that

$$v_i(X_{u_i}) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{u_i, r\}} m_i(X_{u'}) \geq v_i(X_{u_i}) + \frac{1}{n} \sum_{j \in [n-1] \setminus i} m_i(X_{u_j}) \geq \frac{1}{n},$$

and hence $(i, u_i) \in E$. Therefore, $G_X - r$ has a perfect matching $\{(i, u_i) \mid i \in [n-1]\}$, which implies (P1). ◀

The following lemma shows that if we obtain an allocation $X = (X_u)_{u \in V_2}$ to V_2 satisfying (P2), then there exists a PROPavg allocation to N.

▶ **Lemma 8.** *Suppose that $X = (X_u)_{u \in V_2}$ is an allocation to V_2 satisfying (P2). Then, we can construct a PROPavg allocation to N.*

Proof. Let $X = (X_u)_{u \in V_2}$ be an allocation to V_2 satisfying (P2). First, agent n chooses the best bundle X_{u^*} for her valuation among $\{X_u \mid u \in V_2\}$ (if there is more than one such bundle, choose one arbitrarily). Since X satisfies (P2), there exists a perfect matching A in $G_X - u^*$. For each agent $i \in V_1 (= N \setminus n)$, the bundle corresponding to the vertex that matches i in A is allocated to i. By Lemma 3, i is PROPavg-satisfied for each agent $i \in V_1$. Furthermore, since we have $v_n(X_{u^*}) = \max_{u \in V_2} v_n(X_u) \geq \frac{1}{n}$, agent n is also PROPavg-satisfied. Therefore, the obtained allocation is a PROPavg allocation. ◀

The following proposition shows how we update an allocation in each iteration, whose proof is given in Section 4.2.

▶ **Proposition 9.** *Suppose that $X = (X_u)_{u \in V_2}$ is an allocation to V_2 that satisfies (P1) but does not satisfy (P2). Then, there exists another allocation $X' = (X'_u)_{u \in V_2}$ to V_2 satisfying (P1) such that $|X'_r| = |X_r| + 1$.*

We note that, as we will see in Section 4.2, the allocation X' in Proposition 9 is obtained by moving an appropriate good $g \in \bigcup_{u \in V_2 \setminus r} X_u$ to X_r. If Proposition 9 holds, then we can show Theorem 2 as follows. See Algorithm 1 for the algorithm description.

Proof of Theorem 2. By Lemma 7, we first obtain an initial allocation $X = (X_u)_{u \in V_2}$ to V_2 satisfying (P1). By Proposition 9, unless X satisfies (P2), we can increase $|X_r|$ by one while keeping the property (P1). Since $|X_r| \leq |M|$, this procedure terminates in at most m steps, and we finally obtain an allocation X^* to V_2 satisfying (P2). Therefore, there exists a PROPavg allocation to N by Lemma 8. ◀

Algorithm 1 Algorithm for finding a PROPavg allocation.

Input: agents N, goods M, and a valuation v_i for each $i \in N$
Output: a PROPavg allocation to N

Apply Lemma 7 to obtain an allocation X to V_2 satisfying (P1).
while X does not satisfy (P2) **do**
    Apply Proposition 9 to X and obtain another allocation X' to V_2.
    $X \leftarrow X'$.
Apply Lemma 8 to obtain a PROPavg allocation to N.

4.2 Proof of Proposition 9

Let $X = (X_u)_{u \in V_2}$ be an allocation to V_2. For $u^* \in V_2 \setminus r$ and $g \in X_{u^*}$, we say that an allocation $X' = (X'_u)_{u \in V_2}$ to V_2 is obtained from X by *moving g to X_r* if

$$X'_u = \begin{cases} X_r \cup g & \text{if } u = r, \\ X_{u^*} \setminus g & \text{if } u = u^*, \\ X_u & \text{otherwise.} \end{cases}$$

The following lemma guarantees that if there exists an agent $i \in V_1$ such that $(i, r) \notin E$ in the PROPavg-graph $G_X = (V_1, V_2; E)$, then we can move some good in $\bigcup_{u \in V_2 \setminus r} X_u$ to X_r so that the edges incident to i do not disappear. This lemma is crucial in the proof of Proposition 9.

▶ **Lemma 10.** *Let $X = (X_u)_{u \in V_2}$ be an allocation to V_2 and let $i \in V_1$ be an agent such that $(i, r) \notin E$ in the PROPavg-graph $G_X = (V_1, V_2; E)$. Then, there exist $u^* \in V_2$ and $g \in X_{u^*}$ such that $(i, u^*) \in E$, $|X_{u^*}| \geq 2$, and the following property holds: if an allocation X' to V_2 is obtained from X by moving g to X_r, then the corresponding PROPavg-graph $G_{X'}$ has an edge (i, u^*).*

Proof. To derive a contradiction, assume that u^* and g satisfying the conditions in Lemma 10 do not exist. Then, we have the following claim.

▷ Claim 11. For any $u \in V_2$ with $(i, u) \in E$, we obtain

$$v_i(X_u) - m_i(X_u) + \frac{1}{n-1} \sum_{\substack{u' \in V_2 \setminus \{r,u\}: \\ (i,u') \in E}} m_i(X_{u'}) < \frac{1}{n}. \tag{1}$$

Proof of the Claim. Fix $u \in V_2$ with $(i, u) \in E$. If $X_u = \emptyset$, then we have

$$v_i(X_r) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus r} m_i(X_{u'}) \geq v_i(X_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X_{u'}) \geq \frac{1}{n},$$

where the first inequality follows from $v_i(X_u) = 0$ and the second inequality follows from $(i, u) \in E$. This contradicts $(i, r) \notin E$. Therefore, $X_u \neq \emptyset$.

Let g be a good in X_u that minimizes $v_i(g)$. Then, $v_i(g) = m_i(X_u)$. Define $X' = (X'_{u'})_{u' \in V_2}$ as the allocation to V_2 that is obtained from X by moving g to X_r. Let $G_{X'} = (V_1, V_2; E')$ be the PROPavg-graph corresponding to X'. Since u and g do not satisfy the conditions in Lemma 10 by our assumption, we have $(i, u) \notin E'$ or $|X_u| = 1$.

If $(i,u) \in E'$, then we obtain $|X_u| = 1$, and hence

$$\begin{aligned} v_i(X_r) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus r} m_i(X_{u'}) &\geq \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X_{u'}) \\ &= v_i(X'_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X'_{u'}) \\ &\geq \frac{1}{n}, \end{aligned}$$

where the equality follows from $v_i(X'_u) = v_i(\emptyset) = 0$ and the last inequality follows from $(i,u) \in E'$. This contradicts $(i,r) \notin E$.

Thus, it holds that $(i,u) \notin E'$. Since $v_i(X_u) - m_i(X_u) = v_i(X'_u)$, we obtain

$$\begin{aligned} &v_i(X_u) - m_i(X_u) + \frac{1}{n-1} \sum_{\substack{u' \in V_2 \setminus \{r,u\}: \\ (i,u') \in E}} m_i(X_{u'}) \\ &\leq v_i(X_u) - m_i(X_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X_{u'}) \\ &= v_i(X'_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X'_{u'}) \\ &< \frac{1}{n}, \end{aligned}$$

where the last inequality follows from $(i,u) \notin E'$. ◁

By summing up inequality (1) for each $u \in V_2$ with $(i,u) \in E$, we obtain the following inequality:

$$\sum_{\substack{u \in V_2: \\ (i,u) \in E}} v_i(X_u) + \left(-1 + \frac{l-1}{n-1}\right) \sum_{\substack{u' \in V_2 \setminus r: \\ (i,u') \in E}} m_i(X_{u'}) < \frac{l}{n}, \tag{2}$$

where $l = |\{u \in V_2 \mid (i,u) \in E\}|$.

On the other hand, for any $u \in V_2$ with $(i,u) \notin E$, we have

$$v_i(X_u) + \frac{1}{n-1} \sum_{\substack{u' \in V_2 \setminus r: \\ (i,u') \in E}} m_i(X_{u'}) \leq v_i(X_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X_{u'}) < \frac{1}{n}, \tag{3}$$

where the both inequalities follow from $(i,u) \notin E$. Summing up inequality (3) for each $u \in V_2$ with $(i,u) \notin E$, we obtain

$$\sum_{\substack{u \in V_2: \\ (i,u) \notin E}} v_i(X_u) + \left(\frac{n-l}{n-1}\right) \sum_{\substack{u' \in V_2 \setminus r: \\ (i,u') \in E}} m_i(X_{u'}) < \frac{n-l}{n}, \tag{4}$$

where we note that $|\{u \in V_2 \mid (i,u) \notin E\}| = n - l$.

By taking the sum of inequalities (2) and (4), we obtain

$$\sum_{\substack{u \in V_2: \\ (i,u) \in E}} v_i(X_u) + \sum_{\substack{u \in V_2: \\ (i,u) \notin E}} v_i(X_u) < 1,$$

which contradicts $\sum_{u \in V_2} v_i(X_u) = 1$.

Therefore, there exist $u^* \in V_2$ and $g \in X_{u^*}$ satisfying the conditions in Lemma 10. ◀

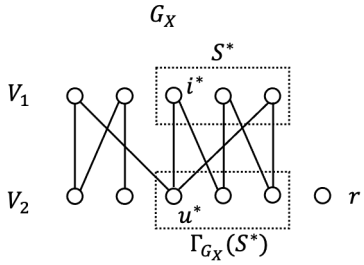


Figure 3 PROPavg-graph G_X corresponding to X in the proof of Proposition 9.

We are now ready to prove Proposition 9.

Proof of Proposition 9. Suppose that $X = (X_u)_{u \in V_2}$ is an allocation to V_2 that satisfies (P1) but does not satisfy (P2). Let $G_X = (V_1, V_2; E)$ be the PROPavg-graph corresponding to X. Since X does not satisfy (P2), there exists a non-empty set $S \subseteq V_1$ such that $|S| + 1 > |\Gamma_{G_X}(S)|$ by Lemma 5. Among such sets, let $S^* \subseteq V_1$ be an inclusion-wise minimal one. Then, $|S^*| \geq |\Gamma_{G_X}(S^*)|$ by the integrality of $|S^*|$ and $|\Gamma_{G_X}(S^*)|$, and $|S| + 1 \leq |\Gamma_{G_X}(S)|$ for any non-empty proper subset $S \subsetneq S^*$. We now show some properties of S^*.

▷ Claim 12. For any $i \in S^*$, it holds that $(i, r) \notin E$.

Proof of the claim. Since X satisfies (P1), we have $|S^*| \leq |\Gamma_{G_X - r}(S^*)|$ by Hall's marriage theorem. Hence, we obtain $|S^*| \leq |\Gamma_{G_X - r}(S^*)| \leq |\Gamma_{G_X}(S^*)| \leq |S^*|$, where the last inequality follows from the definition of S^*. This shows that all the above inequalities are tight. Since $|\Gamma_{G_X - r}(S^*)| = |\Gamma_{G_X}(S^*)|$, we obtain $r \notin \Gamma_{G_X}(S^*)$, that is, $(i, r) \notin E$ for any $i \in S^*$. ◁

▷ Claim 13. For any $i \in S^*$ and $u \in \Gamma_{G_X}(S^*)$ with $(i, u) \in E$, $G_X - r$ has a perfect matching in which i matches u.

Proof of the claim. Fix any $i \in S^*$ and $u \in \Gamma_{G_X}(S^*)$ with $(i, u) \in E$. Note that $r \notin \Gamma_{G_X}(S^*)$ by Claim 12, and hence $u \neq r$.

Since X satisfies (P1), $G_X - r$ has a perfect matching A. In A, it is obvious that every vertex in S^* is matched to a vertex in $\Gamma_{G_X - r}(S^*)$. Conversely, every vertex in $\Gamma_{G_X - r}(S^*)$ is matched to a vertex in S^* as $|S^*| = |\Gamma_{G_X - r}(S^*)|$ (see the proof of Claim 12). Thus, by removing the edges between S^* and $\Gamma_{G_X}(S^*)$ from A, we obtain a matching $A_1 \subseteq A$ that exactly covers $V_1 \setminus S^*$ and $V_2 \setminus (\Gamma_{G_X}(S^*) \cup \{r\})$.

Let G'_X be the subgraph of G_X induced by $(S^* \setminus i) \cup (\Gamma_{G_X}(S^*) \setminus u)$. We now show that G'_X has a perfect matching. Consider any $S \subseteq S^* \setminus i$. If $S = \emptyset$, then it clearly holds that $|S| \leq |\Gamma_{G'_X}(S)|$. If $S \neq \emptyset$, then $|S| + 1 \leq |\Gamma_{G_X}(S)| \leq |\Gamma_{G'_X}(S) \cup u| = |\Gamma_{G'_X}(S)| + 1$, where the first inequality follows from the minimality of S^*. Therefore, $|S| \leq |\Gamma_{G'_X}(S)|$ holds for any $S \subseteq S^* \setminus i$, and hence G'_X has a perfect matching A_2 by Hall's marriage theorem.

Then, $A_1 \cup A_2 \cup \{(i, u)\}$ is a desired perfect matching in $G_X - r$. ◁

Fix any agent $i^* \in S^*$. Since $(i^*, r) \notin E$ by Claim 12, by applying Lemma 10 to agent i^*, we obtain $u^* \in V_2$ and $g \in X_{u^*}$ satisfying the conditions in Lemma 10 (see Figure 3). Let $X' = (X'_u)_{u \in V_2}$ be the allocation to V_2 obtained from X by moving g to X_r and let $G_{X'} = (V_1, V_2; E')$ be the PROPavg-graph corresponding to X'. Then, the conditions in Lemma 10 show that $(i^*, u^*) \in E \cap E'$ and $|X_{u^*}| \geq 2$. We also see that E' satisfies the following.

▷ Claim 14. For any $i \in V_1$ and $u \in V_2 \setminus u^*$, if $(i, u) \in E$ then $(i, u) \in E'$.

Proof of the claim. Since $|X_{u^*}| \geq 2$, we have $m_i(X'_{u^*}) = m_i(X_{u^*} \setminus g) \geq m_i(X_{u^*})$ for any agent $i \in V_1$. Hence, for any $i \in V_1$ and $u \in V_2 \setminus u^*$ with $(i,u) \in E$, we obtain

$$v_i(X'_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X'_{u'}) \geq v_i(X_u) + \frac{1}{n-1} \sum_{u' \in V_2 \setminus \{r,u\}} m_i(X_{u'}) \geq \frac{1}{n},$$

which shows that $(i,u) \in E'$. ◁

By Claim 13 and $(i^*,u^*) \in E$, there exists a perfect matching A in $G_X - r$ in which i^* matches u^*. Then, Claim 14 and $(i^*,u^*) \in E'$ show that $A \subseteq E'$, that is, A is a perfect matching also in $G_{X'} - r$. Therefore, X' satisfies (P1). Since $|X'_r| = |X_r| + 1$ clearly holds by definition, X' satisfies the conditions in Proposition 9. ◀

5 Discussion

In this paper, we have introduced PROPavg, which is a stronger notion than PROPm, and shown that a PROPavg allocation always exists.

As mentioned in Section 1.2, our algorithm runs in pseudo-polynomial time, and we do not know whether it can be improved to a polynomial-time algorithm. This is because we use Theorem 6 as a subroutine in order to obtain an initial allocation X to V_2 satisfying (P1). Actually, the proof of Theorem 6 given in [16] is constructive, but it only leads to a pseudo-polynomial time algorithm. We can see that the other parts of Algorithm 1 run in polynomial time as follows. In line 2, we can check (P2) in polynomial time by applying a maximum matching algorithm for each $G_X - u$. In line 3, it suffices to find a good $g \in \bigcup_{u \in V_2 \setminus r} X_u$ such that (P1) is kept after moving g. Since (P1) can be checked in polynomial time, this can be done in polynomial time by considering all g in a brute-force way. Finally, line 5 is executed in polynomial time by a maximum matching algorithm again. Note that we can speed up lines 2 and 3 by using the DM-decomposition of G_X [19, 20], but we do not go into details, because they are not the most time consuming part. We leave it open whether a PROPavg allocation can be found in polynomial time or not.

In order to devise our algorithm, we have developed a new technique that generalizes the cut-and-choose protocol. This technique is interesting by itself and seems to have a potential for further applications. In fact, we can define a bipartite graph like the PROPavg-graph for another fairness notion, and our argument works if we obtain an allocation satisfying a (P2)-like condition. We expect that this technique will be used in other contexts as well.

References

1 Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, Alexandros Hollender, and Alexandros A Voudouris. Maximum Nash welfare and other stories about EFX. *Theoretical Computer Science*, 863:69–85, 2021.

2 Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, and Alexandros A Voudouris. Fair division of indivisible goods: A survey. *arXiv preprint*, 2022. `arXiv:2202.07551`.

3 Haris Aziz, Bo Li, Herve Moulin, and Xiaowei Wu. Algorithmic fair allocation of indivisible items: A survey and new questions. *arXiv preprint*, 2022. `arXiv:2202.08713`.

4 Haris Aziz, Hervé Moulin, and Fedor Sandomirskiy. A polynomial-time algorithm for computing a Pareto optimal and almost proportional allocation. *Operations Research Letters*, 48(5):573–578, 2020.

5 Artem Baklanov, Pranav Garimidi, Vasilis Gkatzelis, and Daniel Schoepflin. Achieving proportionality up to the maximin item with indivisible goods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5143–5150, 2021.

6 Artem Baklanov, Pranav Garimidi, Vasilis Gkatzelis, and Daniel Schoepflin. PROPm allocations of indivisible goods to multiple agents. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 24–30, 2021.

7 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Greedy algorithms for maximizing Nash social welfare. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 7–13, 2018.

8 Ben Berger, Avi Cohen, Michal Feldman, and Amos Fiat. (Almost full) EFX exists for four agents (and beyond). *arXiv preprint*, 2021. `arXiv:2102.10654`.

9 Steven J Brams, Steven John Brams, and Alan D Taylor. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

10 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016.

11 Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.

12 Ioannis Caragiannis, Nick Gravin, and Xin Huang. Envy-freeness up to any item with high Nash welfare: The virtue of donating items. In *Proceedings of the 20th ACM Conference on Economics and Computation (EC)*, pages 527–545, 2019.

13 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Transactions on Economics and Computation*, 7(3):1–32, 2019.

14 Bhaskar Ray Chaudhury, Jugal Garg, and Kurt Mehlhorn. EFX exists for three agents. In *Proceedings of the 21st ACM Conference on Economics and Computation (EC)*, pages 1–19, 2020.

15 Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, Ruta Mehta, and Pranabendu Misra. Improving EFX guarantees through rainbow cycle number. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC)*, pages 310–311, 2021.

16 Bhaskar Ray Chaudhury, Telikepalli Kavitha, Kurt Mehlhorn, and Alkmini Sgouritsa. A little charity guarantees almost envy-freeness. *SIAM Journal on Computing*, 50(4):1336–1358, 2021.

17 Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In *Proceedings of the 18th ACM Conference on Economics and Computation (EC)*, pages 629–646, 2017.

18 Andreas Darmann and Joachim Schauer. Maximizing Nash product social welfare in allocating indivisible goods. *European Journal of Operational Research*, 247(2):548–559, 2015.

19 Andrew L Dulmage. A structure theory of bipartite graphs of finite exterior dimension. *The Transactions of the Royal Society of Canada, Section III*, 53:1–13, 1959.

20 Andrew L Dulmage and Nathan S Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

21 P Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

22 Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC)*, pages 125–131, 2004.

23 Ryoga Mahara. Existence of EFX for two additive valuations. *arXiv preprint*, 2020. `arXiv:2008.08798`.

24 Ryoga Mahara. Extension of additive valuations to general valuations on the existence of EFX. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA)*, pages 66:1–15, 2021.

25 Hervé Moulin. *Fair division and collective welfare*. MIT press, 2004.

26 Hervé Moulin. Fair division in the internet age. *Annual Review of Economics*, 11:407–441, 2019.

27 Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations. *SIAM Journal on Discrete Mathematics*, 34(2):1039–1068, 2020.

28 Jack Robertson and William Webb. Cake-cutting algorithms: Be fair if you can, 1998.

29 Hugo Steinhaus. The problem of fair division. *Econometrica*, 16(1):101–104, 1948.

Pursuit-Evasion in Graphs: Zombies, Lazy Zombies and a Survivor

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Jean-Lou De Carufel
School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada

Thomas Shermer
School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

We study *zombies and survivor*, a variant of the game of cops and robber on graphs. In this variant, the single survivor plays the role of the robber and attempts to escape from the zombies that play the role of the cops. The zombies are restricted, on their turn, to always follow an edge of a shortest path towards the survivor. Let $z(G)$ be the smallest number of zombies required to catch the survivor on a graph G with n vertices. We show that there exist outerplanar graphs and visibility graphs of simple polygons such that $z(G) = \Theta(n)$. We also show that there exist maximum-degree-3 outerplanar graphs such that $z(G) = \Omega(n/\log(n))$.

Let $z_L(G)$ be the smallest number of *lazy zombies* (zombies that can stay still on their turn) required to catch the survivor on a graph G. We show that lazy zombies are more powerful than normal zombies but less powerful than cops. We prove that $z_L(G) \leq 2$ for connected outerplanar graphs and this bound is tight in the worst case. We show that $z_L(G) \leq k$ for connected graphs with treedepth k. This result implies that $z_L(G)$ is at most $(k+1)\log n$ for connected graphs with treewidth k, $O(\sqrt{n})$ for connected planar graphs, $O(\sqrt{gn})$ for connected graphs with genus g and $O(h\sqrt{hn})$ for connected graphs with any excluded h-vertex minor. Our results on lazy zombies still hold when an adversary chooses the initial positions of the zombies.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Pursuit-evasion games, Outerplanar, Graphs, Treedepth, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.56

Related Version *Full Version*: https://arxiv.org/abs/2204.11926

Funding Research supported in part by NSERC.

Acknowledgements P. Bose would like to thank P. Morin for fruitful discussions on various aspects of structural graph theory and for bringing reference [15] to his attention.

1 Introduction

The game of cops and robber was first introduced by Quillot in his doctoral thesis [21] and then independently by Nowakowski and Winkler [20]. In this pursuit-evasion game, a set of cops move along the edges of a connected and undirected graph G to catch a robber that is also moving along the edges of G. At the beginning, each cop chooses a starting vertex (multiple cops can occupy the same vertex). Then the robber chooses a starting vertex. From there, when it is the cops' turn to play, each cop decides either to stay still or to move to an adjacent vertex. When it is the robber's turn to play, the robber decides either to stay still or to move to an adjacent vertex. (In the classical version of the game, the cops and the robber are aware of each others' locations at all times.) If at least one cop shares a vertex with the robber, then the cops win. However, if the robber indefinitely avoids capture, then the robber wins. The *cop number* of a graph G, denoted as $c(G)$, is the minimum

number of cops required to catch the robber on G. Cops and robber has been studied on several classes of graphs and many variants have been studied where cops are given different restrictions or abilities (see [6] for an overview of the area). Meyniel conjectured that in general, $c(G) = O(\sqrt{n})$.

Zombies and Survivor is a variant of the game of Cops and Robber. The deterministic version of Zombies and Survivor was first introduced by Fitzpatrick et al. [13]. In this game, each cop (thought of as a *zombie*) is restricted to move along an edge incident to its current position and belonging to a shortest path to the robber (thought of as the *survivor*). Moreover, the zombie is active, in the sense that it must move on its turn. If there is more than one shortest path between a given zombie and the survivor, the zombie can choose which path to follow. Since the zombies are active, the starting configuration of the zombies actually plays a role in determining whether a survivor is caught. For example, if several zombies are placed on the same vertex in a cycle with more than 4 vertices, then they will never catch a survivor. However, 2 zombies can be strategically placed in order to always catch a survivor on such a cycle. As such, we define two types of zombie number, one where the zombies are strategically placed and the other where an adversary determines the initial position of the zombies. The *zombie number* $z(G)$ of a graph G is then defined as the minimum number of zombies required to catch the survivor on G, and the *universal zombie number* $u(G)$ is defined as the minimum number of zombies required to catch the survivor when the starting configuration of the zombies is determined by an adversary. The above example shows that $z(G) = 2$ and $u(G) = \infty$ when G is a cycle on more than 4 vertices. Since a cop has more power than a zombie, we have $c(G) \leq z(G) \leq u(G)$. From this observation, we get that zombie-win graphs are also cop-win graphs.

In their paper, Fitzpatrick et al. [13] provide an example showing that if a graph is cop-win, then it is not necessarily zombie-win. They provide a sufficient condition for a graph to be zombie-win. They also establish several results about the zombie number of the Cartesian product of graphs.

The main aspect that makes different variants of these pursuit-evasion problems quite challenging is the fact that the cop number and zombie number is not a monotonic property with respect to subgraphs. For example, both the cop number and zombie number of a clique is 1 but the cop number and zombie number of a cycle on more than 3 vertices is 2.

1.1 Contributions

In this paper, we first consider the deterministic version of Zombies and Survivor. We then turn our attention to a deterministic variant which we call *Lazy Zombies and Survivor*. In this variant, a zombie does not need to move on its turn. The *lazy zombie number* $z_L(G)$ of a graph G is therefore defined as the minimum number of lazy zombies required to catch the survivor on G. The *universal lazy zombie number* of a graph G, denoted $u_L(G)$, denotes the minimum number of lazy zombies required to catch the survivor on G, when the starting positions of the lazy zombies are chosen by an adversary. Observe that $c(G) \leq z_L(G) \leq u_L(G)$.

We show that there exist outerplanar graphs and 2-connected outerplanar graphs G with n vertices such that $z(G)/c(G) = \Omega(n)$. This improves upon a result of Bartier et al. [4] who showed that this ratio is $z(G)/c(G) = \Omega(\log n)$ for outerplanar graphs. We also show that there exist maximum-degree-3 outerplanar graphs G such that $z(G)/c(G) = \Omega\left(n/\log(n)\right)$ and there exist simple polygons whose visibility graph G is such that $z(G)/c(G) = \Omega(n)$.

Then, we show that lazy zombies are more powerful than plain zombies and less powerful than cops. Indeed, we prove that 2 lazy zombies always win (in less than $2n$ rounds) on outerplanar graphs. However, we show that there exist graphs of treewidth 2 that require 3 lazy zombies, whereas 2 cops are sufficient. We then show that k lazy zombies win after

Table 1 Summary of zombie, (universal) lazy zombie, and cop numbers. The column "zombies" shows lower bounds on the zombie number. The other two columns show upper bounds. The upper bound on the cop number for h-vertex excluded minors is for *connected* excluded minors.

	zombies	(universal) lazy zombies	cops
outerplanar	$\Theta(n)$ (Thm. 1)	2 (Thm. 6)	2 ([9])
planar	$\Theta(n)$ (Thm. 1)	$O(\sqrt{n})$ (Cor. 14)	3 ([1])
genus g	$\Theta(n)$ (Thm. 1)	$O(\sqrt{gn})$ (Cor. 14)	$3 + \lfloor \frac{3g}{2} \rfloor$ ([23])
treedepth k	$\Theta(n)$ (Thm. 1)	k (Cor. 10, Thm. 11)	$(k/2) + 1$ ([16])
treewidth k	$\Theta(n)$ (Thm. 1)	$(k+1)\log n$ (Cor. 10, Lem. 12)	$(k/2) + 1$ ([16])
h-vertex excl. minor	$\Theta(n)$ (Thm. 1)	$O(h\sqrt{hn})$ (Cor. 14)	$\frac{1}{2}(h-1)(h-2)$ ([3])

$O(n^{2k})$ rounds on graphs with treedepth k. Finally, we highlight a few implications stemming from this upper bound such as $(k+1)\log n$ lazy zombies win on graphs with treewidth k, $O(\sqrt{n})$ lazy zombies are always sufficient to win on planar graphs, $O(\sqrt{gn})$ lazy zombies win on graphs with genus g and $O(h\sqrt{hn})$ lazy zombies win on all connected graphs G with any excluded h-vertex minor H. Our upper bounds on lazy zombie numbers still hold for universal lazy zombies. These results are summarized in Table 1. Further details together with missing proofs can be found in the full version of the paper [7].

2 Preliminaries and Notation

Let G be a simple and undirected graph with vertex set $V(G)$ and edge set $E(G)$. Throughout this paper n will be used for $|V(G)|$. Given a subset $S \subseteq V(G)$, we denote the graph induced by S as $G[S]$. Given two vertices u and v in $V(G)$, we denote the shortest path from u to v as $\pi_G(u,v)$ and its length[1] as $d_G(u,v)$. If there is no path from u to v, then the length of the path is infinite. The diameter of a connected graph G, denoted as $diam(G)$, is $\max\{d_G(u,v) : u,v \in V(G)\}$.

Recall that $c(G)$ is the cop number of G, $z(G)$ (resp. $u(G)$) is the zombie number of G (resp. universal zombie number) and $z_L(G)$ (resp. $u_L(G)$) is the lazy zombie number of G (resp. universal lazy zombie number). If G is disconnected, then the cop (resp. zombie, lazy zombie) number of G is simply equal to the sum of the cop (resp. zombie, lazy zombie) numbers of its connected components. Therefore, in this paper, we assume that G is connected.

In their paper, Fitzpatrick et al. [13, Figure 5] provide an example of a graph with zombie number 1, where the zombie has to start on a specific vertex to win, which is an example where $z(G) = 1$ and $u(G) = \infty$. As alluded to in the introduction, lazy zombies are more powerful than normal zombies and less so than cops. Thus, we observe that $c(G) \leq z_L(G) \leq z(G)$.

All the pursuit-evasion games we describe in this paper consist of a sequence of rounds, each of which consists of two turns. For each round $i \geq 0$, the zombies play first (zombies' turn) and then the survivor plays (survivor's turn). In round 0, during the zombies' turn, the zombies choose their starting position (or an adversary assigns one to them), and then, during the survivor's turn, the survivor chooses its starting position. Then the zombies move (or wait if the version of the game allows them to) and the survivor moves (or waits if they decide to) in the subsequent rounds.

[1] The length of the shortest path is the number of edges on the path.

When stating results about (universal) (lazy) zombie numbers, we sometimes use asymptotic notation. In this notation, lower-bound asymptotics (Ω or Θ) will be a lower bound on the maximum-valued graph of the given class. For instance, "for outerplanar graphs G, $z(G) \in \Theta(n)$" means not only is the zombie number at most some constant times n for every outerplanar graph, but also the zombie number is at least some constant times n for some family of outerplanar graphs. Throughout the paper, unless specified, the base of the logarithmic function $\log(\cdot)$ is 2.

3 Linear bound on zombie number

In their paper, Fitzpatrick et al. ask how large the ratio $z(G)/c(G)$ can be [13, Question 19]; they note that they have not observed any graph with a ratio that exceeds 2. Here we show that this ratio can be infinite and of size $\Omega(n)$, and we show this even for outerplanar graphs of fixed radius. In independent work, Bartier et al. showed that this ratio can be infinite and of size $\Omega(\log n)$, for outerplanar graphs [4].

▶ **Theorem 1.** *Let $k \geq 2$ be an integer. Then there is a connected outerplanar graph G_k with $23k + 1$ vertices that requires at least k zombies.*

Proof. Let H be the 23-vertex graph shown in Figure 1a. H has two distinguished vertices s and t. To form the graph G_k, first take k disjoint copies $H_1, H_2, \ldots, H_k$ of H, with H_i having distinguished vertices s_i and t_i; to this add a vertex c that is connected to each s_i and each t_i (see Figure 1b).

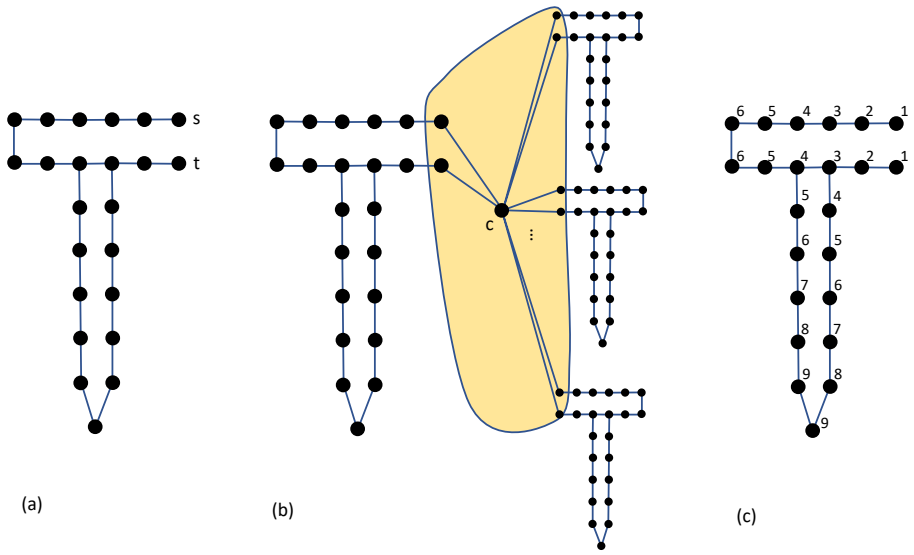


Figure 1 Construction of an outerplanar graph requiring a linear number of zombies. (a) The component graph H. (b) Connecting components into the graph G_k. (c) Each vertex of some H_j labelled with its distance to c.

By construction, G_k has $23k + 1$ vertices. Suppose that $k - 1$ or fewer zombies play on G_k. This mean that in round 0 there is some copy H_i of H that contains no zombie; the survivor chooses the vertex adjacent to s_i in H_i as its starting position, and will stay in H_i forever.

Each zombie will therefore first take a shortest path to c from wherever it starts, as this is the only way to get to H_i. Consider the time unit on which a zombie reaches c; this can be anywhere from 0 to 9, as the zombie could start on c, and 9 is the radius of G_k (and c is the center). See Figure 1c. At one time unit later, the zombie will move to s_i or t_i, whichever is closer to the survivor. We call this time the *arrival time* of the zombie; arrival times are all between 1 and 10, inclusive.

The survivor's strategy is to walk in H_i away from s_i until it reaches a vertex of degree three. At this point they walk along the 13-cycle in H_i, starting in the direction of the vertex of degree two. They continue walking this cycle forever.

The zombies will arrive on s_i if their arrival time is at most five, as this is the closest vertex to the survivor at this time. These zombies will follow the survivor's path. If a zombie has arrival time six or more, it arrives on t_i. These zombies will pursue the survivor by first walking to the 13-cycle and then following the survivor around it.

Therefore, since $k-1$ zombies are insufficient to capture the survivor on G_k, at least k are required and the lemma is proved. ◀

Note that a linear number of zombies always suffices for a graph, as we could use n zombies and initially place one on each vertex (or perhaps leave one free for the survivor). Thus we have shown that for general or for outerplanar graphs, $z(G) \in \Theta(n)$. Since the cop number for outerplanar graphs is at most two 2, the ratio $z(G_k)/c(G_k)$ is $k/2 = (n-1)/46 \in \Theta(n)$.

Modifications of the construction in the proof of Theorem 1 work for other graph classes, as illustrated by the following theorems.

▶ **Theorem 2.** *Let $k \geq 2$ be an integer. There is a 2-connected outerplanar graph G_k with $30k+1$ vertices that requires at least k zombies.*

▶ **Theorem 3.** *Let $k \geq 2$ be an integer. There is a connected graph G_k with $15k$ vertices that requires at least k zombies.*

▶ **Theorem 4.** *Let $k \geq 2$ be an integer. There is a maximum-degree-3 connected outerplanar graph G_k with at most $25k + 16k\lceil \log k \rceil - 1$ vertices that requires at least k zombies.*

Theorem 4 gives us $n \in O(k \log k)$, or $n \leq ck \log k$ for some constant c. Hence, we have $\frac{n}{\log n} \leq \frac{n}{\log k} \leq ck$, or $k \geq \frac{n}{c \log n}$. Since k zombies are required, this gives us a lower bound of $\Omega(\frac{n}{\log n})$ on the zombie number of bounded-degree graphs.

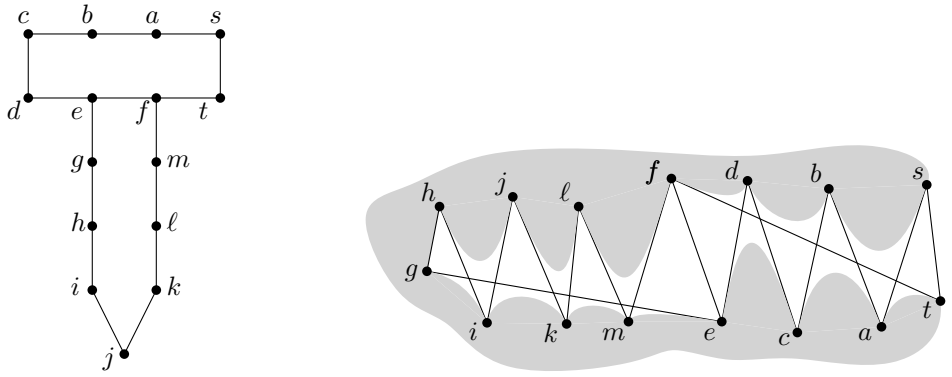
3.1 Polygon visibility graphs

A *polygonal chain* is a finite sequence V of points $v_1, v_2, \ldots, v_n$ in $\mathbf{R}^2$ (called *vertices*) along with the line segments $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n$ (called *edges*). A polygonal chain is called *closed* if $v_1 = v_n$, and *simple* if no two edges intersect except consecutive edges intersecting at their common vertex. A closed simple polygonal chain divides the plane into a finite *interior* and infinite *exterior*. A *simple polygon*, or simply *polygon*, is a closed simple polygonal chain along with its interior.

The *visibility graph* of a simple polygon P [10, 18] is a graph G where $V(G)$ is the set of vertices of the polygon, and

$$E(G) = \{(v_i, v_j) \mid \text{the segment } v_iv_j \text{ does not intersect the exterior of } P\}.$$

In particular, this means that every edge of the polygon is an edge of the visibility graph, but the visibility graph has other edges corresponding to segments that traverse the interior and possibly boundary of P. Visibility graphs are of interest in discrete geometry and have applications, for instance, in motion planning and shape analysis [8, 10].



(a) A graph fragment for proving lower bounds.

(b) An embedding of the graph fragment in the visibility graph of a polygon fragment.

Figure 2 Creation of a polygon fragment to prove a lower bound.

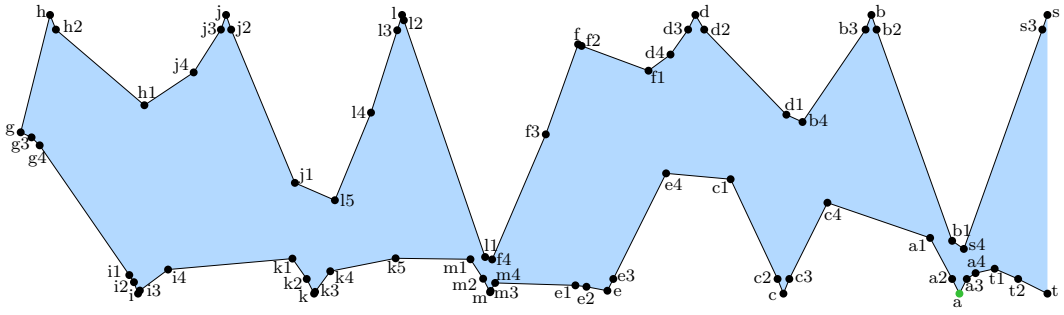


Figure 3 The polygon fragment Q. Note that at this scale, the dots for the vertex-pair $(i, i3)$ merge into what can appear to be a single dot. This also happens for $(k, k3)$ and $(m, m3)$.

Here we show that there is a linear bound on the zombie number of visibility graphs. The proof is messy in that it involves a large polygonal chain with relatively precise vertex locations in order to get a visibility graph with the desired properties. However, it is inspired by the proof of Theorem 1. Consider the graph fragment in Figure 2a. One way to embed this graph inside a polygon visibility graph is sketched in Figure 2b. Complicating matters is that we cannot get the required non-edges without placing vertices inbetween those shown in Figure 2b. Once those non-visibilities are worked out, we get the polygon fragment Q shown in Figure 3. For reproducibility, the exact vertex locations of Q are given in the full version of the paper [7].

To form a polygon whose visibility graph requires at least k zombies, we will connect k copies of Q, denoted $Q_1, Q_2, \ldots, Q_k$, placed in a geometric configuration where the only vertices of Q_i visible to Q_j are s_i (the copy of s in Q_i) and t_i (similar). Thus S, the collection of all s_i's and t_i's, will form a clique in the visibility graph. This is done by taking a small sliver of a circular arc and placing the $2k$ vertices of S evenly along it. If the sliver is small enough, any vertices inside $Q_i \setminus S$ will see only the relative interior of the polygon edge $t_k s_1$. (Another method of ensuring this is to scale each Q_i up in its x-coordinate, effectively pushing interior vertices away from $s_i t_i$. Such a scaling operation does not affect the visibility graph of Q_i.)

The proof that this polygon has zombie number k now roughly follows that of Theorem 1. Suppose that less than k suffices. Then, in the zombies' initial placement, there will be one copy of Q, say Q_i, that has no zombies in it. Start with the survivor on vertex a of Q_i (refer to Figure 2a for the labelling of the vertices).

Q is constructed so that it takes at most 6 turns for a zombie to leave the fragment (or 5 turns at most to get to s or t). This means that if the survivor stays in Q_i for 6 turns (it will), all zombies will have arrived in Q_i. The survivor's strategy will be to walk from a to b, to c, to d, e, g, h, i, j, k, l, m, and f. Unlike in Theorem 1, the survivor cannot now loop back to e (they might be caught by a zombie) but must instead move to $t = t_i$. Once at t_i, the survivor chooses some s_j where $i \neq j$, and moves there. Next they can move to a_j and start the same walk in Q_j as it did in Q_i. It may continue in this way *ad infinitum*.

Since Q has 69 vertices, we have shown the following.

▶ **Theorem 5.** *Let $k \geq 2$ be an integer. Then there is a polygon P_k with $69k$ vertices whose visibility graph requires at least k zombies.*

A linear number of zombies will always work (e.g. $n/3$ of them starting on every third vertex), so the maximum zombie number of the visibility graph of a polygon with n vertices is $\Theta(n)$.

There is a related problem that asks for the zombie number of a *point-visibility graph* of a polygon, which is the infinite graph G where $V(G)$ is taken to be the *points* of the polygon, not simply the vertices. Edges are then defined as in the visibility graph. This problem involves more geometry than the other problems we have studied. Here it is not clear that there are polygons with a point-visibility graph zombie number higher than one.

4 The Lazy Zombie Number of Outerplanar Graphs is 2

In the previous section, we showed that $\Omega(n)$ zombies are sometimes necessary to catch a survivor on an outerplanar graph. In this section, we show that 2 lazy zombies are always sufficient to catch the survivor on outerplanar graphs. Observe that two lazy zombies are sometimes necessary to catch a survivor on an outerplanar graph since a single lazy zombie cannot win on a 4-cycle.

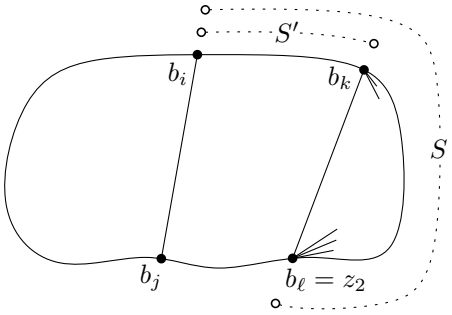
▶ **Theorem 6.** *Let G be a connected outerplanar graph. Then $z_L(G) \leq 2$ and 2 lazy zombies can catch the survivor in less than $2n$ rounds. This bound is tight in the worst case.*

Proof. We only present a proof sketch.

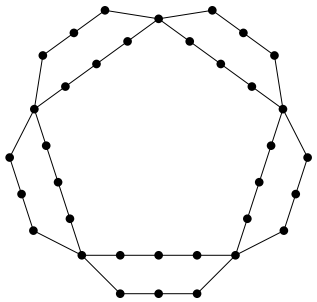
We first modify G by replicating cut vertices and cut edges, and adding chords, so as to make it 2-connected. We start one lazy zombie, the *stationary* lazy zombie, on the end b_j of a chord b_ib_j. This lazy zombie will capture the survivor if the survivor moves to b_i or b_j but otherwise will not move. We refer to the vertices of G where the survivor is known to be restricted as the *survivor territory*. The other lazy zombie (denoted z_2), the *advancing* lazy zombie, also starts at b_j.

Basically, on each turn, the advancing lazy zombie moves along the outerface towards the survivor. Suppose that at some turn, this lazy zombie is at the boundary of survivor territory at a vertex b_l which is on a chord b_lb_k. Assume by symmetry that b_l is counterclockwise of b_j and clockwise of b_i (See Figure 4a). Then, if the survivor is counterclockwise from b_l to b_k (in $S \setminus S'$ in the figure), we switch the roles of the lazy zombies, with a lazy zombie stationary on b_lb_k, and we have reduced the survivor territory. On the other hand, if the survivor is counterclockwise from b_k to b_i (in S' in the figure), the advancing lazy zombie moves to b_k, reducing the survivor territory (to S'). At no point will there be a chord from the survivor territory to the chain from b_j counterclockwise to b_l.

Since each step of the advancing lazy zombie into the survivor territory reduces the survivor territory, and no lazy zombie repeats a move to a vertex, the survivor will be captured in at most $2n$ rounds. ◀



(a) The advancing lazy zombie is on a vertex b_ℓ with a chord (or null chord) whose other end is in S.



(b) An example of a treewidth-2 graph G with $z_L(G) > 2$.

Figure 4

▶ **Corollary 7.** *Let G be a connected outerplanar graph. Then $u_L(G) \leq 2$. This bound is tight in the worst case.*

Outerplanar graphs are a subset of the treewidth-2 graphs, but Theorem 6 cannot be generalized to treewidth-2. Figure 4b shows a graph with treewidth two that requires three lazy zombies. This example shows a distinction between the lazy zombie number and the cop number of a graph since 2 cops are sufficient for a graph of treewidth 2 [16]. We will study general treewidth-k graphs in Section 5.

5 Cut-decomposable Graphs and Lazy Zombies

In this section, we explore the relationship between lazy zombie numbers and various graph parameters. We first define some of these graph parameters and some notation most of which appears in Diestel [11]. We then present the general approach, and finally, we outline some of the consequences of our approach.

Let T be a tree rooted at a vertex r. For a vertex $v \in V(T)$, we denote the unique path from v to r as $\pi_T(v)$. The depth of v, $d_T(v) := d_T(v, r)$, is the length of the path from v to r. If a vertex $u \in V(T)$ is in $\pi_T(v)$ then u is an ancestor of v and v is a descendant of u. Note that v is an ancestor and descendant of itself. The height of T is defined as $H_T := \max\{d_T(v) : v \in V(T)\}$. The subtree of T rooted at v is denoted as $\Lambda_T(v)$. The height of $\Lambda_T(v)$ is defined as $H_T(v) := \max\{d_T(x) - d_T(v) : x \in \Lambda_T(v)\}$. The closure of T, denoted as $clos(T)$, is $T \cup \{uv : u \text{ is an ancestor of } v \text{ in } T\}$. The *treedepth* of a connected graph G, which we denote as $td(G)$, is 1 plus the minimum height H_T over all trees T defined on $V(G)$ such that $G \subseteq clos(T)$.

A *tree decomposition* of a graph G is a pair $(T, \mathcal{B})$ where T is a tree and $\mathcal{B} = \{B_x \subseteq V(G) : x \in V(T)\}$, where each B_x is a subset of $V(G)$ indexed by the nodes of T. The set B_x is sometimes referred to as a *bag*. The following properties must be satisfied:

- For every $v \in V(G)$, $\{x \in V(T) : v \in B_x\}$ induces a non-empty subtree of T.
- For every $uv \in E(G)$, $\exists x \in V(T)$ such that u and v are both in B_x.

The *width* of a tree decomposition is 1 less than the cardinality of the largest bag. The *treewidth* of a graph G, which we denote as $tw(G)$, is the minimum width over all tree decompositions of G.

A *cut decomposition* of a graph G is a pair $(X, \mathcal{C})$ where X is a rooted tree and $\mathcal{C} = \{C_x \subseteq V(G) : x \in V(X)\}$, where each C_x is a subset of $V(G)$ indexed by the nodes of X. The following properties must be satisfied:

- For every $v \in V(G)$, there is a unique $x \in V(X)$ such that $v \in C_x$.
- For every $uv \in E(G)$, $\exists x, y \in V(X)$ such that $u \in C_x$, $v \in C_y$ and x is an ancestor of y in X.
- For each non-leaf node $y \in X$, C_y is a cut-set of $G[Y]$ where $Y = \bigcup_{x \in \Lambda_X(y)} C_x$.

We will refer to the set C_x as the *container* of x to avoid confusion with a bag of a tree decomposition. We will refer to the size of the largest container as the width of the cut decomposition tree, denoted as $cdw(X)$.

Throughout this section, we will assume that $(X, \mathcal{C})$ is a cut decomposition of a graph G where $|V(G)| = n$. We will refer to X as a *cut decomposition tree*. We will refer to the vertices of G as vertices and the vertices of X as nodes, in an attempt to make the distinction clear. We will use u, v to refer to vertices of G and x, y to refer to nodes in X (or nodes in a tree decomposition). For a node $y \in X$, we define the *component* of y to be $G[Y]$ where $Y = \bigcup_{x \in \Lambda_X(y)} C_x$. We slightly abuse notation and refer to the component of y as $G[\Lambda_X(y)]$. Intuitively, a cut decomposition tree is a decomposition of a graph by cuts where an internal node x of the tree X represents a cut set of the graph $G[\Lambda_X(x)]$. The container of the root of the tree contains either the entire vertex set of G, or the vertices of a cut set of G. If it contains a cut, then the children of the root recursively correspond to the different connected components of the graph that result when the cut is removed. Cut decompositions are related to both tree decompositions and treedepth (see Lemma 12).

We define the *load* of a node x in X as:

$$load(x) = \begin{cases} |C_x| & \text{if } x \text{ is a leaf,} \\ |C_x| + \max_y load(y) & \text{otherwise, where } y \text{ is a child of } x. \end{cases}$$

The load of a cut decomposition is defined as the load of the root of the cut decomposition tree. We define the load of a graph G, denoted as $load(G)$, to be the minimum load among all cut decompositions of G. We will show that $load(G)$ is a sufficient number of lazy zombies to catch a survivor in G. We define $time(x)$, where $x \in X$, to help us bound the number of rounds it takes for the lazy zombies to capture the survivor.

$$time(x) = \begin{cases} |C_x|(diam(G) - 1) + 1 & \text{if } x \text{ is a leaf,} \\ \max_y time(y)(|C_x|(diam(G) - 1) + 1) & \text{otherwise, where } y \text{ is a child of } x. \end{cases}$$

The time of the root is an upper bound on the number of rounds it takes the lazy zombies to capture the survivor.

In the following, each lazy zombie z_i may be *assigned* to a vertex v of G. The strategy of the lazy zombie will be the following. If the lazy zombie is not assigned to any vertex, then on its turn to move, it remains at its current location. A lazy zombie assigned to a vertex v has the following behavior: on its turn, it moves off its current vertex u to an adjacent vertex w only if there exists a w that is closer to both v and the survivor. This is precisely where we use the power of a lazy zombie to stand still where regular zombies cannot. Because a shortest path from z_i's location to v has at most $diam(G)$ edges, the survivor can encounter vertex v at most $diam(G) - 1$ times (at or after the time z_i was assigned to v) without being immediately caught. Lazy zombies can and will be reassigned to different vertices during the game.

We proceed by induction. The following lemma establishes the basis.

▶ **Lemma 8.** *Let G be a connected graph with cut decomposition $(X, \mathcal{C})$. Suppose that the survivor is restricted to the vertices of $G[C_x]$ for some leaf x in X. Then, load(x) lazy zombies, starting from anywhere in G, can capture the survivor in at most time(x) rounds.*

▶ **Theorem 9.** *Let G be a connected graph with cut decomposition $(X, \mathcal{C})$. Suppose that x is a node of X and the survivor is restricted to the vertices in $G[\Lambda_X(x)]$. Then, load(x) lazy zombies, starting from anywhere in G, can capture the survivor in at most time(x) rounds.*

Proof. We prove the theorem by induction on $H_X(x)$, the height of $\Lambda_X(x)$. The basis for the induction, $H_X(x) = 0$, i.e. when x is a leaf, follows from Lemma 8.

We assume that *load*(x) lazy zombies are sufficient to catch the survivor in *time*(x) rounds when the survivor is restricted to $G[\Lambda_X(x)]$, where $H_X(x) \leq k$ for $k \geq 0$. We now proceed with the case when $H_X(x) = k + 1$. Let c be the maximum load of a child of x, and d be the maximum time for a child of x. We allocate lazy zombies $z_1, z_2, \ldots, z_c$ to the children of x. These lazy zombies are initially unassigned to any specific vertex but will be assigned to specific vertices depending on the survivor's moves. Note that it is not necessarily the case that we need to use this many lazy zombies, but this number is always sufficient. We assign lazy zombies $z_{c+1}, z_{c+2}, \ldots, z_{c+|C_x|}$, each to a different vertex of C_x, respectively. Since *load*$(x) = |C_x| + c$, we have a sufficient number of lazy zombies.

The survivor may now encounter each vertex of C_x at most $(diam(G) - 1)$ times without immediately being caught in the next round, which again follows from the upper bound of *diam*(G) edges on any shortest path between two vertices in G. Before the survivor's first encounter with a vertex of C_x, or between successive visits of vertices in C_x, or after the last visit, the survivor is restricted to the vertices of the component of the subtree rooted at exactly one child y of x. This follows from the fact that C_x is a cut set for $G[\Lambda_X(x)]$. We apply the inductive hypothesis on $G[\Lambda_X(y)]$, since $H_X(y) \leq k$. By the inductive hypothesis, we know that the survivor is caught after *time*(y) rounds if the survivor remains in $G[\Lambda_X(y)]$. Therefore, the survivor must leave $G[\Lambda_X(y)]$ after *time*$(y) - 1 \leq d - 1$ steps, otherwise it is caught.

Each time it enters one of these subtrees, we assign the lazy zombies $z_1, z_2, \ldots, z_c$ to (specific) vertices in that subtree's component. Since c is the maximum load of any child of x, we have a sufficient number of lazy zombies. By the inductive hypothesis, this number of lazy zombies suffices to either catch the survivor if the survivor remains in the component for d steps or force the survivor out of the component of a child of x and back into C_x in at most $d - 1$ steps.

The survivor's walk length is therefore at most $(|C_x|(diam(G) - 1) + 1) + (d - 1) \times (|C_x|(diam(G) - 1) + 1)$. The first term is the number of rounds the survivor can spend in C_x until it is caught. For the second term, we note that the survivor can enter $G[\Lambda_X(y)]$ where y is a child of x at most $(|C_x|(diam(G) - 1) + 1)$ times. Each time it enters $G[\Lambda_X(y)]$ it must return to a vertex in C_x in $d - 1$ rounds, otherwise the survivor is caught on the dth round. Therefore, we have that the survivor is caught after $(|C_x|(diam(G) - 1) + 1) + (d - 1)(|C_x|(diam(G) - 1) + 1) \leq d((|C_x|(diam(G) - 1)) + 1) = time(x)$. ◀

Among all cut decomposition trees realizing the load of G, we denote the value of the minimum height of such a tree by *cdh*(G). Among all cut decomposition trees whose load is *load*(G) and whose height is *cdh*(G), we denote the value of the minimum width among all such trees by *cdw*(G).

▶ **Corollary 10.** *Given a connected graph G, $u_L(G) \leq load(G)$ and load(G) lazy zombies can catch the survivor in at most $(cdw(G)(diam(G) - 1) + 1)^{cdh(G)+1}$ rounds.*

Proof. We only present a proof sketch.

Let X be a cut decomposition tree with root r, with load *load*(G), with height *cdh*(G) and with width *cdw*(G). Theorem 9 implies that $u_L(G) \leq load(G)$. Using Theorem 9 and the recursive definition of *time*, we show by induction that *time*(r) is at most $(cdw(G)(diam(G) - 1) + 1)^{cdh(G)+1}$. ◀

Recall that $td(G)$ denotes the treedepth of G.

▶ **Theorem 11.** *For any connected graph G, $load(G) = td(G)$*

For $0 < \alpha < 1$, a cut set $S \subseteq V(G)$ of G is an α-separator if every connected component of $G[V(G) - S]$ contains at most αn vertices. The size of the α-separator is the cardinality of S. We highlight the relationship between the sizes of separators, treedepth, and treewidth.

▶ **Lemma 12.** *Let G be a graph. Let $s_G : \{1, \ldots, n\} \to \mathbb{N}$ be a function defined as*

$$s_G(i) = \max_{A \subseteq V(G),\ |A| \leq i} \min\{|S| : S \text{ is a } \tfrac{1}{2}\text{-separator of } G[A]\}.$$

Then $s_G(n) \leq load(G) = td(G) \leq \sum_{i=0}^{\log n} s_G(n/2^i) \leq (tw(G) + 1) \log n$.

Proof. The inequality $s_G(n) \leq td(G) \leq \sum_{i=0}^{\log n} s_G(n/2^i)$ is proven in Lemma 6.6 in [19]. Since it was shown by Robertson and Seymour [22] that $s_G(i) \leq tw(G) + 1$ for all $i \in [1, n]$, we have that $\sum_{i=0}^{\log n} s_G(n/2^i) \leq (tw(G) + 1) \log n$. The equality $load(G) = td(G)$ is proven in Theorem 11. ◀

$s_G(n)$ is sometimes called the separation number of G. The bound in Lemma 12 is tight in certain cases. For example, the treewidth of a path on n vertices is 1 whereas the treedepth is $\Theta(\log n)$. However, for certain classes of graphs, we can remove the $\log n$ term on the upper bound in Lemma 12. Essentially, if $s_G(n/2^i) \leq cs_G(n)/2^i$ for some constant c, then $\sum_{i=0}^{\log n} s_G(n/2^i) \leq cs_G(n) \sum_{i=0}^{\log n} 1/2^i \leq 2cs_G(n) \leq 2c(tw(G) + 1)$. Thus, we have that $td(G)$ is $O(tw(G))$ in this case. Informally, this happens when the size of a separator for any subgraph of size i is at most i^ε, for $0 < \varepsilon < 1$. This is summarized by the following:

▶ **Corollary 13** (Corollary 6.2 in [19]). *Let $0 < \alpha < 1$, let $c > 0$ be a constant and let $\mathcal{G}$ be a hereditary class of graphs such that every $G \in \mathcal{G}$ with n vertices has $tw(G) \leq cn^\alpha$, then every $G \in \mathcal{G}$ has $td(G) \leq \frac{c}{1-2^{-\alpha}} n^\alpha$.*

Treewidth, treedepth and separators are well-studied graph parameters. We highlight a few of the implications of our bound that $u_L(G) \leq td(G)$. The interested reader should consult the following comprehensive surveys on this topic [5, 19, 15, 12]. Using Corollary 13 with separator theorems on various classes of graphs [17, 14, 2], we get the following results.

▶ **Corollary 14.** *For connected planar graphs G, $u_L(G)$ is $O(\sqrt{n})$. For connected genus-g graphs G, $u_L(G)$ is $O(\sqrt{gn})$. For connected graphs G with any excluded h-vertex minor H, $u_L(G)$ is $O(h\sqrt{hn})$. In all these cases, the lazy zombies can catch the survivor in at most $n^{O(\log n)}$ rounds.*

Although $load(G)$ is an upper bound on $u_L(G)$, it is by no means tight. If G is a clique, then $load(G) = n$, but only 1 lazy zombie suffices to catch a survivor. We try to leverage this idea to get a slightly tighter bound.

By assigning one lazy zombie to each clique in a container of the cut decomposition rather than one lazy zombie to each vertex in the container, we can improve the upper bound. This idea leads to an alternative definition of load for a cut decomposition which we call $load^*$. In this definition, for $S \subseteq V(G)$, $\theta(S)$ is the clique cover number of the induced graph of G on the vertices of S.

$$load^*(v) = \begin{cases} \theta(C_v) & \text{if } v \text{ is a leaf,} \\ \theta(C_v) + \max_w load^*(w) & \text{otherwise, where } w \text{ is a child of } v. \end{cases}$$

If C_v is an independent set then $\theta(C_v) = |C_v|$. Thus, without knowing more about the cuts, *load** is no more useful than *load*. However, *load** is a substantial improvement for some graphs. For example, if G is a clique then *load*$^*(G) = 1$ which is optimal. The corresponding notion of time is

$$time^*(v) = \begin{cases} \theta(C_v)diam(G) + 1 & \text{if } v \text{ is a leaf,} \\ (\theta(C_v)diam(G) + 1)\max_w time^*(w) & \text{otherwise, where } w \text{ is a child of } v. \end{cases}$$

▶ **Theorem 15.** *Let G be a connected graph with cut decomposition $(X, \mathcal{C})$. Suppose that v is a vertex of X and the survivor is restricted to the component of v. Then, load$^*(v)$ lazy zombies can capture the survivor in at most time$^*(v)$ rounds.*

▶ **Corollary 16.** *Given a connected graph G, $u_L(G) \leq$ load$^*(G) \leq$ load(G) and load$^*(G)$ lazy zombies can catch the survivor in at most $(\theta(G)(diam(G) + 1))^{cdh(G)+1}$ rounds.*

References

1 M. Aigner and M. Fromme. A game of cops and robbers. *Discret. Appl. Math.*, 8(1):1–12, 1984.
2 N. Alon, P. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In H. Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 293–299. ACM, 1990.
3 T. Andreae. On a pursuit game played on graphs for which a minor is excluded. *J. Comb. Theory, Ser. B*, 41(1):37–47, 1986.
4 V. Bartier, L. Bénéteau, M. Bonamy, H. La, and J. Narboni. A note on deterministic zombies. *Discret. Appl. Math.*, 301:65–68, 2021.
5 H. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.
6 A. Bonato and R. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.
7 P. Bose, J.-L. De Carufel, and T. Shermer. Pursuit-evasion in graphs: Zombies, lazy zombies and a survivor. *CoRR*, abs/2204.11926, 2022. `arXiv:2204.11926`.
8 H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005.
9 N. Clarke. *Constrained cops and robber*. PhD thesis, Dalhousie University, Halifax, Canada, 2002.
10 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.
11 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
12 Z. Dvorák and S. Norin. Treewidth of graphs with balanced separations. *J. Comb. Theory, Ser. B*, 137:137–144, 2019.
13 S. Fitzpatrick, J. Howell, M.-E. Messinger, and D. Pike. A deterministic version of the game of zombies and survivors on graphs. *Discret. Appl. Math.*, 213:1–12, 2016.
14 J. Gilbert, J. Hutchinson, and R. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5(3):391–407, 1984.
15 D. Harvey and D. Wood. Parameters tied to treewidth. *J. Graph Theory*, 84(4):364–385, 2017.
16 G. Joret, M. Kaminski, and D. Theis. The cops and robber game on graphs with forbidden (induced) subgraphs. *Contributions Discret. Math.*, 5(2), 2010.
17 R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

18 T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

19 J. Nesetril and P. Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

20 R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discret. Math.*, 43(2-3):235–239, 1983.

21 A. Quilliot. *Jeux et pointes fixes sur les graphes*. PhD thesis, Université de Paris VI, Paris, France, 1978.

22 N. Robertson and P. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

23 B. Schröder. The copnumber of a graph is bounded by [3/2 genus (g)] + 3. In J. Koslowski and A. Melton, editors, *Categorical Perspectives*, pages 243–263, Boston, MA, 2001. Birkhäuser Boston.

Range Updates and Range Sum Queries on Multidimensional Points with Monoid Weights

Shangqi Lu
Chinese University of Hong Kong, New Territories, Hong Kong

Yufei Tao
Chinese University of Hong Kong, New Territories, Hong Kong

Abstract

Let P be a set of n points in $\mathbb{R}^d$ where each point $p \in P$ carries a *weight* drawn from a commutative monoid $(\mathcal{M}, +, 0)$. Given a d-rectangle r_{upd} (i.e., an orthogonal rectangle in $\mathbb{R}^d$) and a value $\Delta \in \mathcal{M}$, a *range update* adds Δ to the weight of every point $p \in P \cap r_{\text{upd}}$; given a d-rectangle r_{qry}, a *range sum query* returns the total weight of the points in $P \cap r_{\text{qry}}$. The goal is to store P in a structure to support updates and queries with attractive performance guarantees. We describe a structure of $\tilde{O}(n)$ space that handles an update in $\tilde{O}(T_{\text{upd}})$ time and a query in $\tilde{O}(T_{\text{qry}})$ time for arbitrary functions $T_{\text{upd}}(n)$ and $T_{\text{qry}}(n)$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. The result holds for any fixed dimensionality $d \geq 2$. Our query-update tradeoff is tight up to a polylog factor subject to the OMv-conjecture.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Range Updates, Range Sum Queries, Data Structures, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.57

Funding This research was supported in part by GRF Projects 14207820 and 14203421 from HKRGC.

1 Introduction

This paper studies range sum queries on multidimensional points where the point weights are drawn from a commutative monoid and can be modified by range updates. Specifically, let P be a set of n points in $\mathbb{R}^d$ for some constant $d \geq 1$. Denote by $(\mathcal{M}, +, 0)$ an arbitrary commutative monoid[1] where each element in $\mathcal{M}$ is called a *weight*. Each point $p \in P$ carries a weight $w(p) \in \mathcal{M}$; initially, the weights are 0 for all the points. We want to store P in a data structure to support two operations with attractive performance guarantees:

- *Range (sum) query:* given a d-rectangle[2] r_{qry}, the query returns the total weight of all the points $p \in P \cap r_{\text{qry}}$ (where sum is defined using the monoid's operator $+$);
- *Range update:* given a d-rectangle r_{upd} and a weight $\Delta \in \mathcal{M}$, the update adds Δ to the weight of every point $p \in P \cap r_{\text{upd}}$.

We will refer to the above as *the "range sum with range updates"* (RSRU) *problem*. Our complexity analysis assumes the standard unit-cost RAM model and holds on all commutative monoids $(\mathcal{M}, +, 0)$ satisfying: (i) each weight $w \in \mathcal{M}$ can be stored in one word, and (ii) $w_1 + w_2$ can be computed in constant time for any $w_1, w_2 \in \mathcal{M}$.

[1] A commutative monoid $(\mathcal{M}, +, 0)$ is defined by a set $\mathcal{M}$, an operator $+$: $\mathcal{M} \times \mathcal{M} \to \mathcal{M}$ obeying associativity and commutativity, and an identity element $0 \in \mathcal{M}$ satisfying $0 + w = w$ for every $w \in \mathcal{M}$.
[2] Defined as $[a_1, b_1] \times ... \times [a_d, b_d]$.

1.1 Previous Results

Supporting range queries and range updates has important implications in geographical information systems (GIS), online analytical processing (OLAP), and database management systems (DBMS); the reader may refer to [16, 19, 22, 24] for the relevant applications.

For $d = 1$, the RSRU problem admits a folklore structure[3] of $O(n)$ space that supports each query and update in $O(\log n)$ time. The problems become rather challenging as soon as d reaches 2. For any $d \geq 2$, the standard *range tree* [2, 10] uses $\tilde{O}(n)$ space and answers a query in $\tilde{O}(1)$ time (throughout the paper, the notation $\tilde{O}(.)$ suppresses a polylog n factor). It also supports a "point update" – an update whose rectangle r_{upd} degenerates into a point – in $\tilde{O}(1)$ time. Given an update with an arbitrary r_{upd}, however, the range tree issues a point update for each $p \in P \cap r_{\text{upd}}$ and thus can incur a cost of $\tilde{O}(n)$.

For $d \geq 2$, Lau and Ritossa [19] developed an $O(n)$-space structure that supports each query and update in $\tilde{O}(n^{1-1/d})$ time. They also showed a connection to the *OMv-conjecture* [12], which has been widely utilized to characterize the hardness of problems involving dynamic data structures [1, 3–9, 11, 13–15, 17, 18, 20, 21, 23]:

> In *online matrix-vector multiplication* (OMv), an algorithm A is allowed to preprocess an $n \times n$ boolean matrix $\mathbf{M}$ in poly(n) time and then, in the online phase, needs to compute $\mathbf{M}\boldsymbol{v}_i$ for $n \times 1$ boolean vectors $\boldsymbol{v}_1, ..., \boldsymbol{v}_n$ (additions and multiplications are as in the boolean semi-ring). The vectors are supplied in succession, i.e., $\boldsymbol{v}_{i+1}$ arrives only after A has output $\mathbf{M}\boldsymbol{v}_i$. The *cost* of A is the total time it spends in the online phase. The *OMv-conjecture* states that no algorithm can guarantee a cost of $O(n^{3-\delta})$ no matter how small the constant $\delta > 0$ is.

For $d = 2$, Lau and Ritossa [19] proved that, subject to the OMv-conjecture, no structure with update time T_{upd} and query time T_{qry} can guarantee $\max\{T_{\text{upd}}, T_{\text{qry}}\} = O(n^{1/2-\delta})$, regardless of the constant $\delta > 0$. Hence, their aforementioned structure can no longer be improved significantly in 2D space.

The results of [19] leave two intriguing questions. First, the hardness result does not shed much light on the *tradeoff* between T_{upd} and T_{qry}. For example, if we insist on $T_{\text{qry}} = \tilde{O}(1)$, is it possible to improve the update cost $\tilde{O}(n)$ of the range tree by a polynomial factor? Conversely, if T_{upd} must be $\tilde{O}(1)$, what is the best query time achievable? As yet another example, can we hope to obtain $T_{\text{upd}} = \tilde{O}(n^{0.5})$ and $T_{\text{qry}} = \tilde{O}(n^{0.49})$, thereby improving *only* the query time of [19] polynomially? The second question concerns the scenario of $d \geq 3$, where there remains a large gap between the upper and (conditional) lower bounds of [19]. We will answer all these questions in this paper.

The RSRU problem has a degenerated *array* version that has received special attention. In that version, $P := [m]^d$ where $m \geq 1$ is an integer (given an integer $x \geq 1$, $[x]$ represents the set $\{1, 2, ..., x\}$). In other words, P has exactly $n = m^d$ points, and each point's coordinate is an integer in $[m]$ on every dimension; equivalently, P can be regarded as a d-dimensional array. This RSRU variant can be settled by a structure of $O(n)$ space that supports a query and an update both in $O(\log^{d+1} n)$ time [24]. Furthermore, if the monoid is multiplicative[4], the query and update time can be reduced to $O(\log^d n)$ [24]; see also [16, 22] for (array-RSRU) structures designed for the monoid $(\mathbb{R}, +, 0)$ (that is, each weight is a real value).

---

[3] `https://cp-algorithms.com/data_structures/segment_tree.html`.

[4] A monoid $(\mathcal{M}, +, 0)$ is *multiplicative* if, for any weight $w \in \mathcal{M}$ and any integer $c \geq 1$, $c \cdot w := \underbrace{w + w + ... + w}_{c}$ can be calculated in constant time.

Table 1 A comparison of our and previous results on the RSRU problem.

Space	Update, Query	Ref	Remark
$\tilde{O}(n)$	$\tilde{O}(n), \tilde{O}(1)$	[2]	$d \geq 2$
$O(n)$	$\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n})$	[19]	$d = 2$
$O(n)$	$\tilde{O}(n^{1-1/d}), \tilde{O}(n^{1-1/d})$	[19]	$d \geq 3$
$\tilde{O}(n)$	any $\tilde{O}(T_{\text{upd}})$, $\tilde{O}(T_{\text{qry}})$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$	this paper	$d \geq 2$
–	$\max\{T_{\text{upd}}, T_{\text{qry}}\} = O(n^{1/2-\delta})$ impossible	[19]	monoid $(\mathbb{R}, +, 0)$, $d = 2$ cond. on OMv-conjecture
–	$O(n^a), O(n^b)$ with $a + b < 1$ impossible (a, b are constants)	this paper	monoid $(\mathbb{R}, +, 0)$, $d = 2$ cond. on OMv-conjecture

1.2 New Results

For the RSRU problem, we establish a smooth trade-off between the update and query time under fixed dimensions $d \geq 2$:

▶ **Theorem 1.** *For the RSRU problem, there is a structure of $\tilde{O}(n)$ space that supports an update in $\tilde{O}(T_{\text{upd}})$ time and a query in $\tilde{O}(T_{\text{qry}})$ time for arbitrary functions $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. The result holds for any constant dimension $d \geq 2$.*

By setting $T_{\text{upd}} = T_{\text{qry}} = \sqrt{n}$, we obtain a structure of $\tilde{O}(n)$ space that handles an update/query in $\tilde{O}(\sqrt{n})$ time for any d. Compared to [19], for $d = 2$ we obtain the same update and query time (up to a polylog factor), whereas for $d \geq 3$ our update and query time is better by a polynomial factor. The theorem, interestingly, also captures the range tree as a special case with $T_{\text{upd}} = n$ and $T_{\text{qry}} = 1$. By adjusting T_{upd} and T_{qry}, one can obtain a series of structures with different update-query tradeoffs that were not known previously. Our structures are drastically different from the ones in [19] and do not deteriorate with d (ignoring polylog factors).

We further prove that Theorem 1 is nearly tight subject to the OMv-conjecture.

▶ **Theorem 2.** *Consider the RSRU problem defined on $d = 2$ and the monoid $(\mathbb{R}, +, 0)$. Fix any constant c satisfying $0 \leq c < 1$ and an arbitrarily small constant $\delta > 0$. Subject to the OMv-conjecture, the following holds for any structure constructible in $\text{poly}(n)$ time:*

- *if the update time $T_{\text{upd}} = O(n^c)$, then the query time T_{qry} cannot be $O(n^{1-c-\delta})$;*
- *if $T_{\text{qry}} = O(n^c)$, then T_{upd} cannot be $O(n^{1-c-\delta})$.*

The above clearly implies the impossibility of $\max\{T_{\text{upd}}, T_{\text{qry}}\} = O(n^{1/2-\delta})$, as was already proved in [19]. On the other hand, our conditional lower bounds are much more informative; for example, they reveal, somewhat unexpectedly, the range tree – with $T_{\text{qry}} = \tilde{O}(1)$ and $T_{\text{upd}} = \tilde{O}(n)$ – can no longer be improved significantly without breaking the OMv-conjecture. Putting together Theorems 1 and 2, we now have a complete picture on the query-update tradeoff achievable for the RSRU problem under any fixed dimension up to a sub-polynomial factor. Table 1 summarizes the comparison of our and previous results.

1.3 New Techniques

Our structures stem from a new observation on the inherent characteristics of the RSRU problem. The observation, described below, is interesting in its own right and illustrates what separates the RSRU problem from its array variant (defined in Section 1.1).

For any point $p \in \mathbb{R}^d$, we use $p[i]$ ($i \in [d]$) to represent its coordinate on dimension i. Similarly, given a d-rectangle $r := [a_1, b_1] \times ... \times [a_d, b_d]$, we use $r[i]$ to represent its i-th projection $[a_i, b_i]$. Given a subset $S \subseteq [d]$, we define an *S-rectangle* r as a d-rectangle where $r[i] := (-\infty, \infty)$ for every $i \in [d] \setminus S$, namely, r can have a bounded range $r[i]$ only on the dimensions $i \in S$.

Given an update with rectangle r_{upd} and some weight, we call it a *U-update* for some $U \subseteq [d]$ if r_{upd} is a U-rectangle. Likewise, given a query with rectangle r_{qry}, we call it a *Q-query* for some $Q \subseteq [d]$ if r_{qry} is a Q-rectangle.

▶ **Definition 3.** *Fix two (possibly overlapping) subsets U and Q of $[d]$.* A (U, Q)-structure *is a structure that supports only U-updates and Q-queries.*

Our objective in the RSRU problem is to design a $([d], [d])$-structure. We are now ready to state our characteristic observation:

▶ **Theorem 4.** *For the RSRU problem, suppose that, given any disjoint $U \subseteq [d]$ and $Q \subseteq [d]$, there is a (U, Q)-structure of $\tilde{O}(n)$ space that guarantees update time T_{upd} and query time T_{qry}. Then, there is a $([d], [d])$-structure of $\tilde{O}(n)$ space that handles an update in $O(T_{\text{upd}} \cdot \log^d n)$ time and a query in $O(T_{\text{qry}} \cdot \log^d n)$ time.*

The theorem indicates that the core of RSRU lies in dealing with updates and queries that concern *disjoint* sets of dimensions. For example, in 2D space, the core boils down to supporting $U = \{1\}$ and $Q = \{2\}$, namely, every update rectangle r_{upd} is a vertical slab while every query rectangle r_{qry} is a horizontal slab. Interestingly, this is precisely what separates general RSRU from its array variant. As we will see, when P is a 2D array, there is a trivial (U, Q)-structure of $O(1)$ space ensuring $O(\log n)$ update and query time (the time can even be reduced to $O(1)$ if the monoid is multiplicative); in contrast, when P is a generic set of Euclidean points, the hardness in Theorem 2 applies!

Theorem 4 has yet another notable implication: it "trivializes" the array version of RSRU and allows us to recover all the existing results from [16, 22, 24] (reviewed in Section 1.1) with a simple structure. The details can be found in Appendix A.

2 A Dimension Elimination Technique

This section is devoted to proving Theorem 4. Our strategy is to incrementally remove a common dimension of U and Q until the two dimension sets become disjoint, at which point we can apply the U-Q disjoint structure stated in the theorem's assumption statement. The core is to establish the following lemma.

▶ **Lemma 5.** *Consider any overlapping subsets U and Q of $[d]$. Let $i \in [d]$ be an arbitrary dimension in $U \cap Q$. Suppose that we have a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure both of which use $O(n \log^c n)$ space (where $c \geq 0$ is a constant) and support an update in $O(T_{\text{upd}})$ time and a query in $O(T_{\text{qry}})$ time. Then, there is a (U, Q)-structure of $O(n \log^{c+1} n)$ space that handles an update in $O(T_{\text{upd}} \log n)$ time and a query in $O(T_{\text{qry}} \log n)$ time.*

Before proving the lemma, let us first see how it leads to Theorem 4.

Proof of Theorem 4. We will establish a more general claim: fix any integer $k \in [0, d]$; for any subsets U and Q of $[d]$ such that $|U \cap Q| = k$, there is a (U, Q)-structure of $\tilde{O}(n)$ space that guarantees update and query time $O(T_{\text{upd}} \log^k n)$ and $O(T_{\text{qry}} \log^k n)$, respectively. When $k = 0$, U and Q are disjoint and the claim directly follows from the theorem's assumption. Next, we will prove the claim for $k = k_0 + 1$, assuming the claim's correctness on $k = k_0 \geq 0$.

Identify an arbitrary $i \in U \cap Q$; i must exist because $|U \cap Q| = k_0 + 1 \geq 1$. By the inductive assumption, there exist a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure, both of which use $\tilde{O}(n)$ space and ensure update time $O(T_{\text{upd}} \log^{k_0} n)$ and query time $O(T_{\text{qry}} \log^{k_0} n)$. We now apply Lemma 5 to obtain a (U, Q)-structure of $\tilde{O}(n)$ space with update and query time $O(T_{\text{upd}} \log^{k_0+1} n)$ and $O(T_{\text{qry}} \log^{k_0+1} n)$ time, respectively. This completes the proof. ◀

The rest of the section serves as a proof of Lemma 5. Section 2.1 will describe our structure as well as the update and query algorithms. Section 2.2 will present our analysis.

Basic Notations and Concepts. Let U and Q be the dimension sets in Lemma 5. Assume, w.l.o.g., that the value i in the lemma is 1, i.e., $1 \in U \cap Q$. For convenience, we will refer to dimension 1 as the "x-dimension". Accordingly, given a point $p \in \mathbb{R}^d$, its "x-coordinate" is $p[1]$. We will represent an update as (r_{upd}, Δ), where r_{upd} is a d-rectangle and Δ is a weight in $\mathcal{M}$; recall that the update adds Δ to the weight of every point $p \in P \cap r_{\text{upd}}$. We will use $r_{\text{upd}}[2:d]$ to denote the projection of r_{upd} onto dimensions $2, 3, ..., d$, namely, $r_{\text{upd}}[2:d]$ is a $(d-1)$-dimensional rectangle.

Given a set S of n real values, a binary search tree (BST) on S is a binary tree $\mathcal{T}$ such that (i) $\mathcal{T}$ has height $O(\log n)$, (ii) $\mathcal{T}$ has n leaves each storing a different value in S as its *key*, (iii) every internal node has two children, (iv) for each internal node, the elements of S in its left subtree are strictly less than those in its right subtree, and (v) each internal node stores a *key*, which is the smallest element of S in its right subtree. For each leaf/internal node u, denote its key as $key(u)$. The parent of a non-root node u is represented as $parent(u)$ and the root of $\mathcal{T}$ as $root(\mathcal{T})$.

We associate each node u of $\mathcal{T}$ with a *slab* $\sigma(u)$ defined recursively as follows. If $u = root(\mathcal{T})$, then $\sigma(u) := (-\infty, \infty)$. Otherwise, let $v := parent(u)$. If u is the left child of v, $\sigma(u) := \sigma(v) \cap (-\infty, key(v))$; otherwise, $\sigma(u) := \sigma(v) \cap [key(v), \infty)$. Slabs have several easy-to-verify properties:

- If node v is an ancestor of node u, then $\sigma(u) \subseteq \sigma(v)$.
- If u and v have no ancestor-descendant relationships, then $\sigma(u)$ and $\sigma(v)$ are disjoint.
- For each node u, $\sigma(u) \cap S$ is the set of elements stored in the subtree of u.

2.1 Structure and Algorithms

Denote by S the set of *distinct* x-coordinates of the points in P. Build a BST $\mathcal{T}$ on S. For each node u of $\mathcal{T}$, define

$$P_u \quad := \quad \{p \in P \mid p[1] \in \sigma(u)\}$$

namely, the set of points $p \in P$ whose x-coordinates are in the slab $\sigma(u)$ of u. We associate each u with a $(U \setminus \{1\}, Q)$-structure and a $(U, Q \setminus \{1\})$-structure both constructed on P_u. Recall that the two structures are already available by the assumption of Lemma 5. We will call each of them a *secondary structure* on P_u. This completes the description of our (U, Q)-structure.

Each $p \in P$ is in $O(\log n)$ secondary structures. For each secondary structure Υ, define

$$\text{weight of } p \text{ in } \Upsilon \quad := \sum_{(r_{\text{upd}}, \Delta) \in \mathcal{U}_\Upsilon : p \in r_{\text{upd}}} \Delta$$

where $\mathcal{U}_\Upsilon$ is the set of updates[5] ever performed on Υ.

[5] More specifically, each update $(r_{\text{upd}}, \Delta) \in \mathcal{U}$ should be treated as a pair with an id because two updates can have the same (r_{upd}, Δ).

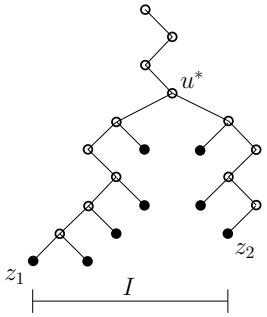


Figure 1 White dots are the internal path nodes of I and black dots are the canonical nodes of I.

Canonical and Internal Path Nodes of an Interval. To pave the way for our discussion, next we define what are the canonical and internal path nodes of an interval $I := [x_1, x_2]$, where both x_1 and x_2 belong to S. Let z_1 and z_2 be the leaves whose keys equal x_1 and x_2, respectively. Denote by π_1 (resp., π_2) the path from $root(\mathcal{T})$ to z_1 (resp., z_2).

- We call u an *internal path node* of I if u is an internal node on π_1 or π_2.
- We call u a *canonical node* of I if
  - $u = z_1$ or z_2, or
  - $parent(u)$ is in $\pi_1 \cup \pi_2$, u itself is not in $\pi_1 \cup \pi_2$, and $\sigma(u)$ is covered by I.

Let $\mathcal{C}_I$ be the set of canonical nodes of I. We must have $|\mathcal{C}_I| = O(\log n)$.

As another way to understand $\mathcal{C}_I$, one can first identify the lowest node $u^* \in \pi_1 \cap \pi_2$ (this is the node where π_1 and π_2 diverge). If u^* is a leaf, it means $\pi_1 = \pi_2$ and u^* is the only node in $\mathcal{C}_I$. Now consider the case where u^* is an internal node. Let us descend the path π_1' from u^* to z_1. Every time we descend into the left child of a node $v \neq u^*$ on π_1', we add to $\mathcal{C}_I$ the right child of v (nothing is added if we descend into the right child of v). Perform also a symmetric process for the path from u^* to z_2. The $\mathcal{C}_I$ at this moment contains all the canonical nodes. See Figure 1 for an illustration.

Update Algorithm. Consider a U-update (r_{upd}, Δ) on our (U, Q)-structure (remember the structure only needs to support U-updates). W.o.l.g., assume that the x-range of r_{upd} has the form $[x_1, x_2]$ where both x_1 and x_2 belong to S.[6] We carry out the update using the following algorithm.

`update` (r_{upd}, Δ)
1. $I_{\text{upd}} \leftarrow r_{\text{upd}}[1]$ /* the x-range of r_{upd} */
2. $r'_{\text{upd}} \leftarrow (-\infty, \infty) \times r_{\text{upd}}[2:d]$ /* r'_{upd} replaces the x-range with $(-\infty, \infty)$ */
3. **for** each internal path node u of I_{upd} **do**
4. perform an update (r_{upd}, Δ) on the $(U, Q \setminus \{1\})$-structure of P_u
5. **for** each canonical node u of I_{upd} **do**
6. perform an update $(r'_{\text{upd}}, \Delta)$ on the $(U \setminus \{1\}, Q)$-structure of P_u

It is worth pointing out that r'_{upd} is a $U \setminus \{1\}$-rectangle. Hence, the update $(r'_{\text{upd}}, \Delta)$ at Line 6 is permitted on the $(U \setminus \{1\}, Q)$-structure of P_u. See Figure 2(a) for an illustration.

[6] This assumption can be easily fulfilled by performing predecessor/successor search in $O(\log n)$ time.

▶ **Proposition 6.** *Let Υ be a structure updated at Line 4 or 6 of* `update`*. Suppose that it is a secondary structure of P_u. For each $p \in P_u$, its weight in Υ increases by Δ if and only if $p \in r_{\text{upd}}$.*

Proof. This is obvious if Υ is a $(U, Q \setminus \{1\})$-structure of P_u (Line 4). Consider, instead, Υ as a $(U \setminus \{1\}, Q)$-structure of P_u (Line 6). It follows that u is a canonical node of I_{upd} and hence $p[1] \in I_{\text{upd}}$. By the assumption of Lemma 5, Υ increases the weight of p if and only if $p \in r'_{\text{upd}}$. Our claim holds because $p \in r'_{\text{upd}}$ if and only if $p \in r_{\text{upd}}$. ◀

Query Algorithm. Consider a Q-query with search rectangle r_{qry} on our (U, Q)-structure. W.o.l.g., we assume that the x-range of r_{qry} has the form $[x_1, x_2]$ where both x_1 and x_2 belong to S. Our query algorithm is shown below.

`query` (r_{qry})
1. $I_{\text{qry}} \leftarrow r_{\text{qry}}[1]$; $r'_{\text{qry}} \leftarrow (-\infty, \infty) \times r_{\text{qry}}[2:d]$
2. OUT $\leftarrow 0$
3. **for** each internal path node u of I_{qry} **do**
4. OUT $\leftarrow$ OUT + output of the query r_{qry} on the $(U \setminus \{1\}, Q)$-structure of P_u
5. **for** each canonical node u of I_{qry} **do**
6. OUT $\leftarrow$ OUT + output of the query r'_{qry} on the $(U \setminus \{1\}, Q)$-structure of P_u
7. OUT $\leftarrow$ OUT + output of the query r'_{qry} on the $(U, Q \setminus \{1\})$-structure of P_u
8. **return** OUT

The reader should note that r'_{qry} is a $Q \setminus \{1\}$-rectangle and hence also a Q-rectangle. Therefore, the queries at Lines 6 and 7 are permitted. See Figure 2(b) for an illustration.

▶ **Proposition 7.** *Let Υ be a structure searched at Line 4, 6, or 7 of* `query`*. Suppose that it is a secondary structure of P_u. For each $p \in P_u$, its weight in Υ is added into* OUT *if and only if $p \in r_{\text{qry}}$.*

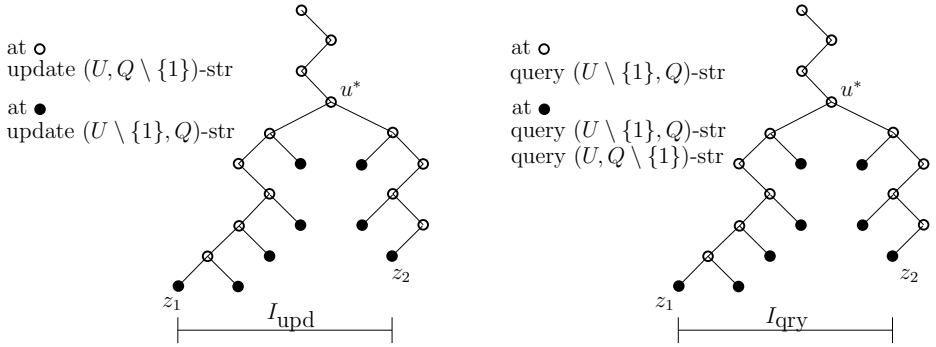
Proof. This is obvious if Υ is a $(U \setminus \{1\}, Q)$-structure at Line 4. If Υ is a $(U \setminus \{1\}, Q)$-structure at Line 6 or a $(U, Q \setminus \{1\})$-structure at Line 7, u must be a canonical node of I_{qry} and hence $p[1] \in I_{\text{qry}}$. By the assumption of Lemma 5, when Υ is searched with r'_{qry}, its output incorporates the weight of p if and only if $p \in r'_{\text{qry}}$. Our claim holds because $p \in r'_{\text{qry}}$ if and only if $p \in r_{\text{qry}}$. ◀

2.2 Analysis

Space and Time Complexities. The update time and query time are clearly $O(T_{\text{upd}} \log n)$ and $O(T_{\text{qry}} \log n)$, respectively. The secondary structures of a node u in $\mathcal{T}$ occupy space $O(|P_u| \log^c n)$. As each point $p \in P$ appears in the P_u of $O(\log n)$ nodes u, the total space of our (U, Q)-structure is $O(n \log^{c+1} n)$.

Correctness. It remains to prove that all queries are answered correctly. Let us start with a concept crucial for our argument: update atom. Formally, each update (r_{upd}, Δ) generates an *atom* $(r_{\text{upd}}, \Delta, p)$ for every $p \in P \cap r_{\text{upd}}$. The atom describes the fact that the update should increase $w(p)$ by Δ. Conceptually, the effect of (r_{upd}, Δ) is achieved by "executing" all of its atoms.

Given a query with search rectangle r_{qry}, we will show that the output OUT of algorithm `query` is exactly $\sum_{p \in P \cap r_{\text{qry}}} w(p)$. Define



(a) Update (b) Query

Figure 2 Illustration of the update and query algorithms.

- $\mathcal{U}$ as the set of updates that have ever been performed on our (U, Q)-structure;
- $\mathcal{A}$ as the collection of atoms generated by the updates in $\mathcal{U}$.

Each atom $(r_{\text{upd}}, \Delta, p) \in \mathcal{A}$ is said to be *relevant* if $p \in r_{\text{qry}}$. For each $p \in P$, it holds that

$$w(p) = \sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}} \Delta$$

which yields

$$\sum_{p \in P \cap r_{\text{qry}}} w(p) = \sum_{p \in P \cap r_{\text{qry}}} \left(\sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}} \Delta \right) = \sum_{\text{relevant } (r_{\text{upd}}, \Delta, p) \in \mathcal{A}} \Delta. \tag{1}$$

Let Υ be a secondary structure searched at Line 4, 6, or 7 of `query`(r_{qry}). Denote by u the node that Υ is associated with. Define:

- $\mathcal{U}_\Upsilon$ as the set of updates $(r_{\text{upd}}, \Delta) \in \mathcal{U}$ such that algorithm `update`(r_{upd}, Δ) modifies Υ at either Line 4 or 6;
- $\mathcal{A}_\Upsilon$ as the collection of atoms $(r_{\text{upd}}, \Delta, p)$ generated by the updates in $\mathcal{U}_\Upsilon$ satisfying $p \in P_u$.

We will refer to $\mathcal{A}_\Upsilon$ as the *atom set* of Υ. By Proposition 6, it holds for each point $p \in P_u$:

$$\text{weight of } p \text{ in } \Upsilon \quad := \sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta.$$

By Proposition 7, when searched in algorithm `query`(r_{qry}), Υ returns:

$$\sum_{p \in P_u \cap r_{\text{qry}}} \text{weight of } p \text{ in } \Upsilon = \sum_{p \in P \cap r_{\text{qry}}} \left(\sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta \right) = \sum_{\text{relevant } (r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta.$$

It follows from the above discussion that

$$\text{OUT} \quad = \sum_{\text{searched } \Upsilon} \left(\sum_{\text{relevant } (r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta \right). \tag{2}$$

Our mission is to draw equivalence between (1) and (2). We achieve the purpose with the following lemma.

▶ **Lemma 8.** *Every relevant atom $(r_{\text{upd}}, \Delta, p) \in \mathcal{A}$ appears in the atom set $\mathcal{A}_\Upsilon$ of exactly one secondary structure Υ searched by* `query`(r_{qry}).

Proof. Consider any relevant atom $(r_{\text{upd}}, \Delta, p) \in \mathcal{A}$. Let $I_{\text{qry}} := r_{\text{qry}}[1]$. By definition of relevance, $p \in r_{\text{qry}}$. Among the canonical nodes of I_{qry}, there is exactly one node – denoted as u_{qry} – satisfying the condition that $p[1]$ falls in the slab $\sigma(u_{\text{qry}})$ of u_{qry}. Similarly, let $I_{\text{upd}} := r_{\text{upd}}[1]$. By definition of atom, $p \in r_{\text{upd}}$. Among the canonical nodes of I_{upd}, there is exactly one node – denoted as u_{upd} – satisfying $p[1] \in \sigma(u_{\text{upd}})$. Nodes u_{qry} and u_{upd} must have an ancestor-descendant relationship.

Fix a secondary structure Υ searched by $\texttt{query}(r_{\text{qry}})$ (at Line 4, 6, or 7). The next two facts follow from how $\texttt{update}(r_{\text{upd}}, \Delta)$ and $\texttt{query}(r_{\text{qry}})$ execute (as illustrated in Figure 2).

Fact 1. Suppose that Υ is the $(U \setminus \{1\}, Q)$-structure of node v. Then, $(r_{\text{upd}}, \Delta, p)$ appears in $\mathcal{A}_\Upsilon$ if and only if

- $v = u_{\text{upd}}$, and
- v is an ancestor of u_{qry} (this includes the case $v = u_{\text{qry}}$).

Fact 2. Suppose that Υ is the $(U, Q \setminus \{1\})$-structure of v. Then, $(r_{\text{upd}}, \Delta, p)$ appears in $\mathcal{A}_\Upsilon$ if and only if

- $v = u_{\text{qry}}$, and
- v is an internal path node of I_{upd}.

We proceed by discussing two cases separately:

Case 1: u_{upd} is a proper descendant of u_{qry}. Atom $(r_{\text{upd}}, \Delta, p)$ cannot belong to the atom set of any $(U \setminus \{1\}, Q)$-structure Υ searched by $\texttt{query}(r_{\text{qry}})$. Otherwise, Υ must be associated with u_{upd} (first bullet of Fact 1), but then the second bullet of Fact 1 contradicts u_{upd} being a proper descendant of u_{qry}. On the other hand, as a proper ancestor of u_{upd}, u_{qry} must be an internal path node of I_{upd}. Fact 2 thus shows that $(r_{\text{upd}}, \Delta, p)$ exists in the atom set of only one $(U, Q \setminus \{1\})$-structure searched by $\texttt{query}(r_{\text{qry}})$: the one at node u_{qry}.

Case 2: u_{upd} is an ancestor of u_{qry}. Atom $(r_{\text{upd}}, \Delta, p)$ cannot belong to the atom set of any $(U, Q \setminus \{1\})$-structure Υ searched by $\texttt{query}(r_{\text{qry}})$. To see why, suppose that such a Υ exists. By Fact 2, Υ must be associated with node u_{qry}, and u_{qry} must be an internal path node of I_{upd}. This is impossible because u_{upd} (being a canonical node of I_{upd}) cannot have any descendant that is an internal path node of I_{upd}. Finally, Fact 1 shows that $(r_{\text{upd}}, \Delta, p)$ appears in the atom set of only one $(U \setminus \{1\}, Q)$-structure searched by $\texttt{query}(r_{\text{qry}})$: the one at node u_{upd}. ◀

This completes the proof of Lemma 5.

3 U-Q Disjoint Structures

Equipped with Theorem 4, we can now concentrate on designing (U, Q)-structures with disjoint U and Q. We will prove:

▶ **Lemma 9.** *Fix an integer $k \geq 1$ and consider the RSRU problem under dimensionality $d = k$. Suppose that, for any disjoint $U, Q \subseteq [d]$, there is a (U, Q)-structure of $\tilde{O}(n)$ space supporting an update in $\tilde{O}(T_{\text{upd}})$ time and a query in $\tilde{O}(T_{\text{qry}})$ time for any functions $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. Then, the following holds for dimensionality $d = k + 1$: for any disjoint $U, Q \subseteq [d]$, we can build a (U, Q)-structure of $\tilde{O}(n)$ space supporting an update in $\tilde{O}(T_{\text{upd}})$ and a query in $\tilde{O}(T_{\text{qry}})$ time for any functions $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$.*

Before delving into the proof, let us see how the lemma leads to Theorem 1.

Proof of Theorem 1. At $d = 1$, it is easy to obtain a $([1], [1])$-structure of $O(n)$ space and $O(\log n) = \tilde{O}(1)$ update and query time (see Section 1.1). The structure can serve as the basis solution for $k = 1$ and any $T_{\text{upd}}(n) \geq 1, T_{\text{qry}}(n) \geq 1$ with $T_{\text{upd}} \cdot T_{\text{qry}} = n$. Lemma 9 then asserts that, for any constant d and any disjoint $U, Q \subseteq [d]$, we can build a (U, Q)-structure that uses $\tilde{O}(n)$ space and handles an update in $\tilde{O}(T_{\text{upd}})$ and a query in $\tilde{O}(T_{\text{qry}})$ time for any $T_{\text{upd}}(n) \geq 1, T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. Combining this with Theorem 4 establishes Theorem 1. ◀

The rest of the subsection serves as a proof of Lemma 9. Let us first eliminate the case of $U = \emptyset$. In this scenario, the rectangle r_{upd} of an update is fixed to $\mathbb{R}^d$ and hence all points in P have the same weight. It suffices to maintain the $w(p^*)$ of an arbitrary $p^* \in P$. In addition, build a standard *range count* structure on P such that uses $\tilde{O}(n)$ space and, given a rectangle r_{qry}, outputs $|P \cap r_{\text{qry}}|$ in $\tilde{O}(1)$ time; the range tree [10] fulfills our purpose here. To answer a query with rectangle r_{qry}, we first obtain $c := |P \cap r_{\text{qry}}|$ and then return $c \cdot w(p^*)$. The query time is $\tilde{O}(1)$, noticing that $c \cdot w(p^*)$ can be calculated in $O(\log c)$ time[7].

Next, we assume $U \neq \emptyset$ and, w.l.o.g., consider that (i) U contains the x-dimension (i.e., dimension 1), (ii) $n := |P|$ is a power of two, and (iii) the points in P have distinct coordinates on each dimension. Fix any $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$.

Structure. We will describe a binary tree $\mathcal{T}$ of $O(\log T_{\text{qry}})$ levels and $O(T_{\text{qry}})$ nodes. Each node u in $\mathcal{T}$ is associated with a subset $P_u \subseteq P$ and an interval $\sigma(u)$ as its slab. If $u = \mathit{root}(\mathcal{T})$, $P_u := P$ and $\sigma(u) := (-\infty, \infty)$. In general, if $|P_u| \leq T_{\text{upd}}$, u is a leaf of $\mathcal{T}$. Otherwise, we split P_u evenly into P_1 and P_2 at some value x such that P_1 (resp., P_2) includes all the points of P_u whose x-coordinates are less (resp., greater) than x. The left and right children of u are associated with P_1 and P_2, respectively, and have slab $\sigma(u) \cap (-\infty, x)$ and $\sigma(u) \cap [x, \infty)$, respectively. The total number of nodes in $\mathcal{T}$ is $O(n/T_{\text{upd}}) = O(T_{\text{qry}})$.

Each internal node u in $\mathcal{T}$ is associated with a $(U \setminus \{1\}, Q)$-structure $\mathcal{T}_u$ on P_u. Since $(U \setminus \{1\}) \cap Q = \emptyset$ and $|(U \setminus \{1\}) \cup Q| \leq k$, we already know how to construct such a structure (see the assumption of Lemma 9). We parameterize $\mathcal{T}_u$ such that it supports an update on P_u in $\tilde{O}(T_{\text{upd}})$ time and answers a query on P_u in $\tilde{O}(|P_u|/T_{\text{upd}})$ time; its space is $\tilde{O}(|P_u|)$.

For each leaf z in $\mathcal{T}$, create a range tree $\mathcal{T}_z$ on P_z. As discussed in Section 1.1, $\mathcal{T}_z$ uses $\tilde{O}(|P_z|)$ space, answers a query on P_z in $\tilde{O}(1)$ time, and supports an update on P_z in $\tilde{O}(|P_z|) = \tilde{O}(T_{\text{upd}})$ time.

Each $p \in P$ appears in $O(\log T_{\text{qry}})$ secondary structures Υ. For every such Υ, define

$$\text{weight of } p \text{ in } \Upsilon \quad := \sum_{(r_{\text{upd}}, \Delta) \in \mathcal{U}_\Upsilon : p \in r_{\text{upd}}} \Delta$$

where $\mathcal{U}_\Upsilon$ is the set of updates ever performed on Υ.

Non-path Canonical Nodes and Path Leaves of an Interval. We now adapt the concepts "canonical" and "path nodes" from Section 2.1 to our context here. Consider an interval $I := [x_1, x_2]$. Let z_1 and z_2 be the leaves of $\mathcal{T}$ such that $x_1 \in \sigma(z_1)$ and $x_2 \in \sigma(z_2)$. Denote by π_1 (resp., π_2) the path from $\mathit{root}(\mathcal{T})$ to z_1 (resp., z_2).

[7] E.g., $15w = w + 2w + 4w + 8w$, where $4w$ (resp. $8w$) can be derived from $2w$ (resp. $4w$) in constant time.

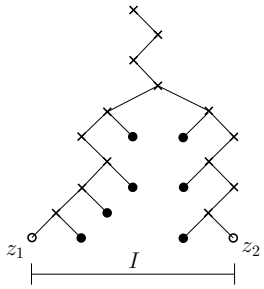


Figure 3 White dots are the path leaves of I and black dots are the non-path canonical nodes.

- We call each of z_1 and z_2 a *path leaf* of I.
- We call u a *non-path canonical node* of I if *parent*(u) is in $\pi_1 \cup \pi_2$, u itself is not in $\pi_1 \cup \pi_2$, and $\sigma(u)$ is covered by I.

See Figure 3 for an illustration.

Update. Consider an update (r_{upd}, Δ). Define $I_{\text{upd}} := r_{\text{upd}}[1]$ and $r'_{\text{upd}} := (-\infty, \infty) \times r_{\text{upd}}[2:d]$. At each non-path canonical node u of I_{upd}, perform an update $(r'_{\text{upd}}, \Delta)$ on $\mathcal{T}_u$. At each path leaf z of I_{upd}, perform an update (r_{upd}, Δ) on $\mathcal{T}_z$.

Query. Given a query with rectangle r_{qry}, we simply access every node u in $\mathcal{T}$ and issue a query with the same rectangle r_{qry} on the secondary structure $\mathcal{T}_u$. Then, we return the sum of the weights returned by those structures.

Analysis. It should have become straightforward that our structure uses $\tilde{O}(n)$ space overall and supports an update in $\tilde{O}(T_{\text{upd}})$ time. Next, we analyze the query time. As $\mathcal{T}$ has $O(T_{\text{qry}})$ leaves and a query spends $\tilde{O}(1)$ time on each leaf, the time spent on all the leaves is $\tilde{O}(T_{\text{qry}})$. Let us now attend to the internal nodes. Consider the i-th level of $\mathcal{T}$.[8]. There are $O(2^i)$ internal nodes and $|P_u| = O(n/2^i)$ for every such node u. The time spent on all the level-i nodes is $\tilde{O}(2^i \cdot (n/2^i)/T_{\text{upd}}) = \tilde{O}(n/T_{\text{upd}}) = \tilde{O}(T_{\text{qry}})$. As $\mathcal{T}$ has $\tilde{O}(1)$ levels, the overall query cost is $\tilde{O}(T_{\text{qry}})$.

It remains to show the correctness of our $(k+1)$-dimensional structure. For this purpose, let us first observe:

▶ **Proposition 10.** *For any $p \in P$, $w(p) = \sum_{\text{node } u \text{ in } \mathcal{T}: p \in P_u}$(weight of p in $\mathcal{T}_u$).*

Proof. The proposition obviously holds after the structure has just been constructed. Consider an update (r_{upd}, Δ). Define $I_{\text{upd}} := r_{\text{upd}}[1]$. Denote by z_1, z_2 the two path leaves of I_{upd} and by $\mathcal{C}$ the set of non-path canonical nodes of I_{upd}. It is easy to verify:

- for any distinct nodes u, v in $\{z_1, z_2\} \cup \mathcal{C}$, P_u and P_v are disjoint;
- $\bigcup_{u \in \{z_1, z_2\} \cup \mathcal{C}} (P_u \cap r_{\text{upd}}) = P \cap r_{\text{upd}}$.

For each point $p \in P \cap r_{\text{upd}}$, there is a unique node $u \in \{z_1, z_2\} \cup \mathcal{C}$ satisfying $p \in P_u$. Our update procedure increases the weight of p in $\mathcal{T}_u$ by Δ and does not change its weight in any other secondary structure. On the other hand, if $p \notin r_{\text{upd}}$, the procedure will not change its weight in any secondary structure. Therefore, if the proposition holds before the update, it still does afterwards. ◀

[8] The root is at level 0 and the level number increases by 1 each time we descend into a child.

Fix any query with rectangle r_{qry}. For each node u in $\mathcal{T}$, denote by OUT_u the answer returned by the structure $\mathcal{T}_u$. The value OUT_u equals $\sum_{p \in P_u \cap r_{\text{qry}}}$ (weight of p in $\mathcal{T}_u$). The final answer returned is

$$\sum_{\text{node } u \text{ in } \mathcal{T}} \sum_{p \in P_u \cap r_{\text{qry}}} \text{weight of } p \text{ in } \mathcal{T}_u = \sum_{p \in P \cap r_{\text{qry}}} \Big(\sum_{\text{node } u \text{ in } \mathcal{T}: p \in P_u} \text{weight of } p \text{ in } \mathcal{T}_u \Big)$$
$$= \sum_{p \in P \cap r_{\text{qry}}} w(p)$$

where the last equality used Proposition 10. With this, we have established the correctness of our structure and thus conclude the proof of Lemma 9.

4 Hardness of RSRU

This section will establish Theorem 2. Let us first review the *γ-uMv problem* from [13]:

> Fix a constant $\gamma > 0$, and choose two integers n_1 and n_2 satisfying $n_1 = \lfloor n_2^\gamma \rfloor$. In the *$\gamma$-uMv problem*, an algorithm A is allowed to preprocess an $n_1 \times n_2$ boolean matrix $\mathbf{M}$ in $\text{poly}(n_1, n_2)$ time, after which A receives a $1 \times n_1$ boolean vector $\boldsymbol{u}$ and an $n_2 \times 1$ boolean vector $\boldsymbol{v}$, and needs to compute $\boldsymbol{u}\mathbf{M}\boldsymbol{v}$ (additions and multiplications are as in the boolean semi-ring). The *cost* of A is the time it spends on computing $\boldsymbol{u}\mathbf{M}\boldsymbol{v}$.

The following result is due to Henzinger et al. [13]:

▶ **Lemma 11** ([13]). *Fix an arbitrary constant $\gamma > 0$. Subject to the OMv-Conjecture, no algorithm can solve the γ-uMv problem with cost $O(n_1^{1-\delta} \cdot n_2 + n_1 \cdot n_2^{1-\delta})$, no matter how small the constant $\delta > 0$ is.*

Given an RSRU structure defying Theorem 2, we will show how to utilize it to develop an algorithm to beat Lemma 11. We use $\mathbf{M}[i, j]$ to denote the entry of $\mathbf{M}$ at the i-th row and j-th column, $\boldsymbol{u}[i]$ to denote the i-th component of $\boldsymbol{u}$, and $\boldsymbol{v}[j]$ to denote the j-th component of $\boldsymbol{v}$, where $i \in [n_1]$ and $j \in [n_2]$.

Proof of the First Bullet of Theorem 2. Consider the RSRU problem under $d = 2$ and monoid $(\mathbb{R}, +, 0)$ and let constants $c \in [0, 1)$ and $\delta > 0$ be chosen as in Theorem 2. Define $U := \{1\}$ and $Q := \{2\}$. We will prove that, subject to the OMv-conjecture, no (U, Q)-structure constructible in $\text{poly}(n)$ time can guarantee update time $O(n^c)$ and query time $O(n^{1-c-\delta})$. This will imply the first bullet of the theorem.

Assume that such a structure Υ exists. Set $\gamma := \frac{1-c-\delta/2}{c+\delta/2}$. Next, we will describe an algorithm for the γ-uMv problem. In preprocessing, we create a set P of 2D points as follows: P has a point (i, j) if and only if $\mathbf{M}[i, j] = 1$ for each $i \in [n_1]$ and $j \in [n_2]$. Initialize $w(p) := 0$ for all $p \in P$ and then create a (U, Q)-structure Υ on P. The preprocessing time is $\text{poly}(n_1, n_2)$ because $|P| \leq n_1 \cdot n_2$. Given vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, we compute $\boldsymbol{u}\mathbf{M}\boldsymbol{v}$ by issuing at most n_1 U-updates and at most n_2 Q-queries. For each $i \in [n_1]$, if $\boldsymbol{u}[i] = 1$, we perform an update with rectangle $(r_{\text{upd}}, 1)$ with $r_{\text{upd}} := [i, i] \times (-\infty, \infty)$ on P, which effectively adds 1 to the weight of every point $p \in P$ satisfying $p[1] = i$. Then, for each $j \in [n_2]$, if $\boldsymbol{v}[j] = 1$, we perform a query with $r_{\text{qry}} := (-\infty, \infty) \times [j, j]$ on P, which effectively checks whether any point $p \in P$ with $p[2] = j$ has a positive $w(p)$. The reader can verify that $\boldsymbol{u}\mathbf{M}\boldsymbol{v} = 1$ if and only if at least one of the queries returns a non-zero value.

To analyze the cost, set $\lambda := n_2^{1/(c+\delta/2)}$. As $n_1 = \lfloor n_2^\gamma \rfloor$, we have $n_1 = \Theta(\lambda^{1-c-\delta/2})$ and $n_2 = \Theta(\lambda^{c+\delta/2})$. The number of points in P is $O(n_1 \cdot n_2) = O(\lambda)$; hence, Υ ensures update time $O(\lambda^c)$ and query time $O(\lambda^{1-c-\delta})$. As the algorithm performs at most n_1 updates and at most n_2 queries, the total cost is

$$O(n_1 \cdot \lambda^c + n_2 \cdot \lambda^{1-c-\delta}) = O(\lambda^{1-\delta/2}) = O((n_1 \cdot n_2)^{1-\delta/2})$$

where the last step used $\lambda = \Theta(n_1 \cdot n_2)$. This contradicts Lemma 11.

Proof of the Second Bullet of Theorem 2. As before, define $U := \{1\}$ and $Q := \{2\}$. We will prove that, subject to the OMv-conjecture, no (U, Q)-structure constructible in poly(n) time can guarantee update time $O(n^{1-c-\delta})$ and query time $O(n^c)$. This will imply the second bullet of the theorem.

Assume that such a structure exists. We deploy it to tackle γ-uMv in the same way as before where $\gamma := \frac{c+\delta/2}{1-c-\delta/2}$. To analyze the cost, set $\lambda := n_2^{1/(1-c-\delta/2)}$. As $n_1 = \lfloor n_2^\gamma \rfloor$, we have $n_1 = \Theta(\lambda^{c+\delta/2})$, $n_2 = \Theta(\lambda^{1-c-\delta/2})$, and $|P| = O(n_1 \cdot n_2) = O(\lambda)$. The structure handles an update and query in $O(\lambda^{1-c-\delta})$ and $O(\lambda^c)$ time, respectively. Because at most n_1 updates and at most n_2 queries are performed, our algorithm's cost is $O(n_1 \cdot \lambda^{1-c-\delta} + n_2 \cdot \lambda^c) = O(\lambda^{1-\delta/2}) = O((n_1 \cdot n_2)^{1-\delta/2})$, contradicting Lemma 11.

▶ Remark. We can extend the above lower bound to any monoid $(\mathcal{M}, +, 0)$ as long as there is a value $e^* \in \mathcal{M}$ satisfying $\sum_{i=1}^{c} e^* \neq 0$ for any $c \in [1, n]$. The only modification is in the online phase: for each $i \in [n_1]$ with $u[i] = 1$, add e^* (rather than 1) to $w(p)$ for all the points $p \in P$ satisfying $p[1] = i$. Then, we have $\boldsymbol{uMv} = 1$ if and only if at least one of the at most n_2 queries defined as before returns a non-zero value.

References

1. Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 477–486, 2016.
2. Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters (IPL)*, 8(5):244–251, 1979.
3. Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1836–1855, 2021.
4. Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 303–318, 2017.
5. Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering ucqs under updates and in the presence of integrity constraints. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:19, 2018.
6. Christoph Berkholz and Maximilian Merz. Probabilistic databases under updates: Boolean query evaluation and ranked enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 402–415, 2021.
7. Katrin Casel and Markus L. Schmid. Fine-grained complexity of regular path queries. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 19:1–19:20, 2021.
8. Raphaël Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya. Upper and lower bounds for dynamic data structures on strings. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 22:1–22:14, 2018.

9 Soren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 48:1–48:14, 2016.

10 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

11 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 153–166, 2020.

12 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2015.

13 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. *CoRR*, abs/1511.06773, 2015. `arXiv:1511.06773`.

14 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In *Innovations in Theoretical Computer Science (ITCS)*, pages 26:1–26:31, 2017.

15 Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. *Electronic Colloquium on Computational Complexity*, page 157, 2021.

16 Nabil Ibtehaz, M. Kaykobad, and M. Sohel Rahman. Multidimensional segment trees can do range updates in poly-logarithmic time. *Theoretical Computer Science*, 854:30–43, 2021.

17 Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Maintaining triangle queries under updates. *ACM Transactions on Database Systems (TODS)*, 45(3):11:1–11:46, 2020.

18 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 375–392, 2020.

19 Joshua Lau and Angus Ritossa. Algorithms and hardness for multidimensional range updates and queries. In *Innovations in Theoretical Computer Science (ITCS)*, pages 35:1–35:20, 2021.

20 Hung Le, Lazar Milenkovic, Shay Solomon, and Virginia Vassilevska Williams. Dynamic matching algorithms under vertex updates. In *Innovations in Theoretical Computer Science (ITCS)*, pages 96:1–96:24, 2022.

21 Shangqi Lu and Yufei Tao. Towards optimal dynamic indexes for approximate (and exact) triangle counting. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 6:1–6:23, 2021.

22 Pushkar Mishra. On updating and querying sub-arrays of multidimensional arrays. *CoRR*, abs/1311.6093, 2013. `arXiv:1311.6093`.

23 Yufei Tao and Ke Yi. Intersection joins under updates. *Journal of Computer and System Sciences (JCSS)*, 124:41–64, 2022.

24 Jason Yang and Jun Wan. On updating and querying submatrices. *CoRR*, abs/2010.13180, 2020. `arXiv:2010.13180`.

A A Simpler Structure for the Array Variant of RSRU

Henceforth, we will focus on the array version of RSRU, defined in Section 1.1, where P is a d-dimensional array $[m]^d$ for some integer $m \geq 1$ (as a result, $n = m^d$). Our goal is to show:

▶ **Theorem 12.** *For the array variant of RSRU, there is a structure of $O(n)$ space that supports each query and update in $O(\log^{d+1} n)$ time. The query and update complexities can be improved to $O(\log^d n)$ if the underlying monoid is multiplicative.*

Recall that a monoid $(\mathcal{M}, +, 0)$ is *multiplicative* if $c \cdot w := \underbrace{w + w + ... + w}_{c}$ can be calculated in constant time for any weight $w \in \mathcal{M}$ and any integer $c \geq 1$. The monoid $(\mathbb{R}, +, 0)$ studied in [16, 22] is multiplicative; hence, the theorem subsumes the results in [16, 22] (reviewed in Section 1.1). For arbitrary commutative monoids, the extra $O(\log n)$ factor arises from the need to compute a multiplication $c \cdot w$ in $O(\log c)$ time; the integer c never exceeds n in our algorithms. In [24], Yang and Wan claimed a structure with query and update time $O(\log^d n)$, but a careful look at their definition reveals that their monoid is multiplicative; for non-multiplicative monoids, their query and update time both slow down by an $O(\log n)$ factor. Hence, Theorem 12 recovers the result of [24] as well. Our structures are drastically different from those in [16, 22, 24].

A.1 The Counterpart of Theorem 4

The characteristics of RSRU revealed by Theorem 4 extend to the array version as well:

▶ **Theorem 13.** *For the array variant of RSRU, suppose that, given any disjoint $U \subseteq [d]$ and $Q \subseteq [d]$, there is a (U, Q)-structure of $O(1)$ space that guarantees update time T_{upd} and query time T_{qry}. Then, there is a $([d], [d])$-structure of $O(n)$ space that handles an update in $O(T_{\text{upd}} \cdot \log^d n)$ time and a query in $O(T_{\text{qry}} \cdot \log^d n)$ time.*

To prove the theorem, we need the lemma below that echoes Lemma 5.

▶ **Lemma 14.** *Consider any two overlapping subsets U and Q of $[d]$. Let $i \in [d]$ be an arbitrary dimension in $U \cap Q$. Suppose that we have a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure both of which use $O(m^{|U \cap Q|-1})$ space and support an update in $O(T_{\text{upd}})$ and a query in $O(T_{\text{qry}})$ time. Then, there is a (U, Q)-structure of $O(m^{|U \cap Q|})$ space that handles an update in $O(T_{\text{upd}} \log n)$ time and a query in $O(T_{\text{qry}} \log n)$ time.*

Proof. Due to symmetry, we assume $i = 1$. Let S be the set of *distinct* x-coordinates of the points in P. $|S| = m$ because P is an array. We use the same reduction in the proof Lemma 5 to obtain a (U, Q)-structure. Recall that $\mathcal{T}$ is a BST on S and $P_u := \{p \in P \mid p[1] \in \sigma(u)\}$ for every node u in $\mathcal{T}$. Associate each u with a $(U \setminus \{1\}, Q)$-structure and a $(U, Q \setminus \{1\})$-structure both constructed on P_u. The update and query algorithms require no changes and finish in $O(T_{\text{upd}} \log n)$ and $O(T_{\text{qry}} \log n)$ time, respectively. Since $\mathcal{T}$ has $O(m)$ nodes and the space at each node is $O(m^{|U \cap Q|-1})$, the total space is $O(m^{|U \cap Q|})$. ◀

Equipped with the above lemma, we will now prove a general claim: fix any integer $k \in [0, d]$; for any subsets U and Q of $[d]$ such that $|U \cap Q| = k$, there is a (U, Q)-structure of $O(m^k)$ space that guarantees update and query time $O(T_{\text{upd}} \log^k n)$ and $O(T_{\text{qry}} \log^k n)$, respectively. Theorem 13 then follows because $m^d = n$.

When $k = 0$, U and Q are disjoint and the claim holds from the theorem's assumption. Next, we will prove the claim for $k = k_0 + 1$, assuming the claim's correctness on $k = k_0 \geq 0$. Fix an arbitrary $i \in U \cap Q$. By the inductive assumption, there exist a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure, both of which use $O(m^{k_0})$ space and ensure update and query time $O(T_{\text{upd}} \log^{k_0} n)$ and $O(T_{\text{qry}} \log^{k_0} n)$ time, respectively. We now apply Lemma 14 to obtain a (U, Q)-structure of $O(m^{k_0+1})$ space with update and query time $O(T_{\text{upd}} \log^{k_0+1} n)$ and $O(T_{\text{qry}} \log^{k_0+1} n)$ time, respectively. This completes the proof.

A.2 U-Q Disjoint Structures

Since P is a d-dimensional array $[m]^d$, henceforth, we consider only d-rectangles of the form $[a_1, b_1] \times ... \times [a_d, b_d]$, where $a_i \in [m]$ and $b_i \in [m]$ for all $i \in [d]$. Accordingly, a U-rectangle is redefined as a d-rectangle r satisfying $r[i] = [1, m]$ for every $i \in [d] \setminus U$, and similarly, a Q-rectangle r is a d-rectangle satisfying $r[i] = [1, m]$ for every $i \in [d] \setminus Q$.

We will show:

▶ **Lemma 15.** *Consider the array version of RSRU. For any disjoint $U \subseteq [d]$ and $Q \subseteq [d]$, there is a (U, Q)-structure of $O(1)$ space that supports an update and a query in $O(\log n)$ time. The update and query time can be improved to $O(1)$ if the underlying monoid $(\mathcal{M}, +, 0)$ is multiplicative.*

Combining Theorem 13 with the above lemma establishes Theorem 12. The rest of the subsection serves as a proof of Lemma 15.

Case 1: $Q = \emptyset$. In other words, the query rectangle r_{qry} always covers the whole $[m]^d$. It suffices to maintain the total weight of all the points: $s := \sum_{p \in P} w(p)$. A query obviously can be settled in $O(1)$ time. Given an update (r_{upd}, Δ), we first calculate the number c of points in P covered by r_{upd}. As P is a multidimensional array, this can be done in $O(1)$ time because $c = \prod_{i \in [d]} |r_{\text{upd}}[i] \cap [m]|$.[9] Then, we increase s by $c \cdot \Delta$, which takes $O(\log n)$ time, or $O(1)$ time if the monoid is multiplicative.

Case 2: $Q \neq \emptyset$. W.o.l.g., we will assume $Q = [\ell]$ for some integer $\ell \in [1, d]$; hence, $U \subseteq [\ell + 1, d]$. Given an *$\ell$-tuple* $t := (x_1, x_2, ..., x_\ell) \in [m]^\ell$, let $P(t) := \{t\} \times [m]^{d-\ell}$, i.e., the set of points $p \in P$ satisfying $p[i] = x_i$ for all $i \in [\ell]$. Define

$$w(t) := \sum_{p \in P(t)} w(p).$$

▶ **Proposition 16.** *For any ℓ-tuples t and t', it always holds that $w(t) = w(t')$.*

Proof. Consider any update (r_{upd}, Δ). As r_{upd} is a U-rectangle, $r_{\text{upd}}[i] = [1, m]$ for each $i \in [\ell]$. The number c of points in $P(t) \cap r_{\text{upd}}$ is $\prod_{i \in [\ell+1,d]} |r_{\text{upd}}[i] \cap [m]|$. Likewise, $|P(t') \cap r_{\text{upd}}| = \prod_{i \in [\ell+1,d]} |r_{\text{upd}}[i] \cap [m]| = c$. Hence, both $w(t)$ and $w(t')$ will increase by $c \cdot \Delta$ after the update. The claim follows because $w(t) = w(t') = 0$ in the beginning (i.e., before the first update). ◀

Our structure simply maintains the $w(t^*)$ for an arbitrary ℓ-tuple t^*. Given a Q-query with rectangle r_{qry}, we first obtain in constant time the number c_1 of ℓ-tuples $t := (x_1, ..., x_\ell)$ satisfying $x_i \in r_{\text{qry}}[i]$ for every $i \in [\ell]$.[10] By Proposition 16 and the fact $r_{\text{qry}}[i] = [1, m]$ for every $i \in [\ell + 1, d]$ (r_{qry} is a Q-rectangle), the query answer is exactly $c_1 \cdot w(t^*)$, which can be computed in $O(\log n)$ time. Given an update (r_{upd}, Δ), we obtain in constant time the number c_2 of points in $P(t^*)$ covered by the U-rectangle r_{upd},[11] and then increase $w(t^*)$ by $c_2 \cdot \Delta$ in $O(\log n)$ time. Both the update and query time can be reduced to $O(1)$ if the monoid is multiplicative.

This completes the proof of Lemma 15.

---

[9] If $r_{\text{upd}}[i] = [a_i, b_i]$, then $|r_{\text{upd}}[i] \cap [m]| = b_i - a_i + 1$.
[10] $c_1 = \prod_{i \in [\ell]} |r_{\text{qry}}[i] \cap [m]|$.
[11] $c_2 = \prod_{i \in [\ell+1,d]} |r_{\text{upd}}[i] \cap [m]|$.

Segment Visibility Counting Queries in Polygons

Kevin Buchin
Department of Computer Science, TU Dortmund, Germany

Bram Custers
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Ivor van der Hoog
Department of Applied Mathematics and Computer Science, DTU, Copenhagen, Denmark

Maarten Löffler
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Aleksandr Popov
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Marcel Roeloffzen
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

Let P be a simple polygon with n vertices, and let A be a set of m points or line segments inside P. We develop data structures that can efficiently count the objects from A that are visible to a query point or a query segment. Our main aim is to obtain fast, $\mathcal{O}(\text{polylog}\, nm)$, query times, while using as little space as possible.

In case the query is a single point, a simple visibility-polygon-based solution achieves $\mathcal{O}(\log nm)$ query time using $\mathcal{O}(nm^2)$ space. In case A also contains only points, we present a smaller, $\mathcal{O}(n + m^{2+\varepsilon} \log n)$-space, data structure based on a hierarchical decomposition of the polygon.

Building on these results, we tackle the case where the query is a line segment and A contains only points. The main complication here is that the segment may intersect multiple regions of the polygon decomposition, and that a point may see multiple such pieces. Despite these issues, we show how to achieve $\mathcal{O}(\log n \log nm)$ query time using only $\mathcal{O}(nm^{2+\varepsilon} + n^2)$ space. Finally, we show that we can even handle the case where the objects in A are segments with the same bounds.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Visibility, Data Structure, Polygons, Complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.58

Related Version *Full Version*: `https://arxiv.org/abs/2201.03490` [7]

Funding *Bram Custers*: Supported by the Dutch Research Council (NWO) under the project number 628.011.005.
Ivor van der Hoog: Supported by the Dutch Research Council (NWO) under the project number 614.001.504.
Maarten Löffler: Partially supported by the Dutch Research Council (NWO) under the project numbers 614.001.504 and 628.011.005.
Aleksandr Popov: Supported by the Dutch Research Council (NWO) under the project number 612.001.801.
Marcel Roeloffzen: Supported by the Dutch Research Council (NWO) under the project number 628.011.005.

1 Introduction

Let P be a simple polygon with n vertices, and let A be a set of m points or line segments inside P. We develop efficient data structures for *visibility counting queries* in which we wish to report the number of objects from A visible to some (constant-complexity) query object Q. An object X in A is *visible* from Q if there is a line segment connecting X and Q contained in P; other objects in A do not block visibility. We focus on the case when Q is a point or a line segment. We aim to obtain fast, $\mathcal{O}(\text{polylog}\, nm)$, query times, using as little space as possible. Our work is motivated by problems in movement analysis where we have sets of moving entities, for example, an animal species and their predators, and we wish to determine if there is mutual visibility between the entities of different sets. We also want to quantify "how much" the sets can see each other. Given measurements at certain points in time, solving the mutual visibility problem between two such times reduces to counting visibility between line segments (for moving entities) and points (for static objects). This visibility counting problem is also of general interest.

Related work. Computing visibility in polygons is a classical problem in computational geometry [19, 30]. Algorithms for efficiently testing visibility between a pair of points, for computing visibility polygons [18, 27, 29], and for constructing visibility graphs [31] have been a topic of study for over thirty years. There is even a host of work on computing visibility on terrains and in other three-dimensional environments [20, Chapter 33.8]. For many of these problems, the data structure version of the problem has also been considered. In these versions, the polygon is given up front, and the task is to store it so that we can efficiently query whether a pair of points p, q is mutually visible [13, 23, 26], or report the visibility polygon $V(q)$ of q [4]. In particular, when P is a simple polygon with n vertices, the former type of queries can be answered optimally – in $\mathcal{O}(\log n)$ time using linear space [26]. The latter type of queries can be answered in $\mathcal{O}(\log^2 n + |V(q)|)$ time using $\mathcal{O}(n^2)$ space [4]. The visibility polygon itself has complexity $\mathcal{O}(n)$ [18]. Visibility polygons inherit structure from the boundaries of P, so the approaches that use it do not transfer to our setting.

Computing the visibility polygon of a line segment has been considered, as well. When the polygon modelling the environment is simple, the visibility polygon, called a *weak visibility polygon,* denoted $V(\overline{pq})$ for a line segment $\overline{pq}$, still has linear complexity, and can be computed in $\mathcal{O}(n)$ time [23]. Chen and Wang [15] consider the data structure version of the problem: they describe a linear-space data structure that can be queried in $\mathcal{O}(|V(\overline{pq})| \log n)$ time, and an $\mathcal{O}(n^3)$-space data structure that can be queried in $\mathcal{O}(\log n + |V(\overline{pq})|)$ time.

Computing the visibility polygon of a line segment $\overline{pq}$ allows us to answer whether an entity moving along $\overline{pq}$ can see a particular fixed point r, i.e. there is a time at which the moving entity can see r if and only if r lies inside $V(\overline{pq})$. If the point r may also move, it is not necessarily true that the entity can see r if the trajectory of r intersects $V(\overline{pq})$. Eades et al. [17] present data structures that can answer such queries efficiently. In particular, they present data structures of size $\mathcal{O}(n \log^5 n)$ that can answer such a query in time $\mathcal{O}(n^{3/4} \log^3 n)$. They present results even in case the polygon has holes. Aronov et al. [4] show that we can also efficiently maintain the visibility polygon of an entity as it is moving.

Visibility counting queries have been studied before, as well. Bose et al. [6] studied the case where, for a simple polygon and a query point, the number of visible polygon edges is reported. The same problem has been considered for weak visibility from a query segment [8]. For the case of a set of disjoint line segments and a query point, approximation algorithms

Table 1 Results in this paper. • and / denote points and line segments, respectively.

A	Q	Data structure			Section
		Space	Preprocessing	Query	
•	•	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm \log n + nm^2)$	$\mathcal{O}(\log nm)$	3.1
•	•	$\mathcal{O}(n+m^{2+\varepsilon} \log n)$	$\mathcal{O}(n + m \log^2 n + m^{2+\varepsilon} \log n)$	$\mathcal{O}(\log n \log nm)$	3.2
/	•	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm \log n + nm^2)$	$\mathcal{O}(\log nm)$	3.1
•	/	$\mathcal{O}(n^2 + nm^{2+\varepsilon})$	$\mathcal{O}(n^2 \log m + nm^{2+\varepsilon})$	$\mathcal{O}(\log n \log nm)$	4
/	/	$\mathcal{O}(n^2 + nm^{2+\varepsilon})$	$\mathcal{O}(n^2 \log m + nm^{2+\varepsilon})$	$\mathcal{O}(\log n \log nm)$	5

exist [3, 21, 32]. In contrast to these settings, we wish to count visible line segments with visibility obstructed by a simple polygon (other than the line segments). Closer to our setting is the problem of reporting all pairs of visible points in a simple polygon [5].

Results and organisation. Our goal is to efficiently count the objects, in particular, line segments or points, in a set A that are visible to a query object Q. We denote this count by $C(Q, A)$. Given P, A, and Q, we can easily compute $C(Q, A)$ in optimal $\mathcal{O}(n + m \log n)$ time (see Lemma 1). We are mostly interested in the data structure version of the problem, in which we are given the polygon P and the set A in advance, and we wish to compute $C(Q, A)$ efficiently once we are given the query object Q. We show that we can indeed answer such queries efficiently, that is, in polylogarithmic time in n and m. The exact query times and the space usage and preprocessing times depend on the type of the query object and the type of objects in A. See Table 1 for an overview. Here and in the rest of the paper, $\varepsilon > 0$ denotes an arbitrarily small constant.

In Section 3, we consider the case where the query object is a point. We show how to answer queries efficiently using the arrangement of all (weak) visibility polygons. As Bose et al. [6, Section 6.2] argued, such an arrangement has complexity $\Theta(nm^2)$ in the worst case. We then show that if the objects in A are points, we can do significantly better. We argue that we do not need to construct the visibility polygons of all points in A, avoiding an $\mathcal{O}(nm)$ term in the space and preprocessing time. We use a hierarchical decomposition of the polygon and the fact that the visibility of a point $a \in A$ in a subpolygon into another subpolygon is described by a single constant-complexity cone. Aronov et al. [4] also use hierarchical decomposition, but the rest of their approach cannot be efficiently used in our setting, since it uses the cyclic ordering of the vertices of a visibility polygon.

In Section 4, we turn our attention to the case where the query object is a line segment $\overline{pq}$ and the objects in A are points. One possible solution in this scenario would be to store the visibility polygons for the points in A so that we can count such polygons stabbed by the query segment. However, since these visibility polygons have total complexity $\mathcal{O}(nm)$ and the query may have an arbitrary orientation, a solution achieving polylogarithmic query time will likely use at least $\Omega(n^2m^2)$ space [1, 2, 24]. So, we again use an approach that hierarchically decomposes the polygon to limit the space usage. Unfortunately, testing visibility between the points in A and the query segment is more complicated in this case. Moreover, the segment can intersect multiple regions of the decomposition, so we have to avoid double counting. All of this makes the problem significantly harder. We manage to overcome these difficulties using careful geometric arguments and an inclusion–exclusion-style counting scheme. Our result in Table 1 saves at least a linear factor compared to an approach based on stabbing visibility polygons. We then show that we can extend these arguments even further and solve the scenario where the objects in A are also line segments. Surprisingly, this does not impact the space or time complexity of the data structure.

Finally, in Section 5, we discuss some extensions of our results. In particular, we show that just testing if the count $C(Q, A)$ is non-zero (i.e. if Q is visible from any of the objects) is easier, and that we can compute the pairwise visibility of two sets of objects – that is, solve the problem that motivated this work – in time subquadratic in the number of objects.

2 Preliminaries

In this section, we briefly review some basic tools we use to build our data structures. We omit some common material and proofs; we refer the reader to the full version for those.

Visibility in a simple polygon. Denote the (weak) visibility polygon in a simple polygon P of a point p (resp. line segment $\overline{pq}$) by $V(p)$ (resp. $V(\overline{pq})$). A *cone* is a subset of the plane that is enclosed by two rays starting at some point p, called the *apex* of the cone; the angle between any two rays in the cone is acute. We refer to the two bounding rays as the *left* and the *right* ray, so that moving clockwise from the left to the right ray traverses the cone. A *subcone* of some cone C is a cone with the same apex as C that is a subset of C. For a segment $\overline{rs} \subset P$, define the *visibility cone* of a point $p \in P$ *through* $\overline{rs}$, denoted $V(p, \overline{rs})$, as a region consisting of the rays from p that intersect $\overline{rs}$ before properly crossing the boundary of P.[1] Define a visibility cone $C(p)$ *into* a subpolygon U for $p \in P \setminus U$ as the visibility cone through the diagonal of P that separates U from the subpolygon containing p.

▶ **Lemma 1.** *Let P be a simple polygon with n vertices, and let A be a set of m points or line segments in P. For a point or a line segment Q, we can find $C(Q, A)$ in time $\mathcal{O}(n + m \log n)$.*

Proof. If A is a set of points, it suffices to compute the visibility polygon of Q and preprocess it for $\mathcal{O}(\log n)$-time point location queries. Both preprocessing steps take linear time [23, 28], and querying takes $\mathcal{O}(m \log n)$ time in total. In case A consists of line segments, we can similarly test if one of the endpoints of each segment of A is visible, thus making the segment visible. We also need to count the number of visible segments whose endpoints lie outside $V(Q)$. This can be done in $\mathcal{O}(n + m \log n)$ time by computing a sufficiently large bounding box B of $V(Q)$, and constructing an $\mathcal{O}(\log n)$-time ray shooting data structure on $B \setminus V(Q)$. This allows us to test if a segment intersects $V(Q)$ in $\mathcal{O}(\log n)$ time. The polygon $B \setminus V(Q)$ has only a single hole, so we can connect the boundary of $V(Q)$ to the boundary of B with a line segment $\overline{rs}$ and cut $B \setminus V(Q)$ along $\overline{rs}$ to obtain a simple polygon. We can then build a ray shooting structure [26] on this simple polygon, and answer a query by $\mathcal{O}(1)$ ray shooting queries. In particular, for any segment in A that does not cross $\overline{rs}$, we get the result directly; and for any segment that crosses $\overline{rs}$, we detect that the ray hits $\overline{rs}$ and do a second query on the other side of the cut. Either way, we use $\mathcal{O}(1)$ ray shooting queries. ◀

▶ **Lemma 2.** *Given a visibility polygon $V(p) \subseteq P$ for some point $p \in P$ and a line segment $\overline{rs} \subset P$, either $\overline{rs}$ and $V(p)$ do not intersect, or their intersection is a line segment.*

▶ **Corollary 3.** *The intersection between the line segment $\overline{rs} \subset P$ and the visibility cone $V(p, \overline{rs})$ for some $p \in P$ is either empty, or a line segment.*

[1] In case $p \in \overline{rs}$ holds, this definition yields $\mathbb{R}^2$. We handle such cases separately.

Cutting trees. A *cutting tree* [11, 14, 16] is a data structure commonly used for efficient half-plane range queries. Nesting multiple cutting trees in levels allows to efficiently perform simplex range searching and solve other related queries; we make extensive use of this. See Figure 1 for an illustration. We now discuss this data structure in more detail.

Suppose we want to preprocess a set $\mathcal{L}$ of m lines in the plane so that given a query point q, we can count the number of lines below the query point. Let $r \in [1, m]$ be a parameter; then a $(1/r)$-*cutting* of $\mathcal{L}$ is a subdivision of the plane with the property that each cell is intersected by at most m/r lines [11]. If q lies in a certain cell of the cutting, we know, for all lines that do not cross the cell, whether they are above or below q, and so we can store the count with the cell, or report the lines in a precomputed *canonical subset;* for the lines that cross the cell, we can recurse. The data structure that performs such a query is called a *cutting tree;* it can be constructed in $\mathcal{O}(m^{2+\varepsilon})$ time, uses $\mathcal{O}(m^{2+\varepsilon})$ space, and supports answering the queries in time $\mathcal{O}(\log m)$, for any constant $\varepsilon > 0$. Intuitively, the parameter r here determines the trade-off between the height of the recursion tree and the number of nodes for which a certain line in $\mathcal{L}$ is relevant. If we pick $r = m$, the $(1/r)$-cutting of $\mathcal{L}$ is just the arrangement of $\mathcal{L}$. The bounds above are based on picking $r \in \mathcal{O}(1)$, so the height of the recursion tree is $\mathcal{O}(\log m)$. This approach follows the work of Clarkson [16], with Chazelle [11] obtaining the bounds above by improving the cutting construction.

An obvious benefit of this approach over just constructing the arrangement on $\mathcal{L}$ and doing point location in that arrangement is that using cuttings, we can obtain $\mathcal{O}(\log m)$ canonical subsets and perform nested queries on them without an explosion in storage required; the resulting data structure is called a *multilevel cutting tree.* Specifically, we can query with k points and a direction associated with each point (above or below) and return the lines of $\mathcal{L}$ that pass on the correct side (above or below) of all k query points. If we pick $r \in \mathcal{O}(1)$ and nest k levels in a k-level cutting tree, we get the same construction time and storage bounds as for a regular cutting tree; but the query time is now $\mathcal{O}(\log^k m)$. Chazelle et al. [14] show that if we set $r = n^{\varepsilon/2}$, each level of a multilevel cutting tree is a constant-height tree, so the answer to the query can be represented using only $\mathcal{O}(1)$ canonical subsets and the query time is reduced to $\mathcal{O}(\log m)$. The space used and the preprocessing time remains $\mathcal{O}(m^{2+\varepsilon})$.

▶ **Lemma 4** ([14]). *Let $\mathcal{L}$ be a set of m lines and let k be a constant. Suppose we want to answer the following query: given k points and associated directions (above or below), find the lines in $\mathcal{L}$ that lie on the correct side of all k points. In time $\mathcal{O}(m^{2+\varepsilon})$, we can construct a data structure using $\mathcal{O}(m^{2+\varepsilon})$ storage that supports such queries. The lines are returned as $\mathcal{O}(1)$ canonical subsets, and the query time is $\mathcal{O}(\log m)$.*

Dualising the problem in the usual way, we can alternatively report or count points from the set A that lie in a query half-plane; or in the intersection of several half-planes, using a multilevel cutting tree.

▶ **Lemma 5** ([14]). *Let A be a set of m points and let k be a constant. In time $\mathcal{O}(m^{2+\varepsilon})$, we can construct a data structure using $\mathcal{O}(m^{2+\varepsilon})$ storage that returns $\mathcal{O}(1)$ canonical subsets with the points in A that lie in the intersection of the k query half-planes in time $\mathcal{O}(\log m)$.*

▶ **Lemma 6.** *Let A be a set of m arbitrary points in P, with each $a \in A$ an apex of some cone C_a. At query time, we get the point q, an apex of a cone C_q. In time $\mathcal{O}(m^{2+\varepsilon})$, we can construct a data structure using $\mathcal{O}(m^{2+\varepsilon})$ space that returns a representation of the points in $A' \subseteq A$, so that for any $p \in A'$, we have $q \in C_p$ and $p \in C_q$. The points are returned as $\mathcal{O}(1)$ canonical subsets, and the query time is $\mathcal{O}(\log m)$; they can be counted in the same time.*

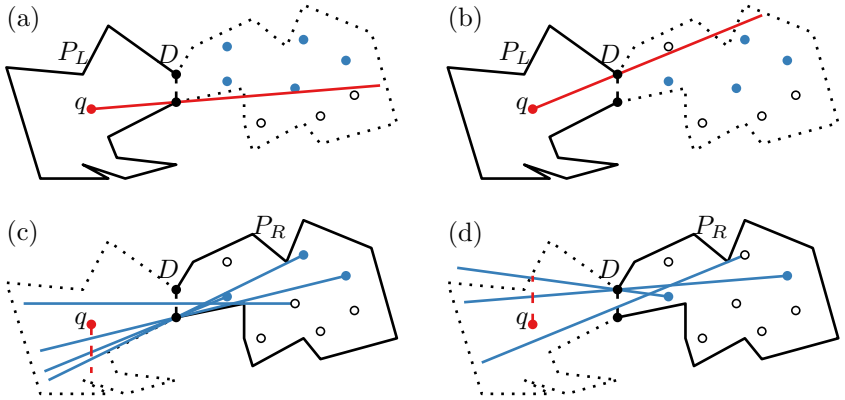


Figure 1 A query in a multilevel cutting tree, top left to bottom right. The query point is red; the selected points of A are blue. Black outline shows the relevant part of the polygon. (a, b) We select points in A above (resp. below) the right (resp. left) cone boundary of q. (c, d) We refine by taking points whose left (resp. right) cone boundary is below (resp. above) q.

▶ **Lemma 7.** *Let L be a vertical line and let A be a set of m cones starting left of L and whose left and right rays intersect L. In time $\mathcal{O}(m^{2+\varepsilon})$, we can construct two two-level cutting trees for A of total size $\mathcal{O}(m^{2+\varepsilon})$, so that for a query segment $\overline{pq}$ that is fully to the right of L, we can count the cones that contain or intersect $\overline{pq}$ in $\mathcal{O}(\log m)$ time.*

▶ **Lemma 8.** *Let $\mathcal{L}$ be a set of m lines and $\overline{pq}$ a query line segment. We can store $\mathcal{L}$ in a multilevel cutting tree, using $\mathcal{O}(m^{2+\varepsilon})$ space and preprocessing time, so that we can count the lines in $\mathcal{L}$ intersected by $\overline{pq}$ in time $\mathcal{O}(\log m)$.*

Polygon decomposition. For a simple polygon P on n vertices, Chazelle [9] shows that we can construct a balanced hierarchical decomposition of P by recursively splitting the polygon into two subpolygons of roughly equal size, using only diagonals (segments between two vertices of the polygon). The recursion stops when reaching triangles. The decomposition can be computed in $\mathcal{O}(n)$ time and stored using $\mathcal{O}(n)$ space in a balanced binary tree [9, 10, 22].

Hourglasses and the shortest path data structure. An *hourglass* for two diagonals $\overline{pq}$ and $\overline{rs}$ in a simple polygon P is the union of geodesic shortest paths in P from points on $\overline{pq}$ to points on $\overline{rs}$ [22]. Such an hourglass is bounded by two diagonals and two inward convex chains. Call one of the diagonals the *left* diagonal and the other the *right* diagonal. By following the hourglass boundary in a clockwise manner, starting from the left diagonal, we visit the *upper convex chain,* the right diagonal, and the *lower convex chain.* If the upper chain and lower chain of an hourglass share vertices, it is *closed,* otherwise it is *open.* We only explicitly use open hourglasses in our constructions.[2] A *visibility glass* is a subset of the hourglass, restricted to the line segments between points on $\overline{pq}$ and points on $\overline{rs}$ [17].

Guibas and Hershberger [22, 25] describe a data structure to compute shortest paths in a simple polygon P. They use the polygon decomposition by Chazelle [9] and also store hourglasses between the splitting diagonals of the decomposition. The data structure uses $\mathcal{O}(n)$ storage and preprocessing time and can answer the following queries in $\mathcal{O}(\log n)$ time:

[2] If an hourglass is closed, the two chains are only inward convex from the endpoints of a diagonal until they meet.

Segment location query. Given a segment $\overline{pq}$, return the two leaf triangles containing p and q in the decomposition and the $\mathcal{O}(\log n)$ pairwise disjoint open hourglasses so that the triangles and hourglasses fully cover $\overline{pq}$. We call this structure the *polygon cover* of $\overline{pq}$.

Shortest path query. Given points $p, q \in P$, return the geodesic shortest path between p and q in P as a set of $\mathcal{O}(\log n)$ nodes of the decomposition. The shortest path between p and q is a concatenation of subcurves of the polygonal chains of the appropriate boundaries of the (open or closed) hourglasses in these $\mathcal{O}(\log n)$ nodes together with at most $\mathcal{O}(\log n)$ segments connecting two consecutive subcurves.

Cone query. Given a point s and a line segment $\overline{pq}$ in P, return $V(s, \overline{pq})$. This can be done by computing the shortest paths from s to p and to q and taking the first segments of each path starting at s to extend them into the bounding rays of a cone.

3 Point Queries

In this section, given a set A of m points in a simple polygon P on n vertices, we count the points of A that are in the visibility polygon of a query point $q \in P$. We present two solutions: (i) an arrangement-based approach that also applies in the case where A contains line segments, which achieves low query time at the cost of large storage and preprocessing time; and (ii) a cutting-tree-based approach with query times slower by a factor of $\mathcal{O}(\log n)$, but with much better storage requirements and preprocessing time.

3.1 Point Location in an Arrangement

The approach relies on the fact that the number of objects in A visible to a query point q is equal to the number of (weak) visibility polygons of the objects in A stabbed by q. We construct all (weak) visibility polygons of the objects in A and compute the arrangement $\mathcal{A}$ of the edges of these polygons. For each cell C in the arrangement, we store the number of visibility polygons that contain C. Then a point location query for q yields the number of visible objects in A. Computing the visibility polygons takes $\mathcal{O}(nm)$ time, and constructing the arrangement using an output-sensitive line segment intersection algorithm takes $\mathcal{O}(nm \log nm + |\mathcal{A}|)$ time [12], where $|\mathcal{A}|$ is the number of vertices of $\mathcal{A}$. Building a point location structure on $\mathcal{A}$ for $\mathcal{O}(\log|\mathcal{A}|)$-time point location queries takes $\mathcal{O}(|\mathcal{A}|)$ time [28]; the space used is $\mathcal{O}(|\mathcal{A}|)$. As Bose et al. [6] show, the worst-case complexity of $\mathcal{A}$ is $\Theta(nm^2)$.

▶ **Theorem 9.** *Let P be a simple polygon with n vertices, and let A be a set of m points or line segments in P. In $\mathcal{O}(nm^2 + nm \log nm)$ time, we can build a data structure of size $\mathcal{O}(nm^2)$ that can report the number of points or segments in A visible from a query point q in $\mathcal{O}(\log nm)$ time.*

3.2 Hierarchical Decomposition

To design a data structure that uses less storage than that of Section 3.1, we observe that if we subdivide the polygon, we can count the visible objects by summing up the number of visible objects in the cells of the subdivision. To efficiently compute these counts, we use the polygon decomposition approach (see Section 2). With each split in our decomposition, we store data structures that can efficiently count the visible objects in the associated subpolygon.

Cone containment. Let us solve the following problem first. We are given a simple polygon P and a (w.l.o.g.) vertical diagonal D that splits it into two simple polygons P_L and P_R. Furthermore, we are given a set A of m points in P_L. Given a query point q in P_R, we want to count the points in A that see q. We base our approach on the following observation.

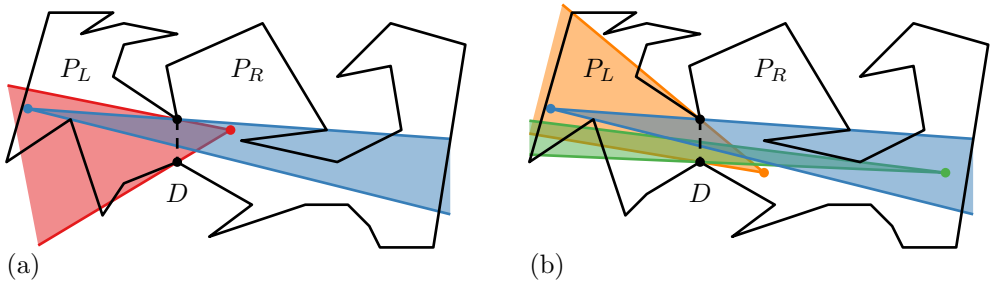


Figure 2 Visibility cones (coloured regions) of (coloured) points w.r.t. some diagonal D. (a) Blue and red are mutually visible. (b) Green and blue cannot see each other, nor can orange and blue.

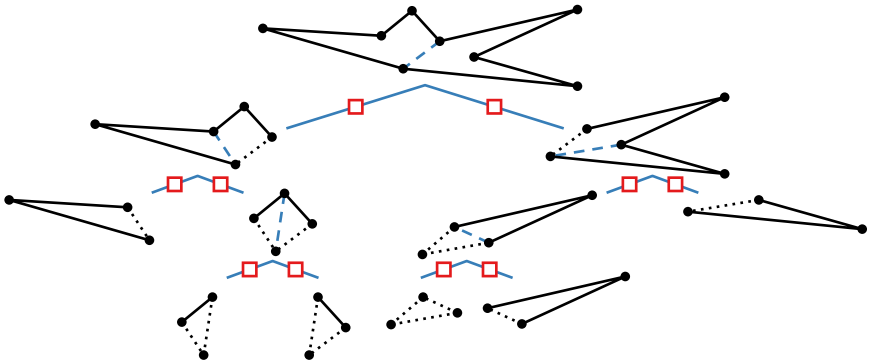


Figure 3 Augmented polygon decomposition following the approach by Chazelle [9]. Each node corresponds to the splitting diagonal (blue dashed line). Along the tree edges (blue lines), we store the multilevel cutting tree (red box) for the polygon in the child using the diagonal of the parent.

▶ **Lemma 10.** *Given a simple polygon P, split into two simple polygons P_L and P_R by a diagonal D between two vertices; and given two points $p \in P_L$ and $q \in P_R$, consider the visibility cones $V(p, D)$ and $V(q, D)$, i.e. the cones from p and q through D into the other subpolygons. Point p sees q in P if and only if $q \in V(p, D)$ and $p \in V(q, D)$.*

Lemma 10 shows that to count the points in A that see q, it suffices to construct the cones from all points in A through D and the cone from q through D and count the points in A satisfying the condition of Lemma 10 (see Figure 2). The cones $V(p, D)$ from all $p \in A$ can be precomputed (we shall handle this later), so only the cone $V(q, D)$ needs to be computed at query time. The query of this type can be realised using a multilevel cutting tree (Lemma 6).

Decomposition. Let us return to the original problem. To solve it, we can use the balanced polygon decomposition [9] (see Section 2). Following Guibas and Hershberger [22, 25], we represent it as a binary tree (see Figure 3). Observe that as long as there is some diagonal D separating our query point from a subset of points of A, we can use the approach above.

Every node of the tree is associated with a diagonal, and the two children correspond to the left and the right subpolygons. With each node, we store two data structures described above: one for the query point to the left of the diagonal and one for the query to the right.

The query then proceeds as follows. Suppose the polygon P is triangulated, and the triangles correspond to the leaves in the decomposition. Given a query point q, find the triangle it belongs to; then traverse the tree bottom up. In the leaf, q can see all the points of A that are in the same triangle, so we start with that count. As we proceed up the tree,

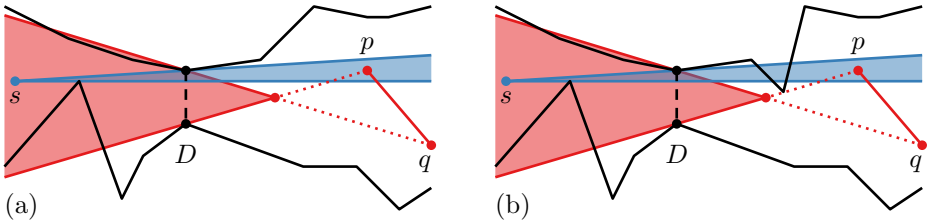


Figure 4 (a) For the cone that describes visibility of $\overline{pq}$ through D, Lemma 10 does not hold – there can be visibility without visibility between the apices of the cones. (b) The segment $\overline{pq}$ intersects the cone of s, and s is in the cone of $\overline{pq}$, but they cannot see each other, so testing intersection between the objects and the cones also does not work directly.

we query the correct associated data structure – if q is to the right of the diagonal, we want to count the points to the left of the diagonal in the current subpolygon that see q. It is easy to see that this way we end up with the total number of points in A that see q, since the subsets of A that we count are disjoint as we move up the tree and cover the entire set A.

▶ **Theorem 11.** *Let P be a simple polygon with n vertices, and let A be a set of m points inside P. In $\mathcal{O}(n + m^{2+\varepsilon} \log n + m \log^2 n)$ time, we can build a data structure of size $\mathcal{O}(n + m^{2+\varepsilon} \log n)$ that can report the number of points from A visible from a query point q in $\mathcal{O}(\log n \log m + \log^2 n)$ time.*

Proof. The correctness follows from the considerations above; it remains to analyse the time and storage requirements. For the query time, we do point location of the query point q in the triangulation of P and make a single pass up the decomposition tree, making queries in the associated multilevel cutting trees. Clearly, the height of the tree is $\mathcal{O}(\log n)$. At every level of the decomposition tree, we need to construct the visibility cone from the query point to the current diagonal; this can be done in $\mathcal{O}(\log n)$ time with a cone query (see Section 2). Then we need to query the associated data structure, except at the leaf, where we simply fetch the count. The query then takes time $\mathcal{O}(\log n \log m + \log^2 n)$. For the storage requirements, we need to store the associated data structures on in total m points at every level of the tree, as well as a single copy of the shortest path data structure, yielding overall $\mathcal{O}(n + m^{2+\varepsilon} \log n)$ storage. Finally, we analyse the preprocessing time. Triangulating a simple polygon takes $\mathcal{O}(n)$ time [10]. Constructing the decomposition can be done in additional $\mathcal{O}(n)$ time [22]. Constructing the associated data structures takes time $\mathcal{O}(m^{2+\varepsilon})$ per level, so $\mathcal{O}(m^{2+\varepsilon} \log n)$ overall, after determining the visibility cones for the points of A to all the relevant diagonals, which can be done in time $\mathcal{O}(m \log^2 n)$, as each point of A occurs a constant number of times per level of the decomposition, and constructing the cone takes $\mathcal{O}(\log n)$ time. Overall we need $\mathcal{O}(n + m^{2+\varepsilon} \log n + m \log^2 n)$ time. ◀

▶ Remark 12. While this approach uses many of the ideas needed to tackle the setting with segment queries, Lemma 10 does not apply – see Figure 4.

4 Segment Queries

In this section, we are given a simple polygon P and a set A of stationary entities (points) in P. We construct a data structure to count the points in A that see a query segment $\overline{pq}$. We cannot reuse the approach of Section 3.1, as the query $\overline{pq}$ may intersect multiple arrangement cells. Thus, we construct a new data structure using the insights of the hierarchical decomposition of Section 3.2. For the complete argument we refer to the full version.

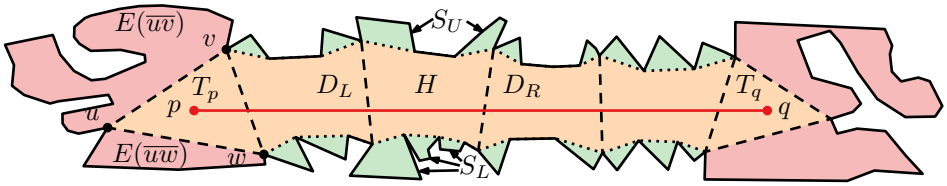


Figure 5 Partitioning of the polygon based on the polygon cover of $\overline{pq}$. The hourglasses and triangles are shown in orange; the side polygons in green; and the end polygons in red.

High-level overview. We use the data structure by Guibas and Hershberger [22] (abbreviated GHDS) on P as the basis and augment the elements of GHDS with data structures that allow us to perform the queries. Recall from Section 2 that we can obtain the polygon cover for a query segment $\overline{pq}$ using the GHDS, consisting of $\mathcal{O}(\log n)$ hourglasses and two triangles T_p, T_q containing p and q. For a given query $\overline{pq}$, the polygon cover partitions P into regions of three types (see Figure 5):

1. Hourglasses and triangles T_p, T_q intersecting $\overline{pq}$ and containing p or q, respectively.
2. Side polygons, each incident to the upper or lower chain of an hourglass.
3. End polygons that are adjacent to T_p and T_q.

For each type, we now count the points in A in a region of that type that see $\overline{pq}$.

4.1 Counting Points from A in Triangles and Hourglasses

We count the points in A contained in a region of Type 1 that see $\overline{pq}$. Triangles are convex, so any point in a triangle can see $\overline{pq}$. Similarly, each hourglass is completely traversed by $\overline{pq}$, so every point inside an hourglass can see $\overline{pq}$. During preprocessing, for every region in the GHDS (a triangle T or an hourglass H), we count the points from A in the region. Specifically, we construct a half-plane range query data structure in $\mathcal{O}(m^{2+\varepsilon})$ time and, for a triangle T, count the points in $A \cap T$ with three consecutive half-plane range queries. The total complexity of the hourglasses in the GHDS is $\mathcal{O}(n \log^3 n)$ [17, Lemma 7]. We triangulate each hourglass H to compute $|A \cap H|$ in $\mathcal{O}(n \log^3 n \log m)$ total time. Given $\overline{pq}$, we compute the sum over all $\mathcal{O}(\log n)$ Type 1 regions in $\mathcal{O}(\log n)$ time.

▶ **Theorem 13.** *We can construct an $\mathcal{O}(n)$-size data structure in $\mathcal{O}(m^{2+\varepsilon} + n \log^3 n \log m)$ time, so that we can count all points in Type 1 regions that see $\overline{pq}$ in $\mathcal{O}(\log n)$ time.*

4.2 Counting Points from A in Side Polygons

The points in a Type 2 region (side polygon) do not always see $\overline{pq}$. Let H be an hourglass with two diagonals D_L, D_R and two chains π_U, π_L. Let S_U be the side polygon bounded by the upper chain π_U; the analysis for S_L bounded by the lower chain π_L is symmetrical. A point $a \in A \cap S_U$ can see $\overline{pq}$ if its visibility cone $C(a)$ into $P \setminus S_U$ is not empty. Furthermore, we discern three mutually exclusive types of cones (Figure 6):

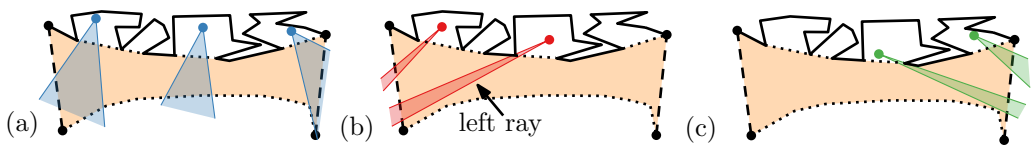


Figure 6 The three cases (a), (b), and (c) for cones originating from a side polygon S_U.

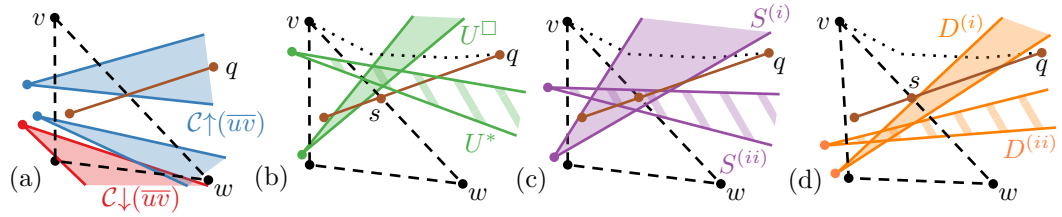


Figure 7 (a) We partition $\mathcal{C}_{\overline{uv}}$ based on vertex w into sets $\mathcal{C}\uparrow(\overline{uv})$ that pass above w or contain it and $\mathcal{C}\downarrow(\overline{uv})$ that pass below w. (b–d) We partition $\mathcal{C}\uparrow(\overline{uv})$ based on $s = \overline{pq} \cap \overline{vw}$ into sets that pass above s (U), below s (D), or contain it (S).

(a) $C(a)$ intersects the lower chain π_L of H;
(b) $C(a)$ intersects *only* the left diagonal D_L of H; and
(c) $C(a)$ intersects *only* the right diagonal D_R of H.
Cones of Type (a) see $\overline{pq}$, since $\overline{pq}$ separates the upper and the lower chain (we count them during preprocessing). Types (b) and (c) are symmetrical; we discuss Type (b) for a fixed side polygon S_U and, without loss of generality, we assume that the directed segment $\overline{pq}$ crosses D_L before D_R. At preprocessing, we store $A_L \subseteq A \cap S_U$ where for all $a \in A_L$, visibility cone $C(a)$ intersects only D_L; and we store $|A_L|$. To count the visible cones of Type (b) at query time, we subtract from $|A_L|$ the number of elements in A_L that do *not* see $\overline{pq}$.

▶ **Lemma 14.** *A point $a \in A_L$ is not visible to $\overline{pq}$ if and only if the left ray of $C(a)$ intersects the shortest path from p to the top of D_L.*

On a high level, we count the points $a \in A_L$ for which the left ray of $C(a)$ intersects the shortest path from p to the top vertex v of D_L as follows. For each hourglass in GHDS, we store the shortest path map SPM(v) rooted at v [23], and for each edge e in SPM(v), we store the number of points $a \in A_L$ for which the left ray of $C(a)$ intersects e. We construct the SPMs and the cutting trees for the $\mathcal{O}(n)$ hourglasses in the GHDS using $\mathcal{O}(nm^{2+\varepsilon} + n^2)$ space and $\mathcal{O}(nm^{2+\varepsilon} + n^2 \log m)$ time. At query time, we obtain the shortest path $\pi(p, v)$ from p to v as a single segment $\overline{pu}$, followed by a path $\pi(u, v)$ in SPM(v) for some vertex u. We count the points $a \in A_L$ where the left ray of $C(a)$ intersects $\overline{pu}$ in $\mathcal{O}(\log m)$ time, and those where the left ray of $C(a)$ intersects $\pi(u, v)$ in $\mathcal{O}(\log n)$ time, thus avoiding double counting. Thus, we count the elements in A_L that do *not* see $\overline{pq}$ in $\mathcal{O}(\log nm)$ time per side polygon. Given $\overline{pq}$, we apply this strategy for all $\mathcal{O}(\log n)$ hourglasses.

▶ **Theorem 15.** *We can construct an $\mathcal{O}(nm^{2+\varepsilon} + n^2)$-size data structure in time $\mathcal{O}(nm^{2+\varepsilon} + n^2 \log m)$, so we can count all points in Type 2 regions that see $\overline{pq}$ in $\mathcal{O}(\log n \log nm)$ time.*

4.3 Counting Points from A in End Polygons

Type 3 regions (end polygons) are bounded by triangles in the decomposition T_p or T_q, containing p or q, respectively. Let $T_p = uvw$; it is incident to at most three end polygons. We describe the data structure for the end polygon $E(\overline{uv})$ incident to T_p through the edge $\overline{uv}$ (see Figure 7). All other data structures are symmetrical. We count the points in $A \cap E(\overline{uv})$ that see $\overline{pq}$. Denote the set of all visibility cones $V(a, \overline{uv})$ by $\mathcal{C}_{\overline{uv}}$ over all $a \in A \cap E(\overline{uv})$.

Consider the special case where $\overline{pq}$ is fully contained in a triangle ($T_p = T_q$). A triangle is convex, so we can immediately apply Lemma 8. Now assume $T_p \neq T_q$. The query segment $\overline{pq}$ intersects some boundary of T_p. We construct a data structure under the assumption that $\overline{pq}$ intersects edge $\overline{vw}$ of T_p; for $\overline{pq}$ intersecting $\overline{uw}$, we construct a symmetrical structure. So,

given a triangle T_p, where $\overline{pq}$ intersects $\overline{vw}$, we want to count the cones $C(a) \in \mathcal{C}_{\overline{uv}}$ whose apex a sees $\overline{pq}$. Our argument is a multilevel case distinction on $\mathcal{C}_{\overline{uv}}$. We first partition $\mathcal{C}_{\overline{uv}}$ during preprocessing (Figure 7a) into two sets:

1. $\mathcal{C}{\downarrow}(\overline{uv})$ are the cones in $\mathcal{C}_{\overline{uv}}$ that pass entirely below the vertex w; and
2. $\mathcal{C}{\uparrow}(\overline{uv})$ are the cones in $\mathcal{C}_{\overline{uv}}$ that pass entirely above or contain the vertex w.

Counting cones in $\mathcal{C}{\downarrow}(\overline{uv})$ whose apex sees $\overline{pq}$. For any cone $C(a) \in \mathcal{C}{\downarrow}(\overline{uv})$, its apex a sees $\overline{pq}$ if and only if p lies below the supporting line of the left ray of $C(a)$. We construct a cutting tree on these lines to count such cones in $\mathcal{O}(\log m)$ time.

Counting cones in $\mathcal{C}{\uparrow}(\overline{uv})$ whose apex sees $\overline{pq}$. We partition $\mathcal{C}{\uparrow}(\overline{uv})$ with an elaborate double case distinction. Observe that this conceptual case distinction can only be made at query time when we have access to $s = \overline{pq} \cap \overline{vw}$ and q (Figures 7 and 8).

(b) $U \subseteq \mathcal{C}{\uparrow}(\overline{uv})$ are the cones where both boundary rays intersect $\overline{vs}$ (but not s):
- $U^* \subseteq U$ are the cones whose apices lie above the supporting line of $\overline{sq}$;
- $U^\square \subseteq U$ is the set $U \setminus U^*$.

(c) $S \subseteq \mathcal{C}{\uparrow}(\overline{uv})$ are the cones which contain $s = \overline{pq} \cap \overline{vw}$:
- $S^{(i)} \subseteq S$ are the cones whose right ray intersects $\overline{sq}$;
- $S^{(ii)} \subseteq S$ is the set $S \setminus S^{(i)}$.

(d) $D \subseteq \mathcal{C}{\uparrow}(\overline{uv})$ are the cones where both boundary rays intersect $\overline{sw}$ (but not s):
- $D^{(i)} \subseteq D$ are the cones whose right ray intersects $\overline{sq}$;
- $D^{(ii)} \subseteq D$ is the set $D \setminus D^{(i)}$.

Counting cones in U whose apex sees $\overline{pq}$. Both U^* and $U^\square$ may contain cones whose apex sees $\overline{pq}$. We show how to count these for both sets using a data structure *only* on $\mathcal{C}{\uparrow}(\overline{uv})$. We show that cones in $U^\square$ are visible if and only if they intersect the segment $\overline{ps}$; we can test this in $\mathcal{O}(\log m)$ time. We count the cones in U^* whose apices see $\overline{pq}$ via an inclusion–exclusion argument. Specifically, we observe that all cones in $\mathcal{C}{\uparrow}(\overline{uv}) \setminus U^*$ have one of two mutually exclusive properties:

(i) For all cones in $S^{(i)}$, $D^{(i)}$, and $U^\square$, their right ray intersects $\pi(v, q)$ and lies above s.

(ii) For all cones in $S^{(ii)}$ and $D^{(ii)}$, their right ray lies below s and does not intersect $\pi(v, q)$.

Cones in U^* never have property (ii). Moreover, they are not visible if and only if their right ray intersects $\pi(v, q)$, i.e. invisible cones have property (i). Thus, the number of cones in U^* whose apices see $\overline{pq}$ is equal to $|\mathcal{C}{\uparrow}(\overline{uv})|$ minus all cones with property (i) or (ii). We count cones with property (i) using a shortest path map in $\mathcal{O}(\log n)$ time, identically to Section 4.2. We count cones with property (ii) using half-plane range queries in $\mathcal{O}(\log m)$ time.

Counting cones in S and D whose apex sees $\overline{pq}$. The apices of all cones in S see $\overline{pq}$. We identify these in $\mathcal{O}(\log m)$ time using a stabbing query on $\mathcal{C}{\uparrow}(\overline{uv})$. For cones in D, we can make a symmetrical case distinction, creating the sets D^* and $D^\square$, and we can handle them through an identical data structure.

▶ **Theorem 16.** *We can construct an $\mathcal{O}(nm^{2+\varepsilon} + n^2)$-size data structure in time $\mathcal{O}(nm^{2+\varepsilon} + n^2 \log m)$, so we can count all points in Type 3 regions that see $\overline{pq}$ in $\mathcal{O}(\log n \log nm)$ time.*

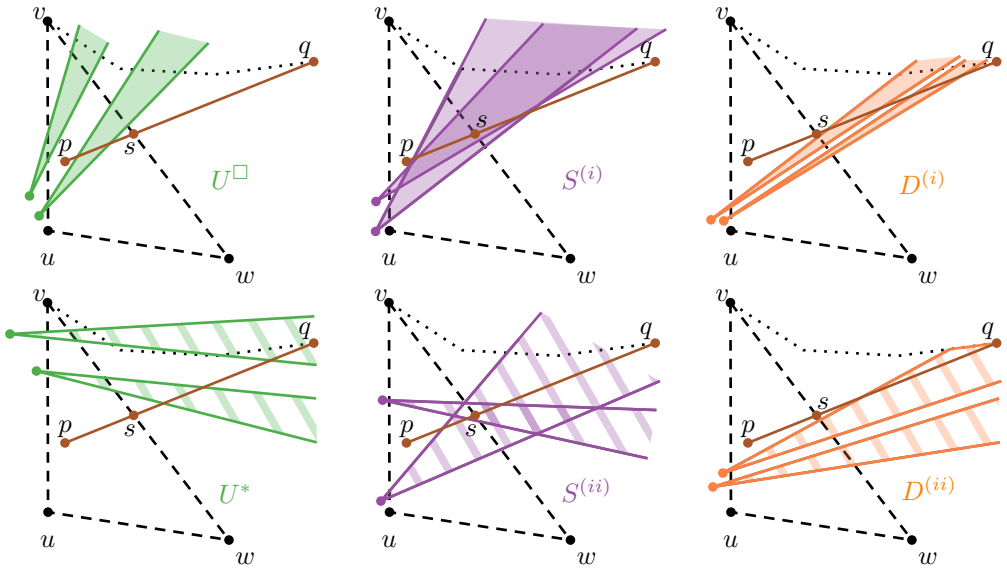


Figure 8 We partition $\mathcal{C}\uparrow(\overline{uv})$ into six sets: U^*, $U^{\square}$, $S^{(i)}$, $S^{(ii)}$, $D^{(i)}$, and $D^{(ii)}$.

Finalising the argument. In the full version, we show a lemma that allows us to efficiently construct all visibility cones from points of A in Type 2 and Type 3 regions. We store all points and visibility cones in the data structures of Theorems 13, 15, and 16 to show the main result of this section.

▶ **Theorem 17.** *Let P be a simple polygon with n vertices, and let A be a set of m points inside P. In time $\mathcal{O}(nm^{2+\varepsilon} + n^2 \log m)$, we can build a data structure of size $\mathcal{O}(nm^{2+\varepsilon} + n^2)$ to count the points from A visible from a query segment $\overline{pq}$ in $\mathcal{O}(\log n \log nm)$ time.*

5 Extensions and Future Work

We consider further variants of the problem. In particular, we extend the approach of Section 4 to the cases where A contains line segments, or even constant-complexity simple polygons, with the same bounds. Furthermore, we consider a version of the problem where we only want to test if at least one object in A sees the query. Finally, we tackle the question of subquadratic counting, where given two sets A and B with m points each, we wish to count the pairs from $A \times B$ that see each other in the simple polygon in time subquadratic in m. We briefly describe these results here and further refer the reader to the full version.

Extending to a set of segments or polygons. A natural extension to our GHDS would be to consider the case where A is a set of segments. As it turns out, there is a straightforward way to reuse our GHDS. Observe that any line segment that is *not* entirely contained in a single end polygon or side polygon intersects a triangle or an hourglass and thus is visible to $\overline{pq}$. Hence, it suffices to count the *invisible* line segments inside the individual side and end polygons and subtract that count from the total. Instead of the visibility cones, we use the *visibility glasses,* i.e. the collections of all visibility lines between two line segments (Figure 9). By intersecting the bitangents of the visibility glass, we get a new cone that describes the potentially visible region on the other side of the diagonal. The preprocessing time for GHDS construction dominates, and we do not add extra query time or storage. To use the same approach with constant-complexity simple polygons, we simply show that we can construct visibility glasses for them, as well. See the full version for details.

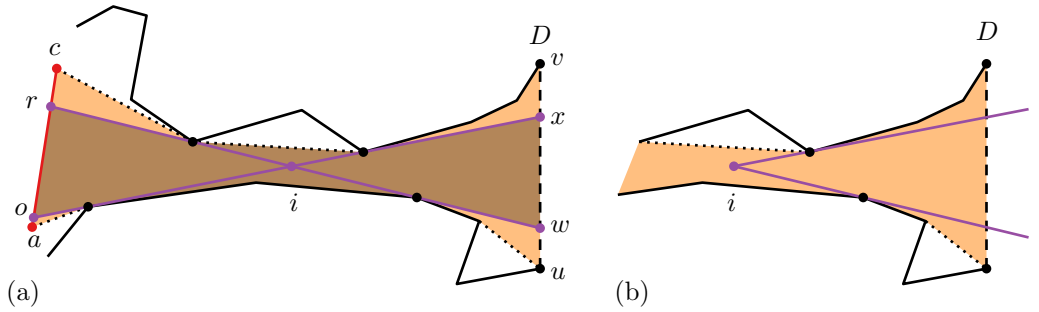


Figure 9 (a) The visibility glass (dark region) inside the hourglass (orange region) from segment $\overline{ac}$ to diagonal $D = \overline{uv}$. (b) The intersection point i and the two rays from i through w and x form a new visibility region in the subpolygon to the right of D.

Testing for visibility. If all we need to compute is if any of the objects from A can see Q we can avoid computing $C(Q, A)$, and answer queries more efficiently. We explicitly compute the union U of all (weak) visibility polygons of all m objects in A, and build point location and ray shooting data structures on it. Irrespective of the type of objects in A (points or segments) and the type of Q, we argue that U has complexity $\mathcal{O}(m(m+n))$, and that we can store using $\mathcal{O}(m(m+n))$ space so that we can answer queries in $\mathcal{O}(\log(m+n))$-time.

Subquadratic counting. Given two sets of points or line segments A and B, each of size m, in a simple polygon P with n vertices, we want to count the pairs in $A \times B$ that see each other. Using the work by Eades et al. [17], we can solve this problem by checking the visibility for all pairs, which is optimal for $n \gg m$. If at least one set consists of points, it runs in time $\mathcal{O}(n + m^2 \log n)$. When $m \gg n$, we want to avoid the m^2 factor, using our data structures. Suppose we have a data structure for visibility counting queries with query time $Q(m, n)$ and preprocessing time $P(m, n)$. Pick $k = m^s$ with $0 \leq s \leq 1$. We split the set A into sets $A_1, \ldots, A_k$, with m/k objects each; then we construct a data structure for each set. Finally, with each point in B, we query these k data structures and sum up the counts. The time that this approach takes is $\mathcal{O}\big(m^s \cdot P(m^{1-s}, n) + m^{1+s} \cdot Q(m^{1-s}, n)\big)$. Picking s to minimise it, we obtain algorithms subquadratic in m in all settings.

Moving points and query variations. We can interpret a line segment as a trajectory traced by a point moving with constant velocity. The results from Sections 3 and 4 directly apply in this setting, too. When A consists of line segments as above, we solve a different problem. Solving the visibility counting problem for moving points seems non-trivial. Alternatively, one could report the visible points, or ask to quantify the visible portions of segments. All of these would be highly exciting directions for future work.

References

1 Pankaj K. Agarwal and Micha Sharir. Applications of a new space-partitioning technique. *Discrete & Computational Geometry*, 9:11–38, 1993. doi:10.1007/BF02189304.

2 Pankaj K. Agarwal and Marc J. van Kreveld. Connected component and simple polygon intersection searching. *Algorithmica*, 15:626–660, 1996. doi:10.1007/BF01940884.

3 Sharareh Alipour, Mohammad Ghodsi, Alireza Zarei, and Maryam Pourreza. Visibility testing and counting. *Information Processing Letters*, 115(9):649–654, 2015. doi:10.1016/j.ipl.2015.03.009.

4 Boris Aronov, Leonidas J. Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *Discrete & Computational Geometry*, 27:461–483, 2002. doi:10.1007/s00454-001-0089-9.

5 Boaz Ben-Moshe, Olaf Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. Computing the visibility graph of points within a polygon. In Jack S. Snoeyink and Jean-Daniel Boissonnat, editors, *Proceedings of the 20th Annual Symposium on Computational Geometry (SoCG 2004)*, pages 27–35, New York, NY, USA, 2004. ACM. doi:10.1145/997817.997825.

6 Prosenjit Bose, Anna Lubiw, and James Ian Munro. Efficient visibility queries in simple polygons. *Computational Geometry: Theory & Applications*, 23(3):313–335, 2002. doi:10.1016/S0925-7721(01)00070-0.

7 Kevin Buchin, Bram Custers, Ivor van der Hoog, Maarten Löffler, Aleksandr Popov, Marcel Roeloffzen, and Frank Staals. Segment visibility counting queries in polygons, 2022. arXiv:2201.03490.

8 Mojtaba Nouri Bygi, Shervin Daneshpajouh, Sharareh Alipour, and Mohammad Ghodsi. Weak visibility counting in simple polygons. *Journal of Computational and Applied Mathematics*, 288:215–222, 2015. doi:10.1016/j.cam.2015.04.018.

9 Bernard Chazelle. A theorem on polygon cutting with applications. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*, pages 339–349, Piscataway, NJ, USA, 1982. IEEE. doi:10.1109/SFCS.1982.58.

10 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991. doi:10.1007/BF02574703.

11 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.

12 Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, 1992. doi:10.1145/147508.147511.

13 Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. *Discrete & Computational Geometry*, 4:551–581, 1989. doi:10.1007/BF02187747.

14 Bernard Chazelle, Micha Sharir, and Emo Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992. doi:10.1007/BF01758854.

15 Danny Ziyi Chen and Haitao Wang. Weak visibility queries of line segments in simple polygons. *Computational Geometry: Theory & Applications*, 48(6):443–452, 2015. doi:10.1016/j.comgeo.2015.02.001.

16 Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987. doi:10.1007/BF02187879.

17 Patrick Eades, Ivor van der Hoog, Maarten Löffler, and Frank Staals. Trajectory visibility. In Susanne Albers, editor, *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, number 162 in Leibniz International Proceedings in Informatics (LIPIcs), pages 23:1–23:22, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2020.23.

18 Hossam El Gindy and David Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2(2):186–197, 1981. doi:10.1016/0196-6774(81)90019-5.

19 Subir Kumar Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, Cambridge, UK, 2007. doi:10.1017/CBO9780511543340.

20 Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors. *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, New York, NY, USA, 3rd edition, 2017. doi:10.1201/9781315119601.

21 Joachim Gudmundsson and Pat Morin. Planar visibility: Testing and counting. In David G. Kirkpatrick and Joseph S. B. Mitchell, editors, *Proceedings of the 26th Annual Symposium on Computational Geometry (SoCG 2010)*, pages 77–86, New York, NY, USA, 2010. ACM. doi:10.1145/1810959.1810973.

22 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.

23 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. doi:10.1007/BF01840360.

24 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.

25 John Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991. doi:10.1016/0020-0190(91)90064-O.

26 John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995. doi:10.1006/jagm.1995.1017.

27 Barry Joe and Richard B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27:458–473, 1987. doi:10.1007/BF01937271.

28 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983. doi:10.1137/0212002.

29 Der-Tsai Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, 22(2):207–221, 1983. doi:10.1016/0734-189X(83)90065-8.

30 Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, volume 3 of *The International Series of Monographs on Computer Science*. Oxford University Press, Oxford, UK, 1987. URL: http://www.science.smith.edu/~jorourke/books/ArtGalleryTheorems/art.html.

31 Mark H. Overmars and Emo Welzl. New methods for computing visibility graphs. In Herbert Edelsbrunner, editor, *Proceedings of the 4th Annual Symposium on Computational Geometry (SoCG 1988)*, pages 164–171, New York, NY, USA, 1988. ACM. doi:10.1145/73393.73410.

32 Subhash Suri and Joseph O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In Alok Aggarwal, editor, *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG 1986)*, pages 14–23, New York, NY, USA, 1986. ACM. doi:10.1145/10515.10517.

Shortest Beer Path Queries in Interval Graphs

Rathish Das
Department of Computer Science, University of Liverpool, UK

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Eitan Kondratovsky
Cheriton School of Computer Science, University of Waterloo, Canada

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo, Canada

Anurag Murty Naredla
Cheriton School of Computer Science, University of Waterloo, Canada

Kaiyu Wu
Cheriton School of Computer Science, University of Waterloo, Canada

Abstract

Our interest is in paths between pairs of vertices that go through at least one of a subset of the vertices known as beer vertices. Such a path is called a beer path, and the beer distance between two vertices is the length of the shortest beer path.

We show that we can represent unweighted interval graphs using $2n \log n + O(n) + O(|B| \log n)$ bits where $|B|$ is the number of beer vertices. This data structure answers beer distance queries in $O(\log^{\varepsilon} n)$ time for any constant $\varepsilon > 0$ and shortest beer path queries in $O(\log^{\varepsilon} n + d)$ time, where d is the beer distance between the two nodes. We also show that proper interval graphs may be represented using $3n + o(n)$ bits to support beer distance queries in $O(f(n) \log n)$ time for any $f(n) \in \omega(1)$ and shortest beer path queries in $O(d)$ time. All of these results also have time-space trade-offs.

Lastly we show that the information theoretic lower bound for beer proper interval graphs is very close to the space of our structure, namely $\log(4 + 2\sqrt{3})n - o(n)$ (or about $2.9n$) bits.

2012 ACM Subject Classification Theory of computation → Data compression; Mathematics of computing → Paths and connectivity problems

Keywords and phrases Beer Path, Interval Graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.59

Related Version *Full Version*: `https://arxiv.org/abs/2209.14401` [6]

1 Introduction

The concept of a beer path was recently introduced by Bacic et al. [2]. The premise is simple, suppose you wish to visit a friend, and wish to pick up some beer along the way because you don't want to show up empty handed, what is the fastest way to do so? More formally, for a graph, we specify a set of vertices, which will act as beer stores. A beer path is one which passes through at least one of these designated vertices. We will say a beer graph is one where we have designated a subset of the vertices to be beer stores. Though this premise may be somewhat silly, it can have many applications related to delivering objects. For example, suppose you are going on a road trip and to be efficient, want to drop something off at a post office on the way. Or perhaps on your trip, you realize that you currently don't have enough gas, so you must visit a gas station somewhere along the way. Another hypothetical situation would be if a package needs to be transported, but due to regulations, one of the stops must be equipped for an inspection.

It is easily seen that a shortest beer path may not be simple, but will always consist of two shortest paths: from the beer store to the source, and to the destination.

In this paper, we study the shortest beer path problem on unweighted interval graphs: intersection graphs of intervals on the real line. Interval graphs are a well-known class of graphs and have applications in operations research [3] and bioinformatics [18]. For a more indepth treatment of interval graphs and their applications, see the book of Golumbic [7].

Related Work. Bacic et al. [2] studied the problem on weighted outerplanar graphs. They showed that on an outerplanar graph of n vertices, a data structure of size $O(m)$ words for any $m \geq n$ can be constructed in $O(m)$ time to support shortest beer path and beer distance – the length of the shortest beer path in $O(\alpha(m, n))$ time, where α is the inverse Ackermann function.

On the more general problem of graph data structures, Acan et al. [1] showed that interval graphs may be represented succinctly using $n \log n + O(n)$ bits of space to answer basic navigational queries: `adjacent`, `degree`, `neighbourhood` and `shortest_path` in optimal time: $O(1)$ or $O(1)$ for each vertex in the output. Building on Acan et al.'s work, He et al. [9] added the `dist` query for interval graphs. Their data structure has the same space $n \log n + O(n)$[1] bits, the same run time of the old operations but also supports `dist` in $O(1)$ time.

1.1 Our Results and Paper Layout

We give data structures for beer interval graphs and beer proper interval graphs that have time-space trade offs. An interval graph is a graph where we may assign an interval on the real line to each vertex - $v \mapsto [l_v, r_v]$ such that two vertices u, v are adjacent exactly when the corresponding intervals intersect [8, 11]. A proper interval graph is an interval graph where the intervals must be chosen so that no two intervals nest. Furthermore, we prove a lower bound result on the space required for beer proper interval graphs.

The main obstacle in constructing the data structures for the beer path queries is that the set of paths between two vertices (which are normally the feasible solutions to the shortest path problem) are arbitrarily filtered by the beer nodes into a smaller set of feasible solutions to the beer shortest path problem - by whether a beer node exists on the path or not. In the case that a beer node exists on one of the shortest paths, then it is clearly optimal and we must be able to detect this. Thus we must be able conduct this filtering process as well, and we achieve this by using orthogonal range search on the previously established data structures which looks at all paths.

In section 3 we study the beer distance problem in proper interval graphs. We first outline the steps that we need to implement in our data structures. Then in subsection 3.2 we give a data structure occupying $3n + o(n) + O(|B| \log n)$ bits of space, supporting all regular operations in the same complexity as the previous works of Acan et al. and He et al., and the queries related to beer distance:

- `beer_dist` in $O(\log^{\varepsilon} n)$ time, for any constant $\varepsilon > 0$
- `beer_shortest_path` in $O(1)$ time per vertex on the path.

We may also utilize a trade-off provided by our auxiliary data structures, which decrease the query times by substituting $\log^{\varepsilon} n$ with $\log \log n$ at the cost of increasing one of the space cost terms from $O(|B| \log n)$ to $O(|B| \log n \log \log n)$ bits. We note that in this data structure,

[1] We will use log to denote $\log_2$.

the space is dependent on $|B|$ the number of beer vertices, and is therefore undesirable if $|B|$ is large. In the case that there are many beer vertices, the above data structure can use $O(n \log n)$ bits of space.

In subsection 3.3, we use the tree structure of the distance queries to eliminate the dependence on $|B|$ at the cost of slightly increasing the run time. For beer proper interval graphs, we have a data structure using $3n + o(n)$ bits of space, which supports all the regular queries in their original optimal complexities, and the beer queries:

- `beer_dist` in $O(f(n) \log n)$ time for any function $f(n) \in \omega(1)$. Different $f(n)$ will impact the lower order term $o(n)$ in the space complexity.
- `beer_shortest_path` in $O(1)$ per vertex on the path.

In section 4 we study the beer distance problem in interval graphs. For this, we give a data structure using $2n \log n + O(n) + O(|B| \log n)$ bits, where $|B|$ is the number of beer vertices in the graph, and supports all regular operations in the same time complexity as above, and

- `beer_dist` in $O(\log^{\varepsilon} n)$ time, for any constant $\varepsilon > 0$.
- `beer_shortest_path` in $O(\log^{\varepsilon} n + d)$ time, where d is the beer distance between the two vertices.

Again we may utilize the same trade off to replace the $\log^{\varepsilon} n$ term by $\log \log n$ at the cost of increase the space term $|B| \log n$ to $|B| \log n \log \log n$.

Finally in section 5 we count the number of non-isomorphic beer proper interval graphs and use this to give an information theoretic lower bound on the space required for any data structure for beer proper interval graphs that can support `adjacent` and `beer_dist`. It may seem natural that to store which vertices are beer nodes will require an additional n bits but we show that the lower bound is actually asymptotically $\log(4 + 2\sqrt{3})n \approx 2.9n$ bits. The main insight into seeing why an additional n bits is not required is that for a clique, it suffices to only store $\log n$ bits for the count of how many beer nodes. For general interval graphs, the space required for the beer nodes is at most n and is a lower order term to the lower bound of $n \log n$ bits. Thus there is nothing study in this case.

A full version with all proofs can be found on arXiv [6].

2 Preliminaries

In this paper, we will use the standard graph theoretic notation. We will use $G = (V, E)$ to denote a graph with vertex set V and edge set E. We will use $n = |V|$ and $m = |E|$ to denote the number of vertices and edges. All of our graphs will be unweighted.

As we will be discussing both trees and graphs in general, we will use vertices to denote the vertices of a graph which may or may not be a tree, and nodes to denote the vertices of a tree.

In the paper, we assume the word-RAM model with $\Theta(\log n)$-size words. We use $\log(\cdot)$ to denote $\log_2(\cdot)$.

In a beer graph, we take any underlying graph G together with a set $B \subseteq V$ of *beer vertices*. This allows us to define the following queries:

- `beer_shortest_path`(u, v): return a shortest path between the vertices u and v such that at least one of the beer vertices appears on the path.
- `beer_dist`(u, v): return the length of the shortest path between vertices u and v such that at least one of the beer vertices appears on the path.

These are the restricted queries to the ordinary `shortest_path` and `dist` queries, which do not have the constraint that it must pass through a beer vertex.

For example, if $B = V$, then the two queries reduces to ordinary shortest path or distance in the graph. On the other extreme, if $B = \{b\}$ is a singleton, then the query reduces to two ordinary shortest path or distance queries in the graph.

2.1 Interval Graphs

An interval graph G is a graph where we may assign an interval on the real line to each vertex - $v \mapsto [l_v, r_v]$ such that two vertices u, v are adjacent exactly when the corresponding intervals intersect [8]. In particular, we may sort the endpoints so that the values are integers between 1 and $2n$. We sort the vertices based on their left endpoints, so that when we refer to vertex v, we are referring to a number (the rank of the vertex in the sorted order), and thus, a statement such as $u < v$ makes sense.

Acan et al. [1] showed that interval graphs can be represented using $n \log n + O(n)$ bits to support `adjacent`, `degree`, `neighbourhood`, `shortest_path` queries in optimal time. `adjacent` answers whether two given vertices are adjacent, `degree` answers the degree of the given vertex, `neighbourhood` gives a list of the neighbours of the given vertex, and `shortest_path` gives a shortest path between the two given vertices.

He et al. [9] showed that we can also answer the `dist` query in optimal time. Therefore we have the following theorem on interval graphs:

▶ **Lemma 1.** *An interval graph G on n vertices can be represented succinctly using $n \log n + O(n)$ bits to support* `adjacent`, `degree` *and* `dist` *queries in $O(1)$ time,* `neighbourhood` *in $O(1)$ time per neighbour and* `shortest_path` *in $O(1)$ time per vertex on the path.*

An interval graph G is *proper* (or a *proper interval graph*) if we can choose the intervals corresponding to vertices such that no two intervals are nested.

Acan et al. [1] also showed that proper interval graphs can be represented using $2n + o(n)$ bits to support `adjacent`, `degree`, `neighbourhood`, `shortest_path` queries in optimal time. He et al. [9] showed that we may also support the `dist` query in optimal time. Thus we have the following theorem on proper interval graphs:

▶ **Lemma 2.** *A proper interval graph G on n vertices can be represented succinctly using $2n + o(n)$ bits to support* `adjacent`, `degree` *and* `dist` *queries in $O(1)$ time,* `neighbourhood` *in $O(1)$ time and* `shortest_path` *in $O(1)$ time per vertex on the path.*

2.2 Dyck Paths

For our lower bound, we will be discussing Dyck paths. A Dyck path of length $2n$ is a path from $(0, 0)$ to $(2n, 0)$ using $2n$ steps, n of which are $(1, 1)$ steps which are referred to as up-steps, and n of which are $(1, -1)$ steps and are referred to as down-steps. Such a path must also satisfy the condition that it never reaches below the x-axis. It is well known that the number of Dyck paths of length $2n$ is $C_n = \frac{1}{n+1}\binom{2n}{n}$ the n-th Catalan number.

A Dyck path that never touches the x-axis except at the start and end, is referred to as an irreducible Dyck path. By removing the up-step at the beginning and the down-step at the end, the remainder of the path is simply a Dyck path of length $2(n-1)$. Thus the number of irreducible Dyck paths of length $2n$ is simply C_{n-1}.

For any Dyck path, we may associate an up-step with an open parenthesis (and a down-step with a close parenthesis). The sequence we obtain from a Dyck path is a balanced parenthesis sequence (and vice versa) as the Dyck path condition is exactly the condition that the excess in the balanced parenthesis sequence is never negative. This well known bijection allows us to associate an forest to each Dyck path, using the well known bijection for forests and balanced parentheses (via a depth-first traversal). In particular, if the Dyck path were irreducible, then the forest is just a single tree.

2.3 Succinct Data Structures

The information theoretic lower bound to represent a family of objects with N elements is $\lceil \log N \rceil$ bits. Any fewer bits and we do not have enough bit strings to assign a unique one to each object, and thus cannot distinguish between them. A succinct data structure aims to use $\log N + o(\log N)$ bits to represent these objects while supporting the relevant queries.

A bit vector is a length n array of bits, that supports the queries `rank`(i): given an index, return the number of 1s up to index i, `select`(j): given a number j, return the index of the jth one in the array, and `access`(i): return the bit at index i.

▶ **Lemma 3** (Munro et al. [12]). *A bit vector of length n can be succinctly represented using $n + o(n)$ bits to support* `rank`, `select` *and* `access` *in $O(1)$ time.*

We will also require the compressed form of bit-vectors, where if the number of 1s is small, we are able to get away with using less space.

▶ **Lemma 4** (Patrascu [14]). *A bit vector of length n with m 1s can be represented using $\log \binom{n}{m} + O(\frac{n}{\log^c n} + m) \leq m \log \frac{n}{m} + O(\frac{n}{\log^c n} + m)$ bits for any constant c. The data structure supports* `rank`, `select`, `access` *in $O(1)$ time.*

We will be working with trees and thus will be discussing and using various tree operations. These will mainly be conversions between the nodes' numbers in the different traversals: pre-order, post-order, level-order. These operations can be done in $O(1)$.

▶ **Lemma 5** (He et al. [9]). *An ordinal tree on n nodes can be represented succinctly using $2n + o(n)$ bits and can support a variety of operations in $O(1)$ time. For the full list see table 1 in their paper.*

2.4 Orthogonal Range Queries

In these data structures, we store n d-dimensional points. The queries we wish to answer are: given a d-dimensional axis aligned rectangle $[p_1, p_2] \times [p_3, p_4] \ldots [p_{2d-1}, p_{2d}]$ (we use the closed intervals here in the definition, but when our points are integers it is easy to see how to support open or semi-open intervals as well), *emptiness*: does it contain a point? *count*: how many points does it contain? *reporting*: return each point. We say that the rectangle is k-sided if there is at most k coordinates that are finite (in general the coordinates p_i may be $\pm\infty$). When we do not mention how many sides, it is assumed that it is the maximum possible: $2d$.

When $d = 2$, Chan et al. [5] showed that we solve the emptiness problem:

▶ **Lemma 6.** *We can solve the 2d range emptiness queries using $O(n \log n \log\log n)$ bits of space and $O(\log\log n)$ query time or $O(n \log n)$ bits of space and $O(\log^\epsilon n)$ query time for any constant $\epsilon > 0$.*

When $d = 3$, Nekrich [13] showed that we may solve the emptiness and reporting problems:

▶ **Lemma 7.** *For n 3D points and constant $\epsilon > 0$, we may support 5-sided orthogonal reporting queries using $O(n \log n)$ bits of space and $O(k \log^\epsilon n)$ time or $O(n \log n \log\log n)$ bits of space and $O(k \log\log n)$ time, where k is the size of the output.*

By setting $k = 0$, we may achieve the complexities to emptiness queries as well.

2.5 Predecessor Queries

Given a set of numbers S in a universe U, we wish to answer the following query: `pred`(i), given an element i of U, return the largest element j of S that is smaller than i.

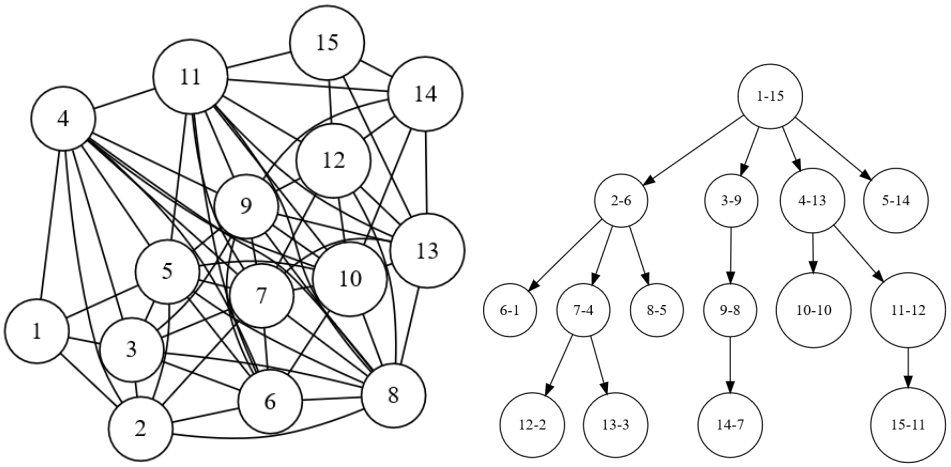
Though there have been a lot of work on this problem, we will only need one of the basic results by Willard [17] which gives a $O(N)$ word space solution to the problem with $O(\log \log U)$ query time.

▶ **Lemma 8.** *There is a data structure for the predecessor problem that uses $O(N)$ words of space and query time $O(\log \log U)$.*

3 Beer Paths in Proper Interval Graphs

In this section we investigate beer paths in proper interval graphs. We will be using the data structure of He et al. [9] as it supports the `dist` operation, which we will modify to account for the beer vertices. We begin with an example:

▶ **Example 9.** Consider the graph with the interval representation given by the bit string: 000001000101001110011011011111. This gives the vertex 1 a left endpoint at coordinate 1 and right index at coordinate 6 (the first 0 and first 1 in the sequence respectively). A graphical representation of the graph and the corresponding distance tree is given.



The first number in the node of the distance tree is the node's level-order number and the second its post-order traversal number.

Consider the shortest path between nodes 13 and 3. The shortest path algorithm given by He et al. [9] will give the path $13 \to 7 \to 3$. The problem would be easy if one of these nodes were a beer node, say node 7 then there would be far less to do. However consider the case that the only beer node were node 6, then a `beer_shortest_path` would be $13 \to 7 \to 6 \to 3$. On the other hand, if there were also a beer node at node 8, then we can take the path $13 \to 8 \to 3$.

3.1 Calculating Beer Distance

In the data structure of He et. al [9], we represent the proper interval graph G using a distance tree T. There is a bijection between the vertices of the graph G and the nodes of the tree - vertex v is mapped to the vth node in the tree in level-order. Thus by v we

will simultaneously refer to the vertex and the node in the distance tree. As the conversion between level-order ranks, pre-order ranks and post-order ranks in a tree are all $O(1)$ (and typically, nodes in a tree are referred by their pre-order ranks), we will implicitly convert between them as the situation requires. All the queries are reduced to tree operations and can be done in $O(1)$ time.

Since we will be building upon the `dist` and `shortest_path` queries, we will explain it in detail. The distance tree's parent child relationship is the following: for a node v, the parent of v is the smallest (indexed) node that is adjacent to v.

Let $u < v$, be the nodes involved in the `dist`/`shortest_path` query, and suppose that $\texttt{depth}(u) = k_1 \leq k_2 = \texttt{depth}(v)$. We repeatedly take the parent of v to form the following chain: $v_{k_1}, v_{k_1+1}, \ldots, v_{k_2} = v$, where v_j is at depth j. If $v_{k_1} < u$ then a shortest path is $u, v_{k_1+1}, \ldots, v_{k_2} = v$. If $v_{k_1} = u$, then the shortest path is $u = v_{k_1}, v_{k_1+1}, \ldots, v_{k_2} = v$. Finally if $v_{k_1} > u$, then a shortest path is $u, v_{k_1}, v_{k_1+1}, \ldots, v_{k_2} = v$. To compute the length without getting every node, we simply subtract the depths of u and v, use level-ancestor to find v_{k_1} and find which case we fall into to adjust the distance.

We note that this is only one of many possible shortest paths. To accommodate the beer vertices, we will investigate what all the possible shortest paths might look like. To this end, we will consider the following question: let $u < w < v$, is w on a shortest path between u and v? Equivalently, is $\texttt{dist}(u, v) = \texttt{dist}(u, w) + \texttt{dist}(w, v)$? In this case we say that w *preserves the distance*. Let $\texttt{post}(v)$ denote the post-order rank of a node in the tree. It is clear that if $\texttt{post}(u) < \texttt{post}(v)$, then u is to the left of v_{k_1}, so that $u < v_{k_1}$, and similarly for the reversed inequality. Finally, we note that for nodes on the same level of the tree, their post-order numbers are sorted. That is if $u < v$ are on the same level, then $\texttt{post}(u) < \texttt{post}(v)$ (and vice versa).

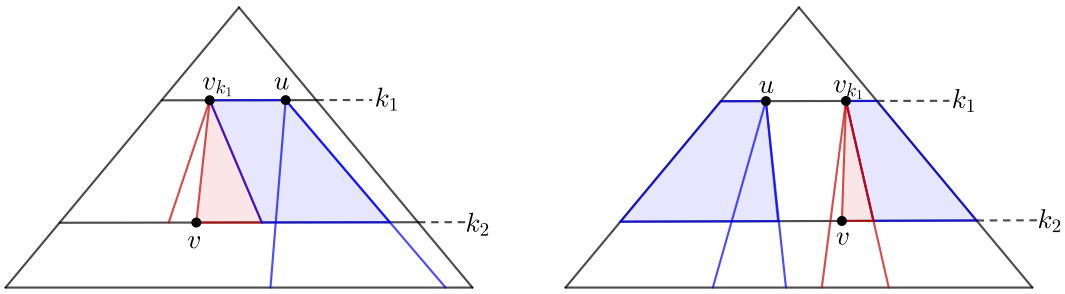


Figure 1 The union of the two shaded regions capture the nodes which preserve the distance in Lemma 10. The left is the case when $\texttt{post}(u) > \texttt{post}(v)$, and the right is the case when $\texttt{post}(u) < \texttt{post}(v)$. The blue region represents complete subtrees that are included, while the red represents the nodes to the right of the path to the root from v, used in subsection 3.3. Note the nodes on level k_1 to the left of u and on level k_2 to the right of v are not included.

▶ **Lemma 10.** *Let $u < w < v$ be 3 nodes in a proper interval graph. Suppose that $\texttt{post}(u) < \texttt{post}(v)$, then w is on a shortest path exactly when either $\texttt{post}(w) < \texttt{post}(u)$ or $\texttt{post}(w) > \texttt{post}(v)$ (that is w preserves the distance). If $\texttt{post}(u) > \texttt{post}(v)$, then w preserves the distance exactly when $\texttt{post}(v) < \texttt{post}(w) < \texttt{post}(u)$. Furthermore, if w does not preserve the distance, then the path passing through w increases the distance by 1. That is $\texttt{dist}(u, v) + 1 = \texttt{dist}(u, w) + \texttt{dist}(w, v)$.*

See figure 1 for a pictorial representation of the criteria; Now we can describe the process of determining the beer distance. The idea is to cover the nodes of the tree with 3 sets, and determine the best possible beer distance using beer vertices in each of the 3 sets. Finally we take the minimum of the 3. We will call the best vertex in each set a *candidate*.

First we note that if either $u, v \in B$, then we do not need to do anything and simply return $\texttt{dist}(u, v)$ (or $\texttt{shortest\_path}(u, v)$).

Candidate 1. The set of nodes is $\{w \in B; w > v\}$. We claim that the best beer node in this set is the smallest one. To show this, we will use the following lemma.

▶ **Lemma 11.** *Let $u < v < w$ be 3 nodes in a proper interval graph, then $\texttt{dist}(u, v) \leq \texttt{dist}(u, w)$. By symmetry, $\texttt{dist}(v, w) \leq \texttt{dist}(u, w)$.*

The node in $\{w \in B; w > v\}$ that minimizes the value of $\texttt{dist}(u, w) + \texttt{dist}(v, w)$ is of course the w of minimal index.

Candidate 2. The set of nodes $\{w \in B; w < u\}$. We take the largest node in the set as the candidate using Lemma 11.

Candidate 3. The set of nodes $\{w \in B; u < w < v\}$. By Lemma 10, we need to determine whether there exists a node such that either $\texttt{post}(u) < \texttt{post}(w) < \texttt{post}(v)$ or ($\texttt{post}(w) < \texttt{post}(u)$ or $\texttt{post}(w) > \texttt{post}(v)$), depending on $\texttt{post}(u) < \texttt{post}(v)$. If such a node exists, then it is the candidate, with distance $\texttt{dist}(u, v)$. If no such node exists, we may take any node as the candidate, with distance $\texttt{dist}(u, v) + 1$.

3.2 First Data Structure For Beer Distance

Here we discuss how to use the previous results to create a data structure for the queries. In this subsection, we discuss a relatively simple data structure which has decent run times. However, the space is dependent on $|B|$ the number of beer nodes and in the case that $|B| = \Theta(n)$ is large, the space bound is also unacceptably large. In the next subsection, we will show how to remove this dependence on $|B|$ at the cost of slightly worse run times.

To store the beer vertices, we store a bit vector B of length n so that $B[i] = 1$ if the ith vertex is a beer vertex. This uses $n + o(n)$ bits of space. As above we will assume that neither u nor v are beer vertices (and we can check by looking at $B[u], B[v]$).

Candidate 1. We use $\texttt{rank}(v)$ to find how many beer nodes are up to v. The smallest beer node that is larger than v can be found using $\texttt{select}(\texttt{rank}(v) + 1)$.

Candidate 2. Similarly to candidate 1, we find it by $\texttt{select}(\texttt{rank}(u))$.

Candidate 3. For every beer node w, we store it in a 2D range emptiness data structure using the coordinates $(w, \texttt{post}(w))$ (by our notation w is just the level-order number of the node w). In the case that $\texttt{post}(u) < \texttt{post}(v)$, we need the nodes such that $\texttt{post}(w) < \texttt{post}(u)$ or $\texttt{post}(w) > \texttt{post}(v)$ and $u < w < v$. This is translated to the rectangles $(u, v) \times (-\infty, \texttt{post}(u))$ and $(u, v) \times (\texttt{post}(v), \infty)$.

For the second case when $\texttt{post}(u) > \texttt{post}(v)$, we need the nodes that $\texttt{post}(v) < \texttt{post}(w) < \texttt{post}(u)$. This is the rectangle $(u, v) \times (\texttt{post}(v), \texttt{post}(u))$.

This suffices to determine the distance. To list out the path, we first determine which candidate to use. Candidates 1 and 2 can simply list out the path using two $\texttt{shortest\_path}$ queries. For candidate 3, we list out the path between u, v one step at a time, and, at each step, we consider the set V_k, which is an interval in level order. We note that the nodes preserving the distance is a prefix of this interval. Thus we find the first beer vertex in V_k,

and check if it preserves the distance, if so add it to the path and list out the path from there. Otherwise, we continue to the next level. Furthermore, as listing out the path for Candidate 3 is at most $O(\texttt{dist}(u,v))$ time, we may do this whenever $\texttt{dist}(u,v) = O(\log^{\epsilon} n)$ rather than spending the time on the orthogonal range search in the distance query. Thus we have the following theorem:

▶ **Theorem 12.** *A beer proper interval graph G can be represented using $3n + o(n) + O(|B| \log n)$ bits to support the interval graph queries plus* `beer_shortest_path` *in $O(1)$ time per vertex on the path and* `beer_dist`(u,v) *in $O(\min(\log^{\epsilon} n, \texttt{dist}(u,v)))$ time. If we increase the extra space to $O(|B| \log n \log\log n)$ bits, we may support* `beer_dist`(u,v) *in $O(\min(\log\log n, \texttt{dist}(u,v)))$ time instead.*

Proof. The space required is the distance tree of [9], a single bit vector for the beer nodes, and a single 2D range emptiness data structure. ◀

We note that if there are many beer vertices, so that $|B| \in \Theta(n)$, then the space usage would be $\Theta(n \log n)$. However if $|B|$ were small (ex. $|B| = O(n/\log^2 n)$), then this data structure will suffice.

3.3 Improved Data Structure for Beer Distance

Here we show how to improve the space usage of our specialized ranged emptiness query, so that it no longer has any dependence on $|B|$. Since we have the tree structure, the range emptiness can be reduced to checking whether a certain set of tree nodes have any beer nodes in them. In particular, as seen from the previous subsection, we need to support the following rectangles using $o(n)$ bits: 1) $(u,v) \times (-\infty, \texttt{post}(u))$, 2) $(u,v) \times (\texttt{post}(v), \infty)$, and 3) $(u,v) \times (\texttt{post}(v), \texttt{post}(u))$. We will call these type 1,2 and 3 rectangles.

To make our notation cleaner, we will use the depth of a node in the first coordinate of a rectangle. This means to include all the nodes on that level. To accomplish this, we simply find the first node on that level (in $O(1)$ time) and substitute its level-order number as the value to be used in the rectangle; similarly use the last node of a level for the right end point of the rectangle.

Fix $\Delta = \omega(1)$, and choose an index $1 \le i \le \Delta$ such that the number of nodes on levels $k = i \mod \Delta$ is minimized. We will call these levels *selected levels*, and the nodes on them *selected nodes*. Thus the number of selected nodes is $O(n/\Delta)$ by the pigeonhole principle. For each of these nodes, consider the subtree rooted at them that extends down to the next select level. We will build the contracted tree T' with these subtree as nodes, and the appropriate edges. A node in T' is a beer node if at least one of the original nodes in the corresponding subtree *except the root* is a beer node.

We use a bit vector to store which nodes in level order are selected. As the number of nodes is $O(n/\Delta) = o(n)$, this compressed bit-vector uses $o(n)$ bits of space. We store the contracted tree T' succinctly, using $2n/\Delta + o(n) = o(n)$ bits. The bit vector to mark which nodes of T' are beer nodes is also $n/\Delta = o(n)$ bits. Thus the total space for our contracted tree is $o(n)$ bits.

To support these rectangles, we first reduce the general case to one where both u, v are on a selected level.

▶ **Lemma 13.** *We may assume that the inputs u, v are on selected levels at the cost of $O(\Delta)$ extra time.*

We will now assume that both u, v are on selected levels. To deal with these queries, we will build the 2D range emptiness query on our contracted tree T' and the nodes on the selected levels in the same way. We wish to convert as much of the query to the 2D range emptiness on the contracted tree as possible. We will denote the corresponding node in the contracted tree to u by u'.

Type 1 rectangles. We convert $(u, v) \times (-\infty, \texttt{post}(u))$ to $[\texttt{depth}(u'), \texttt{depth}(v')) \times (-\infty, \texttt{post}(u'))$ in the contracted tree and the same rectangle $(u, v) \times (-\infty, \texttt{post}(u))$ in the selected nodes. We note that in T', we exclude all nodes on $\texttt{depth}(v')$ since in T this includes only the nodes on that level, but in T' this would include the subtrees as well, which extend down. We also change the left endpoint so that we include the subtrees to the left of u, but as we exclude the roots from those subtrees (in our decision to mark them as beer nodes or not), we do not include more nodes in our search than required.

Type 2 rectangles. In the type 2 rectangles, we note that unfortunately, the rectangle does not contain all the nodes in the subtree $v_{\texttt{depth}(u)}$, only those to the right of the path to v. It does however include the entire subtrees of all the nodes to the right of $v_{\texttt{depth}(u)}$. To handle these complete subtrees (and exclude the subtree rooted at $v_{\texttt{depth}(u)}$) we use the rectangle $[\texttt{depth}(u'), \texttt{depth}(v')) \times (\texttt{post}(v_{\texttt{depth}(u')}), \infty)$. Of course we may find $v_{\texttt{depth}(u)}$ using level-ancestor. We again handle the selected nodes using the same rectangle on them $(u, v) \times (\texttt{post}(v), \infty)$.

Finally, we need to handle the the nodes in the subtree rooted at $v_{\texttt{depth}(u)}$, to the right of the path to v and above the level of v. We start with the entire subtree of $v_{\texttt{depth}(u)}$, whose nodes are an interval in post-order. Then nodes w with $\texttt{post}(w) > \texttt{post}(v)$ are exactly the ones we want, except that all the subtrees to the right of v extend down to the bottom of the tree, rather than being cut off at the depth of v. Thus we need to handle them as well. The way to do this in encoded in the lemma below, where $k_1 = \texttt{depth}(u)$.

▶ **Lemma 14.** *Let v be a node in a tree T at depth k_2 and v_{k_1} be the ancestor of v at depth k_1, where both k_1, k_2 are selected levels with a fixed $\Delta > 0$. Let T' be the contracted tree as defined above. Then we are able to answer the query: does the rectangle $(v_{k_1}, v) \times (\texttt{post}(v), \infty)$ in either:*

$O(n/\Delta \log n) + o(n)$ additional space and $O(\log n)$ time.
$O(n/\Delta \log n) + n + o(n)$ additional space and $O(\log \log n)$ time.
Here we do not count the space taken by the tree T.

Type 3 Rectangles. Type 3 rectangles are similar to type 2 rectangles. As above, we use the same rectangle $(u, v) \times (\texttt{post}(v), \texttt{post}(u))$ for the selected nodes. We again use the rectangle $[\texttt{depth}(u'), \texttt{depth}(v')) \times (\texttt{post}(v_{\texttt{depth}(u')}), \texttt{post}(u')]$ to capture the complete subtrees that we wish to use. The incomplete subtree is exactly the same as in type 2, so we are able to apply lemma 14.

Thus putting everything together we have the following theorem:

▶ **Theorem 15.** *Let G be a beer proper interval graph. Fix Δ, then G can be represented using $3n + o(n) + O((n/\Delta) \cdot \log n)$ bits and can support* `adjacent`, `degree`, `neighbourhood`, `dist` *in $O(1)$ time,* `shortest_path`, `beer_shortest_path` *in $O(1)$ time per vertex on the path and* `beer_dist` *in $O(\Delta + \log n)$ time.*

In particular, if we take $\Delta = \log n$, then the space is $O(n)$ with time $O(\log n)$ and if we take $\Delta = f(n) \log n$ for some $f(n) = \omega(1)$, then the space is $3n + o(n)$ and the time is $O(f(n) \log n)$.

If we wish to further our trade off and improve the time, we must explicitly store P as in the proof lemma 14, and use the space inefficient (but time efficient) range query data structures. Thus we obtain:

▶ **Theorem 16.** *Let G be a beer proper interval graph. Fix Δ, then G can be represented using $4n + o(n) + O((n/\Delta) \cdot \log n \log\log n)$ bits and can support* `adjacent`, `degree`, `neighbourhood`, `dist` *in $O(1)$ time,* `shortest_path`, `beer_shortest_path` *in $O(1)$ time per vertex on the path and* `beer_dist` *in $O(\Delta + \log\log n)$ time.*

In particular, if we take $\Delta = \log\log n$, then the space is $O(n \log n)$ with time $O(\log\log n)$.

4 Beer Paths in Interval Graphs

In this section, we study how to compute and construct data structures for beer paths and beer distances. We will use the the data structure of Acan et al. [1] as a black box, and work with the distance tree T of He et al. [9]. The major difference between interval graphs and proper interval graphs is how adjacency can be checked. In a proper interval graph, for a vertex v, and its parent p in the distance tree, v is adjacent to every vertex between v and p. However, in interval graphs, this is not the case, and depending on the graph structure, any of those vertices can be adjacent or not adjacent to v.

4.1 Calculating Beer Distance

As in proper interval graphs, we begin by investigating the conditions in which nodes *preserve* the distance. For a node w, we say that w is $+k$ distance if $\texttt{dist}(u, v) + k = \texttt{dist}(u, w) + \texttt{dist}(v, w)$ (and thus preserving the distance is equivalent to being $+0$ the distance). So, using w as a beer node will add k to the optimal (non-beer path) distance. To do this, we will add one more condition to that of Lemma 10.

Let u be a node in T. As shown in the proof of lemma 13, the node on the next level that splits it in the same way as u is the largest neighbour of u. We denote this by $\texttt{last}(u)$. For example, in example 9, the largest neighbour of the node 8, is the node 13, as 8 is adjacent to node 13, but not node 14. Thus $\texttt{last}(8) = 13$.

▶ **Lemma 17.** *Let $u < v$ be vertices in a beer interval graph G with depths $\texttt{depth}(u) = k_1 \leq k_2 = \texttt{depth}(v)$. Consider the nodes $u < w < v$. Then we have the following two criteria:*

- *If $\texttt{post}(\texttt{last}(u)) < \texttt{post}(v)$, then either $\texttt{post}(w) > \texttt{post}(v)$ or $\texttt{post}(w) < \texttt{post}(\texttt{last}(u))$. If $\texttt{post}(\texttt{last}(u)) > \texttt{post}(v)$, then $\texttt{post}(v) < \texttt{post}(w) < \texttt{post}(\texttt{last}(u))$.*
- *If $\texttt{post}(w) < \texttt{post}(v)$, then $\texttt{post}(\texttt{last}(w)) > \texttt{post}(v)$ or $\texttt{post}(\texttt{last}(w)) < \texttt{post}(w)$. If $\texttt{post}(w) > \texttt{post}(v)$, then $\texttt{post}(v) < \texttt{post}(\texttt{last}(w)) < \texttt{post}(w)$.*

If w satisfies both criteria, then w preserves the distance. If w satisfies one criteria, then w is $+1$ distance and if w satisfies neither criteria, then w is $+2$ distance.

Again, we will find several sets that cover the beer nodes, and argue about the optimal beer node in these sets. We then take the minimum distance of these candidates. As before, we will assume that neither u nor v are beer nodes.

Candidate 1. This is the same as the proper interval graphs: $\{w \in B; w > v\} = \{w \in B; l_w > l_v\}$. We again claim that the best beer node is the smallest one. It turns out the exact same proof of Lemma 11 will work here.

Candidate 2. We wish to use the symmetric set of Candidate 1. Unfortunately, in interval graphs, this is not as simple. The set is $\{w \in B; r_w < r_u\}$. We note that this condition and the proper interval graph non-nesting condition gives $l_w < l_u$ so that $w < u$, and thus this is the right analogous set to consider. We claim that the best node is the node with the largest r_w in this set. This can be seen by reflecting the intervals of the vertices - equivalent to sorting them by the right endpoints instead. In this reflected graph, apply Lemma 11, and the result follows.

Candidate 3. The nodes $\{w \in B; u < w < v\}$. By Lemma 17, we can obtain the distance of the best beer node using the criteria in the lemma.

Candidate 4. The left over nodes. The nodes that do not belong to the previous candidate sets are w with: $l_w < l_v$ and $r_w > r_u$ and $l_w < l_u$. Thus these are the nodes with $l_w < l_u < r_w$, so they are adjacent to u. Formally, this is the set: $\{w \in B; w < u, r_w > r_u\}$. As these nodes are adjacent to u, all we need to check is their distance to v.

First consider the path from u to v. By the distance algorithm, we have the path to the root from v: $v_1, \ldots, v_{k_2} = v$, and furthermore suppose that k is the index such that $v_k \leq u < v_{k+1}$. The end of the path could look like either $u, v_{k+1}, \ldots$ or $u, v_k, v_{k+1}, \ldots$, depending on whether v_{k+1} is adjacent to u or not.

First assume that v_{k+1} is not adjacent to u, then for any possible candidate w, if w were adjacent to v_{k+1} (that is $\texttt{last}(w) \geq v_{k+1}$), then w preserves the distance (as $\texttt{dist}(w, v) = \texttt{dist}(u, v) - 1$). Otherwise, as $r_w > r_u > l_{v_k}$, w is adjacent to v_k and hence $\texttt{dist}(w, v) = \texttt{dist}(u, v)$ and w is $+1$ distance.

Next assume that v_{k+1} is adjacent to u. Then again for any candidate w, $l_w < l_u < l_{v_k+1} < r_u < r_w$, so w is adjacent to v_{k+1} and w is $+1$ distance. We note that w cannot be adjacent to v_{k+2} as in this case we would contradict that fact that v_{k+1} is the smallest node adjacent to v_{k+2}, by the parent relationship in T.

Finally we note that the only property of w that we used is that w is adjacent to u, and that if $x > u$ is adjacent to u, then w is also adjacent to x and hence we may relax the set to $\{w \in B; w < u, \texttt{last}(w) \geq \texttt{last}(u)\}$.

4.2 Data Structure for Beer Distance

We discuss how to use the previous results to create a data structure for the queries. We use the data structure of He et al. [9] which supports the interval graph queries in optimal time. This uses $n \log n + O(n)$ bits of space. We note that this data structure itself builds upon the data structure of Acan et al. [1]. We store a bit vector B as before, which stores which nodes are beer nodes in level-order. This take $n + o(n)$ bits.

Candidate 1. We handle this in exactly the same way as in proper interval graphs.

Candidate 2. We need to be able to find nodes in the mirrored graph.

▶ **Lemma 18.** *We can find the desired node in the mirrored graph using $n \log n + O(n)$ bits in $O(1)$ time.*

Candidate 3. We are able to handle this using 3D 5-sided orthogonal range emptiness data structures.

▶ **Lemma 19.** *We can check if a beer node satisfies the criteria of lemma 17 using a constant number of 3D 5-sided orthogonal range emptiness data structures. Thus the space/time requirements are either $O(n \log n)$ space and $\log^{\varepsilon} n$ time or $O(n \log n \log\log n)$ space and $\log\log n$ time.*

Candidate 4. We will use the following lemma:

▶ **Lemma 20.** *We can convert the criteria of candidate 4 into a constant number of 5-sided rectangles.*

Finally, to handle shortest paths, we use the reporting query rather than the emptiness query. When the reporting query returns the first point, we stop. After we find the best beer node, we list out the path using two `shortest_path` queries.

▶ **Theorem 21.** *Let G be a beer interval graph, with beer nodes B. The there exists a data structures using $2n \log n + O(n) + O(|B| \log n)$ bits that supports `degree`, `adjacent`, `dist` in $O(1)$ time, `neighbourhood`, `shortest_path` in $O(1)$ time per vertex in the path/neighbourhood, `beer_dist` in $O(\log^{\epsilon} n)$ time and `beer_shortest_path` in $O(\log^{\epsilon} n + d)$ time where d is the distance between the two vertices.*

Alternatively, we may increase the space from $O(|B| \log n)$ to $O(|B| \log n \log\log n)$ and replace the $\log^{\epsilon} n$ in `beer_shortest_path`, `beer_dist` with $\log\log n$.

5 Lower Bounds for Beer Interval Graphs

In this section, we will derive lower bounds for beer interval graphs and beer proper interval graphs. The lower bounds we will derive will be information theoretic, so that for a set of objects X, we will need at least $\log |X|$ bits in the worst case to represent any specific object.

First, we note that it is not interesting for beer interval graphs, since adding beer vertices can increase the lower bound by at most n bits. Since the lower bound for interval graphs is already $n \log n - o(n \log n)$ bits, the increase in space to account for the beer vertices is a lower order term and does not impact our data structures. For proper interval graphs however, the lower bound is $2n$ and thus it is natural to ask whether adding the beer vertices requires the full n bits to store them. That is, is the lower bound for beer proper interval graphs $3n$? In the main result of this section, we will show that it is not necessarily the case, and that if X were the set of beer proper interval graphs, then $\log |X| = n \log(4 + 2\sqrt{3}) - o(n) \approx 2.9n$.

In our case, we are interested in beer graphs, that is a graph G together with a set $B \subseteq V$ of beer vertices. We will refer to B as a beer vertex pattern. We will say that two beer graphs (G_1, B_1) and (G_2, B_2) are isomorphic (and thus are the same object) if there exists a bijection $f : V(G_1) \mapsto V(G_2)$ such that $(u, v) \in E(G_1) \Leftrightarrow (f(u), f(v)) \in E(G_2)$ and $u \in B_1 \Leftrightarrow f(u) \in B_2$. The first condition is the standard condition for two graphs to be isomorphic and the second condition says that this isomorphism also preserves beer vertices. Thus for two beer graphs to be isomorphic, the underlying graphs must also be isomorphic as well.

▶ **Example 22.** Suppose our graph class are cliques, then how many beer cliques are there? On n vertices, there is exactly one underlying graph $G = K_n$ on n vertices that is a clique. Thus it remains to see how many different beer vertex patterns we can have. By definition, if (K_n, B_1) were isomorphic to (K_n, B_2), then there exists an automorphism σ of K_n mapping vertices $u \in B_1$ to $\sigma(u) \in B_2$ bijectively, and thus $|B_1| = |B_2|$. Conversely, if $|B_1| = |B_2|$ then there exists a bijection σ that maps the elements of B_1 to B_2 and fixes every other

vertex. As the underlying graph is the complete graph K_n, this σ is also an automorphism of the underlying graph as well. Thus (K_n, B_1) is isomorphic to (K_n, B_2) exactly when $|B_1| = |B_2|$. The number of different ways to add beer nodes to a clique on n vertices is thus $n + 1$. ◀

As $(G_1, B_1) \cong (G_2, B_2)$ happens only when $G_1 \cong G_2$, it remains to develop the theory to compute the number of beer vertex patterns that are different when given a specific underlying graph G. Let $Aut(G)$ denote the automorphism group of a graph G. We will view $B \subseteq V$ as a vector $B \subseteq 2^n$ on the hypercube (where the i-th bit denotes whether the i-th vertex belong to the set or not), and $Aut(G)$ as a group that acts on 2^n. In this lens, two beer vertex patterns B_1, B_2 are the same if there exists a group element σ mapping B_1 to B_2, and thus B_1 and B_2 belong to the same orbit of this group action. The number of different beer vertex patterns is thus the number of orbits $|2^n/Aut(G)|$. To count the number of orbits, we will use the Polya enumeration theorem [15], which in its most basic form, states that if we denote $c(\sigma)$ as the number of cycles in σ when viewed as a permutation of $V(G)$, $|2^n/Aut(G)| = \frac{1}{|Aut(G)|} \sum_{\sigma \in Aut(G)} 2^{c(\sigma)}$.

5.1 Automorphism Groups of Proper Interval Graphs

Klavic and Zeman [10] showed that Aut(connected PROPER INT) = Aut(CATERPILLAR). A caterpillar graph/tree is a path together with a set of leaves that are adjacent some vertex on the path. In particular, the automorphism group of any particular connected proper interval graph is generated by 2 types of automorphisms. First are automorphisms that swap twin vertices - which corresponds to those that swap the leaves adjacent to the same vertex on the path of a caterpillar graph. In a proper interval graph, twin vertices are those that have the same set of maximal cliques. The second is an automorphism that reverses the proper interval graph, which corresponds to reversing the path of a caterpillar graph. This reversal corresponds to a reversal of the maximal cliques. Of course, for any particular graph, there may not be any twin vertices, and thus there are no automorphisms of the first type. As for the second type, it can only exist when the number of vertices in the maximal cliques are symmetrical - as the vertices in the first maximal clique are mapped to those in the last maximal clique etc.

We will assume that the maximal cliques are not symmetrical and thus no automorphisms of the second type exists. To see this, we may always desymmetrize the sequence of maximal cliques by adding one vertex to only the first maximal clique if necessary.

Now suppose that G is a connected proper interval graph. As being twin vertices are an equivalence relation, let $S_1, \ldots, S_h$ be the equivalence classes of twin vertices, that is $u, v \in S_i$ implies that u, v are twins. Let $k_i = |S_i|$ and we will say that vertices which have no twins are in a class of size 1, so that $\sum_i k_i = n$.

▶ **Lemma 23.** *Let G be a proper interval graph with twin vertex classes of sizes $|S_1| = k_1, \ldots, |S_h| = k_h$. Then $|2^n/Aut(G)| \leq (k_1 + 1)(k_2 + 1) \cdots (k_h + 1)$. This is an equality in the case that the graph is connected and the maximal cliques are non-symmetrical.*

For a proper interval graph, we will also refer to the above quantity as its weight, as the number of beer proper interval graphs would be the weighted sum of proper interval graphs.

5.2 Lower Bound

In the section, we will prove a tight lower bound on the number of beer proper interval graph. The main idea is to decompose a Dyck path (which is in more or less a one-to-one correspondence to proper interval graphs) in such a way that it preserves the weights.

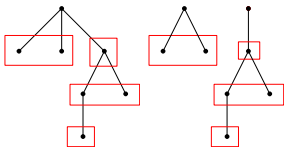


Figure 2 The twin vertex classes in the distance tree, and a way to decompose the tree.

To see the one-to-one correspondence, we see that by the interval graph recognition algorithm of Booth and Leuker [4], the PQ-tree of the maximal cliques is a single Q node, so that there are at most two ordering of maximal cliques representing each graph (one order and the reverse of that order). As each Dyck path gives a different sequence of maximal cliques, each graph can have at most two Dyck paths representing it.

For a particular proper interval graph, with twin vertex classes of sizes $k_1, \ldots, k_l$, the weight assigned to it is $\Pi_i(k_i + 1)$. Now consider the following blocking scheme for the distance tree associated with the proper interval graph: start at the root and continue in level-order, add the vertices to the block until either: the vertex has a different parent, or the vertex is not a leaf. In this manner, we consider the root as a sibling of its left child.

▶ **Lemma 24.** *In the above blocking scheme, two vertices u, v are in the same block if and only if they are twins.*

We may look at this in the same way by replacing the root with a dummy root and dropping the original root as the first child of the dummy root. This blocking scheme is illustrated in Figure 2.

Note that we do not consider the dummy root as part of our blocks and we can also view this as deleting the dummy root and consider the roots of this new forest as siblings. The second is our proposed way to decompose the tree into two trees while preserving all the blocks. If we consider the balanced parenthesis view of the tree (without the dummy root), we see that the sequences are the same ()()|((())()), but we cut it into two at the |. Precisely, | is at the first spot in the sequence such that the excess is 0 and the next two parentheses are ((. In the language of Dyck paths, this is the first time the path touches the x-axis and the next two steps are both up-steps.

Let $C(n)$ be the n-th Catalan number and the number of Dyck paths of length $2n$. The above decomposes the path into two subpaths. Let $L(n)$ be the number of paths of length n for the subpath to the left of |. and $R(n)$ be the number of paths of length n of the subpath to the right. We will also abuse notation and use $L(n), R(n)$ as the set of Dyck paths of the respective forms. Thus we have the recurrence $C(n) = \sum_{k=0}^{n} L(k)R(n-k)$.

Now we consider the weighted versions. Let $\bar{C}(n)$ be the sum of all Dyck paths with our weighting system. Similarly for $\bar{L}(n)$ and $\bar{R}(n)$. Because we preserve all the blocks with our split, we have the same recurrence $\bar{C}(n) = \sum_{k=0}^{n} \bar{L}(k)\bar{R}(n-k)$, which holds for $n \geq 1$. For $n = 0$, we see that $\bar{L}(0) = 0, \bar{R}(0) = 1$ and $\bar{C}(0) = 1$. Now it remains to compute $\bar{L}(n)$ and $\bar{R}(n)$.

▶ **Lemma 25.** $\bar{L}(n) = \sum_{l=0}^{n} \bar{C}(n-l-1)(l+2)$ *and* $\bar{R}(n) = \bar{C}(n) - \sum_{l=1}^{n}(l+1)\bar{R}(n-l)$.

Now consider the following generating functions. Let $f(x) = \sum_{n\geq 0} \bar{C}(n)x^n$, $g(x) = \sum_{n\geq 0} \bar{L}(n)x^n$ and $h(x) = \sum_{n\geq 0} \bar{R}(n)x^n$. The above recurrences says that these generating functions are linked and that we have a very nice closed form for f.

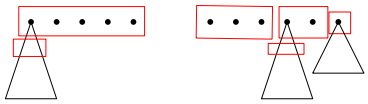


Figure 3 Decomposition of Dyck paths (as viewed as trees) of the forms L and R.

▶ **Lemma 26.** *Let* $b(x) = (1-x)^{-2}$. *Then we have* $f = gh + 1$, $g = f \cdot (b-1)$ *and* $h = f/b$. *Finally* $f = \left(1 - \sqrt{1-8x+4x^2}\right) / \left(4x - 2x^2\right)$.

We note that the sequence A108524 of OEIS [16] has the same generating function and thus $\bar{C}(n)$ is exactly A108524. Thus we can finally prove our desired lower bound for the number of beer proper interval graphs.

▶ **Theorem 27.** *The number of beer proper interval graphs on* n *vertices is asymptotically* $(4+2\sqrt{3})^n \cdot poly(n, 1/n)$. *Therefore to represent a beer proper interval graph* G *which is able to support* `adjacent` *and* `beer_dist` *will require at least* $\log(4+2\sqrt{3})n - o(n) \approx 2.9n$ *bits in the worst case.*

References

1 Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, and Srinivasa Rao Satti. Succinct encodings for families of interval graphs. *Algorithmica*, 83(3):776–794, 2021. doi:10.1007/s00453-020-00710-w.

2 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 62:1–62:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.62.

3 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001. doi:10.1145/502102.502107.

4 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

5 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the ram, revisited. In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.

6 Rathish Das, Meng He, Eitan Kondratovsky, J. Ian Munro, Anurag Murty Naredla, and Kaiyu Wu. Shortest beer path queries in interval graphs, 2022. arXiv:2209.14401.

7 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

8 G Hajós. Über eine art von graphen. int. *Math. Nachr*, 11:1607–1620, 1957.

9 Meng He, J. Ian Munro, Yakov Nekrich, Sebastian Wild, and Kaiyu Wu. Distance oracles for interval graphs via breadth-first rank/select in succinct trees. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.25.

10 Pavel Klavík and Peter Zeman. Automorphism Groups of Geometrically Represented Graphs. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 540–553, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2015.540.

11 C Lekkeikerker and Johan Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.

12 J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. *J. Algorithms*, 39(2):205–222, 2001. doi:10.1006/jagm.2000.1151.

13 Yakov Nekrich. New data structures for orthogonal range reporting and range minima queries. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1191–1205. SIAM, 2021. doi:10.1137/1.9781611976465.73.

14 Mihai Patrascu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.83.

15 G. Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica*, 68(none):145–254, 1937. doi:10.1007/BF02546665.

16 N. Sloane. The on-line encyclopedia of integer sequences. *Notices Amer. Math. Soc.*, 50:912–915, September 2003.

17 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space theta(n). *Inf. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

18 Peisen Zhang, Eric A. Schon, Stuart G. Fischer, Eftihia Cayanis, Janie Weiss, Susan Kistler, and Philip E. Bourne. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *Comput. Appl. Biosci.*, 10(3):309–317, 1994. doi:10.1093/bioinformatics/10.3.309.

Simon's Congruence Pattern Matching

Sungmin Kim
Department of Computer Science, Yonsei University, Seoul, Republic of Korea

Sang-Ki Ko
Department of Computer Science & Engineering, Kangwon National University, Chuncheon-si, Republic of Korea

Yo-Sub Han[1]
Department of Computer Science, Yonsei University, Seoul, Republic of Korea

Abstract

Testing Simon's congruence asks whether two strings have the same set of subsequences of length no greater than a given integer. In the light of the recent discovery of an optimal linear algorithm for testing Simon's congruence, we solve the Simon's congruence pattern matching problem. The problem requires finding all substrings of a text that are congruent to a pattern under the Simon's congruence. Our algorithm efficiently solves the problem in linear time in the length of the text by reusing results from previous computations with the help of new data structures called X-trees and Y-trees. Moreover, we define and solve variants of the Simon's congruence pattern matching problem. They require finding the longest and shortest substring of the text as well as the shortest subsequence of the text which is congruent to the pattern under the Simon's congruence. Two more variants which ask for the longest congruent subsequence of the text and optimizing the pattern matching problem are left as open problems.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases pattern matching, Simon's congruence, string algorithm, data structure

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.60

Funding This research was supported by the NRF grant (NRF-2020R1A4A3079947) and the AI Graduate School Program (No. 2020-0-01361) funded by the Korea government (MSIT).

1 Introduction

In the realm of string algorithms, subsequences are extensively studied with a lot of applications [3, 10, 14, 16]. Problems related to subsequences can be formulated using the string equivalence operation, which has many perks such as linear pattern matching time by the KMP algorithm [12] and linear construction of suffix trees [17]. For example, the longest common subsequence problem requires finding the longest subsequence of a string w_1 that is equal to some subsequence of a string w_2. Variants of the longest common subsequence such as the longest common increasing subsequence [1, 4] can also be interpreted as finding the longest common subsequence between three strings, where a string w_3 is obtained by sorting string w_1w_2. Meanwhile, recent findings on a congruence relation made it feasible to construct new problems that capture stronger properties related to subsequences. This congruence relation is called Simon's congruence defined by Simon in his study of piecewise testable languages [15]. The relation is based on the equivalence of the subsequence set rather than the equivalence of individual subsequences.

Given a positive integer k and a string w, let $\mathbb{S}_k(w)$ denote the set of all subsequences of w with length no more than k. We say that two strings w_1 and w_2 are $\sim_k$-congruent if the subsequence sets $\mathbb{S}_k(w_1)$ and $\mathbb{S}_k(w_2)$ are equal. For example, for $k = 2$, strings $w_1 = ababb$

[1] Corresponding author

and $w_2 = baba$ satisfy $w_1 \sim_2 w_2$ because the subsequence sets $\mathbb{S}_k(w_1)$ and $\mathbb{S}_k(w_2)$ are both $\{aa, ab, ba, bb, a, b, \lambda\}$. On the other hand, if $k = 3$, then there exists a string $u = abb$ that is a subsequence of w_1 but not a subsequence of w_2. Thus, we have $w_1 \not\sim_3 w_2$. The Simon's congruence decision problem asks whether or not $w_1 \sim_k w_2$.

For binary strings, Hébrard [9] presented an $O(|w_1| + |w_2|)$ algorithm. For an arbitrary alphabet Σ, Garel [7] suggested an $O(|\Sigma||w_1|)$ algorithm when $w_1 = w_2\sigma$ for a character σ; namely, w_2 is obtained from w_1 by removing the last character σ. Fleischer and Kufleitner [5] designed a normalization algorithm that reduces all $\sim_k$-congruent strings into a single string called the ShortLex normal form string and tested Simon's congruence with the string equivalence operation in $O(|\Sigma|(|w_1| + |w_2|))$ time. Thus, until recently, testing Simon's congruence for general conditions was a superlinear process, which was a giant hurdle in formulating general problems.

Recently, Barker et al. [2] improved Fleischer and Kufleitner's normalization algorithm and obtained a linear time algorithm. Later, Gawrychowski et al. [8] proposed data structures that help find the maximum k value such that $w_1 \sim_k w_2$ in linear time. Since the decision and the optimization problems for Simon's congruence can be solved in linear time, we are now ready to apply Simon's congruence to practical contexts such as pattern matching.

While Gawrychowski and his co-authors [8] suggested an optimal algorithm for optimizing k for Simon's congruence, they also proposed three open problems, where two are based on Simon's congruence and the other is based on a variant of Simon's congruence. The first problem, named LangSimK, is a membership testing problem. Given a set S of strings, which is either regular or context-free, a string w and an integer k, the problem asks to decide if there is a string $x \in S$ such that $w \sim_k x$. Kim et al. [11] proved that the problem is NP-complete in general and presented an efficient algorithm when the alphabet size is fixed.

▶ **Problem 1** (Gawrychowski et al. [8]). *The remaining two problems are as follows:*

- *Simon's Congruence Pattern Matching (MatchSimK): Given a pattern* `P`*, a text* `T`*, and an integer k, find all substrings of* `T` *that are $\sim_k$-congruent to* `P`*.*
- *Subsequence Set Inclusion Problem (SubseqSetInclusion): Given two strings w_1 and w_2, find the maximum integer k such that $\mathbb{S}_k(w_1) \subseteq \mathbb{S}_k(w_2)$.*

We tackle MatchSimK from the string pattern matching perspective, and design efficient algorithms.

Our contributions

We solve MatchSimK in linear time in the size of a text `T`. Our linear-time algorithm relies on data structures called X-trees and Y-trees that reduce the number of testings and reuse the computation results of the previous computations. Then, we study possible variants of MatchSimK with different objectives: The first two variants are called the longest congruent substring problem (LCongStrK) and the shortest congruent substring problem (SCongStrK).

▶ **Problem 2.** *Since we aim to extend problems based on the string equivalence operation into problems that use Simon's congruence, we first modify the longest common substring problem to use Simon's congruence.*

- *Longest Congruent Substring (LCongStrK): Given a pattern* `P`*, a text* `T`*, and an integer k, find a longest substring of* `T` *that is $\sim_k$-congruent to* `P`*.*
- *Shortest Congruent Substring (SCongStrK): Given a pattern* `P`*, a text* `T`*, and an integer k, find a shortest substring of* `T` *that is $\sim_k$-congruent to* `P`*.*

LCongStrK immediately extends the longest common substring problem by specifying that the returned substring of T should be $\sim_k$-congruent to P, instead of being equal to some substring of P. On the other hand, SCongStrK is interesting because its string equivalence counterpart, the shortest common substring problem, is nonsensical. The next set of problems, the longest congruent subsequence problem (LCongSeqK) and the shortest congruent subsequence problem (SCongSeqK), modifies LCongStrK and SCongStrK to consider subsequences as follows:

▶ **Problem 3.** *The listed problems also extend the longest common subsequence problem.*

- *Longest Congruent Subsequence (LCongSeqK): Given a pattern* P*, a text* T*, and an integer k, find a longest subsequence of* T *that is $\sim_k$-congruent to* P*.*
- *Shortest Congruent Subsequence (SCongSeqK): Given a pattern* P*, a text* T*, and an integer k, find a shortest subsequence of* T *that is $\sim_k$-congruent to* P*.*

We solve the LCongStrK, SCongStrK SCongSeqK problems, and leave the LCongSeqK problem as an open problem.

2 Preliminaries

String Notations

For a string w over an alphabet Σ, $\texttt{alph}(w)$ denotes the set of characters that appears in w. The length $|w|$ of w is the number of characters in w. We denote the empty string as λ. Given a string $w = w[0]w[1]\cdots w[n-1]$ over Σ such that $w[i] \in \Sigma$ for $0 \le i < n$, we assign a *space position* of w from the space right before $w[0]$ as 0 toward the space right after $w[n-1]$ as n; namely, we map $0, 1, \ldots, n$ to the corresponding positions. Then, we call $w[i]$ the corresponding character of a space position i. Given two space positions i, j such that $i \le j$, we define a string $w[i]w[i+1]\ldots w[j-1]$ to be a *substring* of w, which is denoted by $w[i:j]$. For convenience, we express the last character of w as $w[-1]$ and the substring $w[0:|w|-1]$ as $w[:-1]$. A string $u = w[a_1]w[a_2]\cdots w[a_i]$ is a *subsequence* of w if $0 \le a_1 < a_2 < \cdots < a_i \le |w|-1$. The substring relation is written as $u \prec_s w$ and the subsequence relation is written as $u \prec w$. The set of subsequences of length at most k is denoted by $\mathbb{S}_k(w) = \{u \in \Sigma^* \mid u \prec w, |u| \le k\}$. Finally, the *reversal* $w^R = w[|w|-1]w[|w|-2]\cdots w[2]w[1]w[0]$ of string w is the string obtained by concatenating characters of w in the reverse order of w.

Rankers and Coordinates

A function *ranker* is defined as follows: given a string w, a space position i of w, and a character $\sigma \in \Sigma$, $R(w, i, \sigma)$ returns a space position of w whose corresponding character is the nearest occurrence of σ from i towards a specific direction [5, 13, 18]. There are two types of ranker directions; from left to right (X-type) and from right to left (Y-type). Thus, an *X-ranker* $R_X(w, i, \sigma)$ returns a space position $j > i$ if $w[j-1] = \sigma$ and the substring $w[i:j-1]$ does not contain σ. If such a position does not exist, $R_X(w, i, \sigma) = \infty$. Similarly, a *Y-ranker* $R_Y(w, i, \sigma)$ returns a space position $j < i$ if $w[j] = \sigma$ and the substring $w[j+1:i]$ does not contain σ. When the output is not defined, then $R_Y(w, i, \sigma) = -1$. For example, for $w = aabcbabc$, we have $R_X(w, 2, a) = 6$ since $w[2:5] = bcb$ does not contain a and $w[5] = a$.

A *ranker chain* $R(w, i, x)$ is a ranker that takes a string instead of a single character for its third argument. Ranker chains can be recursively defined as $R_X(w, R_X(w, i, x[0]), x[1:|x|])$ or $R_Y(w, R_Y(w, i, x[-1]), x[:-1])$. Semantically, $R_X(w, 0, x)$ refers to the smallest space

position j such that $x \prec w[0:j]$. Similarly, $R_Y(w, |w|, x)$ is the largest space position j such that $x \prec w[j:|w|]$. For example, the X-ranker value for $w = ababaab$, $i = 0$, and $x = abb$ is $R_X(w, 0, abb) = R_X(w, 1, bb) = R_X(w, 2, b) = 4$.

Fleischer and Kufleitner [5] defined the *X- and Y-coordinates* of space position i of string w to be the length of the shortest string x such that $R_X(w, 0, x) = i + 1$ and $R_Y(w, |w|, x) = i$, respectively. We denote each as $X(w, i)$ and $Y(w, i)$. For the same example $w = ababaab$, we have $X(w, 6) = 3$ because $x = bbb$ is the shortest string such that $R_X(w, 0, x) = 6$. Kim et al. [11] extended the notion of coordinates by defining *X- and Y-vectors*, which are Σ-indexed arrays of potential X- and Y-coordinates if such a character is inserted at space position i. Formally, $\overrightarrow{X}(w, i)[\sigma] = X(w[0:i]\sigma w[i:|w|], i)$ and $\overrightarrow{Y}(w, i)[\sigma] = Y(w[0:i]\sigma w[i:|w|], i)$. Computing $\overrightarrow{X}(w, i{+}1)$ from $\overrightarrow{X}(w, i)$ or computing $\overrightarrow{Y}(w, i)$ from $\overrightarrow{Y}(w, i{+}1)$ is completed in $|\Sigma|$ time by incrementing the cell corresponding to character $w[i]$ by one and then setting the value of all cells that exceed the value at cell $w[i]$ to that value. Recall our example $w = ababaab$. We have $\overrightarrow{X}(w, 6) = [5, 3]$ because $X(ababaaab, 6) = 5$ and $X(ababaabb, 6) = 3$ If we compute $\overrightarrow{X}(w, 7)$ from $\overrightarrow{X}(w, 6)$, we first set $\overrightarrow{X}(w, 7)[b] = \overrightarrow{X}(w, 6)[b] + 1$ because $w[6] = b$. Since $\overrightarrow{X}(w, 6)[a] = 5$ exceeds $\overrightarrow{X}(w, 7)[b] = 4$, we set $\overrightarrow{X}(w, 7)[a] = \overrightarrow{X}(w, 7)[b]$. We refer to this single-step vector computation as one *iteration* of the vector computation and denote the function as $\mathrm{Iter}(\overrightarrow{v}, \sigma)$ for a vector $\overrightarrow{v}$ and a character σ. Also, for an integer i we define a *uniform vector* $\overrightarrow{U}(i)$ as the vector with all cell values equal to i. Vector addition and subtraction follow the general definition for vector operations. Barker et al. [2] defined the *k-universality* as follows: A string w is k-universal if $\mathbb{S}_k(w)$ is the set of all strings of length no greater than k. An example of a 3-universal string over $\Sigma = \{a, b, c\}$ is *abcbacccba*.

Arch Factorization

Given a string w, the jth *arch* of w, written as $\mathtt{ar}_j(w)$, is the minimal substring of w that starts at the end of the $j-1$th arch and satisfies $\mathtt{alph}(\mathtt{ar}_i(w)) = \Sigma$. We denote the sum of the lengths of arches 1 to i by $\mathtt{ArchSum}(i, w) = \sum_{j=1}^{i} |\mathtt{ar}_j(w)|$ and define $\mathtt{ArchSum}(0, w) = 0$. If no such space position $j > \mathtt{ArchSum}(i, w)$ satisfies $\mathtt{alph}(w[\mathtt{ArchSum}(i, w) : j]) = \Sigma$, then the integer i, which is the number of arches of w, becomes the *universality index* $\iota(w)$ [2] of w. The universality index refers to the maximum k for which w is k-universal. We call the string $w[\mathtt{ArchSum}(\iota(w), w) : |w|]$ – the suffix of w that starts at the space position $\mathtt{ArchSum}(\iota(w), w)$ – the *rest*, and denote it as $\mathtt{rest}(w)$. Finally, w can be decomposed as $\mathtt{ar}_1(w)\mathtt{ar}_2(w)\cdots\mathtt{ar}_{\iota(w)}(w)\mathtt{rest}(w)$. This factorization scheme is called the *arch factorization* of w [9]. The arch factorization defines the *modus* of a string such that $\mathtt{modus}(w) = \mathtt{ar}_1(w)[-1]\mathtt{ar}_2(w)[-1]\cdots\mathtt{ar}_{\iota(w)}(w)[-1]$. For example, if we have $w = aacabccbcbaacbcbc$ over $\Sigma = \{a, b, c\}$, we find $\mathtt{ar}_1(w) = aacab$, $\mathtt{ar}_2(w) = ccbcba$, and $\mathtt{ar}_3(w) = acb$. Since there cannot be any more arches, we set $\iota(w) = 3$ and $\mathtt{rest}(w) = cbc$. Thus, we have $w = \mathtt{ar}_1(w)\mathtt{ar}_2(w)\mathtt{ar}_3(w)\mathtt{rest}(w)$ and $\mathtt{modus}(w) = bab$. Note that we can apply arch factorization to w^R as well as w. We call the arches of w and w^R the *X-arches* and *Y-arches* of w, respectively. From a pair of overlapping X-arch and Y-arch, we construct an *arch link*, which is the maximal substring of w where the two arches overlap. They are called arch links because X- and Y-arches are chained together by a series of arch links. There are two types of arch links. We define *YX-links* of w as the substrings $w[\mathtt{ArchSum}(i, w) : |w| - \mathtt{ArchSum}(\iota(w) - i, w^R)]$ for all nonnegative integers $i \leq \iota(w)$. Likewise, we define *XY-links* of w as the substrings $w[|w| - \mathtt{ArchSum}(\iota(w) - i, w^R) : \mathtt{ArchSum}(i+1, w)]$ for all nonnegative integers $i < \iota(w)$. For YX-links, the Y-arch comes before the X-arch, while for XY-links, the X-arch comes before the Y-arch. Using our example $w = aacabccbcbaacbcbc$, the arch factorization of the

reverse of w results in $\mathtt{ar}_1(w^R)^R = acbcbc$, $\mathtt{ar}_2(w^R)^R = cba$, $\mathtt{ar}_3(w^R)^R = abccb$, and finally $\mathtt{rest}(w^R)^R = aac$. Thus, we have YX-links $w[0:3]$, $w[5:8]$, $w[11:11]$, and $w[14:17]$. On the other hand, XY-links of w include $w[3:5]$, $w[8:11]$, and $w[11:14]$. A similar factorization scheme was used by Fleischmann et al. [6] as well.

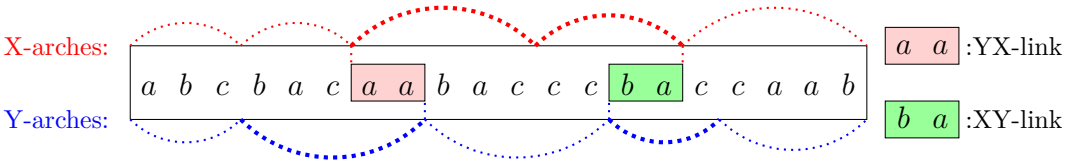


Figure 1 An illustration of X- and Y-arches of the string *abcbacaabacccbaccaab*. An XY-link, a YX-link, and the arches that produce them are highlighted to emphasize the difference between the two types of links.

Simon's Congruence

Given two strings w_1 and w_2, we say w_1 and w_2 are $\sim_k$-congruent if $\mathbb{S}_k(w_1) = \mathbb{S}_k(w_2)$. The congruence class defined by $\sim_k$ and a string w is written as $\text{Closure}_k(w) = \{x \in \Sigma^* \mid x \sim_k w\}$. Each congruence class $\text{Closure}_k(w)$ has a unique ShortLex normal form (SNF) string $\mathtt{ShortLex}_k(w)$ which is the lexicographically least string among the shortest strings in $\text{Closure}_k(w)$. For example, for an integer $k = 2$ and a string $w = babaabacaabba$, the SNF string of w is *abcab*. Thus, $\text{Closure}_k(w)$ is a set of strings u_1cu_2 such that u_1 and u_2 are strings over $\{a, b\}$ and both strings contain at least one a and one b. Fleischer and Kufleitner [5] presented a normalization algorithm that takes $O(|\Sigma||w|)$ time, which performs $O(|\Sigma|)$-time computations for each position of the input string. The algorithm was improved by Barker et al. [2] to run in $O(|w|)$ time.

▶ **Proposition 4** (Fleischer and Kufleitner [5]). *For a string w, an integer k, and a space position i of w, if $X(w,i) + Y(w,i) > k+1$, then $w \sim_k w[0:i]w[i+1:|w|]$. If no such i satisfies the above, then the string w is length-minimal among the $\sim_k$-congruent strings with w. Moreover, for all length-minimal strings $z \in \text{Closure}_k(w)$ and any string x with $|x| = |z|$, $x \sim_k z$ if and only if x can be obtained by repetitively rearranging characters in contiguous positions i, j such that $X(w,i) + Y(w,i) = k+1$, $Y(w,i) = Y(w,j)$ and $X(w,i) = X(w,j)$.*

Based on Proposition 4, both algorithms [2, 5] repeatedly remove characters in w to derive a length-minimal string in $\text{Closure}_k(w)$. Then, the algorithms rearrange contiguous characters that have the same pair of X- and Y-coordinates whose sums are $k+1$, and obtain a lexicographically smallest string among the shortest strings in $\text{Closure}_k(w)$. This two-step procedure is called the *ShortLex normalization algorithm*.

▶ **Proposition 5** (Barker et al. [2]). *Given an integer k and two strings w_1 and w_2, we can check whether or not $w_1 \sim_k w_2$ in $O(|w_1| + |w_2|)$ time. Moreover, for a string w, we can obtain $\mathtt{ShortLex}_k(w)$ in optimal time $O(|w|)$.*

Finally, in the context of MatchSimK, we say that a substring w of $\mathtt{T}$ is a *match* of $\mathtt{P}$ if w is $\sim_k$-congruent to $\mathtt{P}$.

3 Main Contributions

3.1 Simon's Congruence Pattern Matching

3.1.1 A simple algorithm

Before we design an efficient algorithm, let us consider a simple algorithm based on the number of possible matches.

▶ **Lemma 6.** *There exist a pattern* $\mathtt{P}$*, a text* $\mathtt{T}$ *and a number* k *such that the total number of matches of* $\mathtt{P}$ *is quadratic in* $|\mathtt{T}|$*.*

Note that a naive algorithm checking the congruence of all substrings of $\mathtt{T}$ would have a cubic running time in $|\mathtt{T}|$, since it takes $O(|\mathtt{T}|)$ time to check whether or not each substring is $\sim_k$-congruent to $\mathtt{P}$. With the goal of performing fewer tests of Simon's congruence, we present the following lemma.

▶ **Lemma 7.** *For a string* u *over* Σ*, let* $A = \{a \in \Sigma \mid u \sim_k ua\}$*. Then, for any string* $w \in A^*$ *and any character* $b \in \Sigma \setminus A$*, we have* $u \sim_k uw$ *and* $u \not\sim_k uwb$*.*

Lemma 7 implies that it is sufficient to check Simon's congruence exactly once for each starting space position i of $\mathtt{T}$. Specifically, we can test for a shortest match candidate $\mathtt{T}[i:m]$ ending at space position m for which the match candidate has a $\sim_k$-congruent character rearrangement of $\mathtt{ShortLex}_k(\mathtt{P})$ as a subsequence. First, assume that there exists a match $\mathtt{T}[i:l]$ for some space position l. Then, by Proposition 4, there exists a subsequence z of $\mathtt{T}[i:l]$ such that $|z| = |\mathtt{ShortLex}_k(\mathtt{P})|$ and $z \sim_k \mathtt{P}$. Thus, $l \geq m$. Moreover, if a subsequence of $\mathtt{T}[i:m]$ is $\sim_k$-congruent to $\mathtt{P}$, then we have $\mathbb{S}_k(\mathtt{P}) \subseteq \mathbb{S}_k(\mathtt{T}[i:m])$. It follows that if $\mathtt{T}[i:m] \not\sim_k \mathtt{P}$, then there exists some $u \in \mathbb{S}_k(\mathtt{T}[i:m])$ that is not a member of $\mathbb{S}_k(\mathtt{P})$, and in turn, no such l exists. Finally, if $\mathtt{T}[i:m] \sim_k \mathtt{P}$, then, we can extend our match result based on Lemma 7 to find all matches of $\mathtt{P}$ that start at i. Otherwise, there cannot be any match that starts at i.

On the other hand, we assume that $\mathtt{ShortLex}_k(\mathtt{P})$ takes the form of a stack of sets. Specifically, by Proposition 4, we can obtain $\mathtt{ShortLex}_k(\mathtt{P})$ from the shortest subsequence of $\mathtt{P}$ that is $\sim_k$-congruent to $\mathtt{P}$ by rearranging characters in substrings with indices that have the same X- and Y-coordinates whose sum equals $k+1$. Thus, we construct a stack by grouping the rearrangeable positions together into a set and repetitively pushing the sets into a stack starting with the set with the highest indices. Note that indices that are not rearrangeable with any other index each produce a singleton set. The peek operation returns the set at the top of the stack. However, the pop operation removes a given character from the set at the top of the stack and removes the set if the resulting set is empty. The stack representation of $\mathtt{ShortLex}_k(\mathtt{P})$ is convenient for finding a subsequence of $\mathtt{T}$ that is $\sim_k$-congruent to $\mathtt{P}$.

Based on these observations, we can solve MatchSimK in quadratic time in $|\mathtt{T}|$. For each space position i of $\mathtt{T}$, we find the minimal match candidate by finding the minimum space position m such that $\mathtt{T}[i:m]$ has a $\sim_k$-congruent character rearrangement of $\mathtt{ShortLex}_k(\mathtt{P})$ as a subsequence. We use the stack representation of $\mathtt{ShortLex}_k(\mathtt{P})$ to keep the testing time linear for each iteration. Thereon, we apply Lemma 7 to obtain all matches that start at space position i.

▶ **Theorem 8.** *Given a pattern* $\mathtt{P}$*, a text* $\mathtt{T}$*, and an integer* k*, we can find all space position pairs* (f, b) *of* $\mathtt{T}$ *for which* $\mathtt{T}[f:b] \sim_k \mathtt{P}$ *in* $O(|\mathtt{T}|(|\mathtt{T}| + |\Sigma|))$ *time.*

Note that $|\Sigma|$ is usually a constant. Even without the assumption, $|\mathtt{T}|$ dominates $|\Sigma|$ and thus the bound becomes $O(|\mathtt{T}|^2)$.

3.1.2 Can we do better?

Pondering on the simple algorithm, we identify two main causes that make the algorithm quadratic in $|\mathtt{T}|$. First, we report all matches of $\mathtt{P}$ by putting every match in a set, one-by-one. Recall that Lemma 6 proves that the number of matches in the worst-case is quadratic in $|\mathtt{T}|$. However, for a given space position f, Lemma 7 proves that all space positions b such that $\mathtt{T}[f:b] \sim_k \mathtt{P}$ are contiguous. Thus, despite the quadratic worst-case lower bound on the number of matches obtained in Lemma 6, we can design a faster algorithm if, instead of listing all matching substrings, we report the intervals $[f_1, f_2]$ and $[b_1, b_2]$ where all starting and ending space positions $(f, b) \in [f_1, f_2] \times [b_1, b_2]$ satisfy $\mathtt{T}[f:b] \sim_k \mathtt{P}$.

The remaining cause of the quadratic running time of the simple algorithm is that we need to check whether the candidate string that starts at each space position of $\mathtt{T}$ is $\sim_k$-congruent to $\mathtt{P}$ every time, which leads to a quadratic runtime with respect to $|\mathtt{T}|$. Using the concept of arch factorization, we can improve the running time by reusing substrings of SNF strings of other match candidates. The following lemma illustrates how arches can be used to find reusable substrings in $\mathtt{T}$.

▶ **Lemma 9.** *For a string w, and all non-negative integers $i \leq \iota(w)$, the X-vector at the right end of the ith X-arch $\mathtt{ar}_i(w)$ and the Y-vector at the left end of the ith Y-arch $\mathtt{ar}_i(w^R)^R$ are $[i+1, i+1, \ldots, i+1]$. In other words,*

$$\overrightarrow{X}\left(w, \mathtt{ArchSum}(i, w)\right) = \overrightarrow{Y}\left(w, |w| - \mathtt{ArchSum}(i, w^R)\right) = \overrightarrow{U}(i+1).$$

The key observation is that all X- or Y-vectors for space positions at the borders of every ith X- or Y-arch are static uniform vectors. Thus, for a space position i, the X- and Y-coordinates for i can be computed exclusively from the pair of X- and Y-arch which i is a member of. For example, let $w = aaaabaaaaaabaaaa$. Position $i = 8$ is covered by X-arch $\mathtt{ar}_2(w) = w[5:12]$ and Y-arch $\mathtt{ar}_2(w^R)^R = w[4:11]$. Using Lemma 9, we can directly let $\overrightarrow{X}(w, 5) = [2, 2]$ and $\overrightarrow{Y}(w, 11) = [2, 2]$. Finally, we can compute $X(w, 8) = 5$ and $Y(w, 8) = 4$ through only 6 calls of Iter(). Moreover, for a pattern $\mathtt{P}$ that is not k-universal, all matches x of $\mathtt{P}$ must satisfy $\iota(x) = \iota(\mathtt{P})$ and $\mathtt{alph}(\mathtt{rest}(x)) = \mathtt{alph}(\mathtt{rest}(\mathtt{P}))$ by the following lemma.

▶ **Lemma 10.** *For two strings w_1 and w_2, if $w_1 \sim_k w_2$ and w_1 is not k-universal, then $\iota(w_1) = \iota(w_2)$ and $\mathtt{alph}(\mathtt{rest}(w_1)) = \mathtt{alph}(\mathtt{rest}(w_2))$.*

Note that, for a k-universal pattern $\mathtt{P}$, we always have $\iota(\mathtt{P}) \geq \iota(\mathtt{ShortLex}_k(\mathtt{P})) = k$. Since we want to utilize Lemma 7 to extend borders of a minimal match, we first investigate the substrings of $\mathtt{T}$ that have the same universality index as $\mathtt{ShortLex}_k(\mathtt{P})$ following Lemma 10. Let w_1 and w_2 be substrings of $\mathtt{T}$ such that $\iota(w_1) = \iota(w_2)$. If a YX- or XY-link of w_1 is a YX- or XY-link of w_2, respectively, then we can reuse the substring of $\mathtt{ShortLex}_k(w_1)$ that corresponds to that arch link of w_1 in constructing $\mathtt{ShortLex}_k(w_2)$. Specifically, let the overlapping arch link be u. If u is a substring of the ith X-arch of w_1 as well as a substring of the jth X-arch of w_2, then the X-vectors for space positions of u in w_2 must differ by exactly $\overrightarrow{U}(j-i)$ from the X-vector values for space positions of u in w_1. The Y-arches' and Y-vectors' case is symmetric. Thus, if we let $w_1 = v_1 u x_1$ and $w_2 = v_2 u x_2$, then applying the repeated removal procedure of the ShortLex normalization algorithm on both strings will result in $z_1 = v_1' u' x_1'$ and $z_2 = v_2' u' x_2'$, where $v_1' \prec v_1$, $v_2' \prec v_2$, $x_1' \prec x_1$, $x_2' \prec x_2$, and finally $u' \prec u$.

It remains to find the borders of X- and Y-arches to determine which arch links can be reused. We solve this problem by building two trees, named the X-tree and Y-tree. An X-tree $T_X(\mathtt{T})$ or a Y-tree $T_Y(\mathtt{T})$ is a tree with at most $\iota(\mathtt{T})|\Sigma| + 1$ space positions of $\mathtt{T}$ as

nodes. The root is a virtual node that corresponds to space position ∞ for X-trees and -1 for Y-trees. For X-trees, each node is a right end point of an X-arch for some substring of $\mathtt{T}$. A node i's parent is the end point of the X-arch that starts at space position i. In other words, $\mathtt{prnt}(i) = \max\{R_X(\mathtt{T}, i, \sigma) \mid \sigma \in \Sigma\}$. Each node i maintains an interval of space positions $\mathtt{chld}(i)$ that would have i as the end point of an X-arch. Specifically, $\mathtt{chld}(i) = \{j \mid i = \max\{R_X(\mathtt{T}, j, \sigma) \mid \sigma \in \Sigma\}\}$. Finally, each node i holds a position $r(i)$ which is the minimum position for which $\mathtt{rest}(z) \prec \mathtt{T}[i : r(i)]$ for some $\sim_k$-congruent character rearrangement z of $\mathtt{ShortLex}_k(\mathtt{P})$. If there is no such position, then $r(i) = \infty$.

Conversely, a node in a Y-tree is a left end point of a Y-arch for some substring of $\mathtt{T}$. A node i's parent $\mathtt{prnt}(i)$ is the starting point of the Y-arch that ends at space position i. Formally, $\mathtt{prnt}(i) = \min\{R_Y(\mathtt{T}, i, \sigma) \mid \sigma \in \Sigma\}$. The child set $\mathtt{chld}(i)$ is the interval of space positions that would have i as a parent. Again, $\mathtt{chld}(i) = \{j \mid i = \min\{R_Y(\mathtt{T}, j, \sigma) \mid \sigma \in \Sigma\}\}$. Lastly, each node i holds a space position $r(i)$ which is the maximum position for which $\mathtt{rest}(z^R)^R \prec \mathtt{T}[r(i) : i]$ for some $\sim_k$-congruent character rearrangement z of $\mathtt{ShortLex}_k(\mathtt{P})$. If there is no such position, then $r(i) = -1$. The notation for an edge $\langle c, p\rangle$ of an X- or Y-tree follows the convention $\langle \text{child}, \text{parent}\rangle$, thus $c < p$ for X-trees and $c > p$ for Y-trees. Finally we abuse the notation $\mathtt{prnt}(i)$ for every space position i so that $\mathtt{prnt}(i) = j$ such that $i \in \mathtt{chld}(j)$.

Figure 2 is an example of an X-tree and a Y-tree constructed from $\mathtt{T} = abcbacaabacccbaccaab$. Arrows represent the edges $\langle c, p\rangle$ of each tree. Since the SNF string $abcabcabc$ of the pattern has $\mathtt{rest}(\mathtt{P}) = \mathtt{rest}(\mathtt{P}^R)^R = \lambda$, the value of $r(i)$ for each node i is itself.

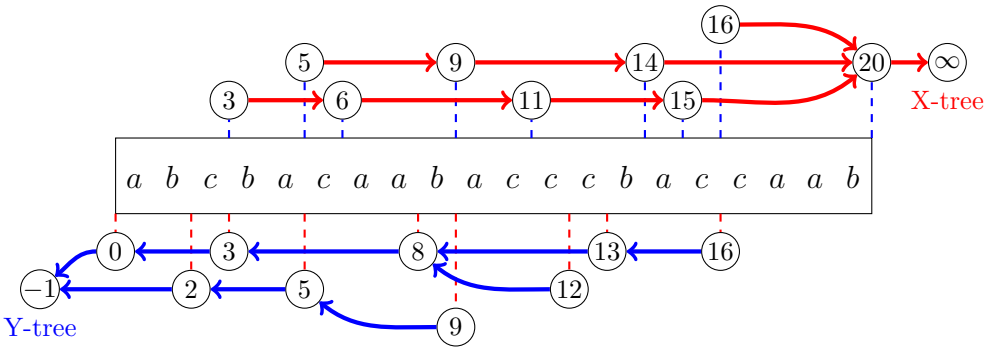


Figure 2 A pair of X-tree and Y-tree constructed from $\mathtt{T} = abcbacaabacccbaccaab$ and $\mathtt{ShortLex}_k(\mathtt{P}) = abcabcabc$ when $k = 3$. The X-tree is drawn in red and the Y-tree is drawn in blue.

We establish the following result – the characterization of the number of nodes in each interval defined by an edge of an X- or Y-tree – for a running time bound on the construction of X-trees and Y-trees.

▶ **Proposition 11.** *For an X-tree or Y-tree constructed from a text* $\mathtt{T}$*, and a space position* j *of* $\mathtt{T}$*, there are at most* $|\Sigma|$ *nodes in every half-open interval* $[\mathtt{prnt}(j), \mathtt{prnt}(\mathtt{prnt}(j)))$.

Building on the characterization from Proposition 11, Lemma 12 bounds the maximum number of edges that includes a given space position of $\mathtt{T}$ along with the number of nodes of the tree.

▶ **Lemma 12.** *For an X-tree (or a Y-tree) constructed from a text* $\mathtt{T}$*, there are at most* $|\Sigma|$ *edges* $\langle c, p\rangle$ *that satisfy* $c < i \le p$ *(or* $p \le i < c$*, respectively) for every position* i *of* $\mathtt{T}$*. Moreover, an X- or Y-tree has at most* $\iota(\mathtt{T})|\Sigma| + 1$ *nodes.*

Using the bound obtained above, we compute the worst-case time complexity of the construction of an X-tree.

▶ **Lemma 13.** *Given a preprocessed X-ranker (or Y-ranker) array and the stack representation of* $\texttt{ShortLex}_k(\texttt{P})$*, an X-tree (or Y-tree) for a text* T *can be constructed in* $O(|\texttt{T}||\Sigma|)$ *time.*

With our X- and Y-trees, we can obtain the X- and Y-arches for any minimal substring w of T starting at space position i such that $\iota(w) = \iota(\texttt{P})$ and $\texttt{rest}(\texttt{P}) \prec \texttt{rest}(w)$.

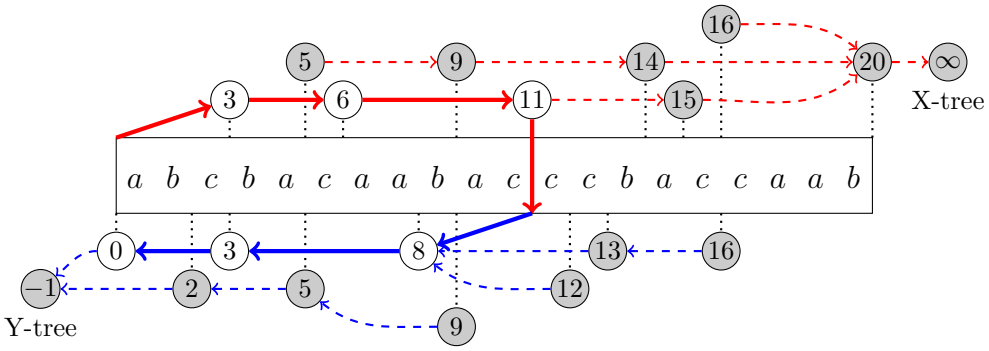


Figure 3 Using the X- and Y-trees $T_X(\texttt{T})$ and $T_Y(\texttt{T})$ to fetch the borders of arch links. The figure illustrates the traversal path for $\iota(\texttt{ShortLex}_k(\texttt{P})) = 3$ starting at space position 0. The thick arrows and the bright nodes indicate the traversal path of the X- and Y-tree.

Figure 3 illustrates the process of fetching the borders of X- and Y-arches from our running example $\texttt{T} = abcbacaabacccbaccaab$ and $\texttt{ShortLex}_k(\texttt{P}) = abcabcabc$. Starting with space position $i = 0$, we check the X-tree to obtain the node $T_X(\texttt{T}).\texttt{prnt}(i) = 3$. This step is indicated by the thick arrow starting at position 0 and ends at node 3. Afterwards, we repeat going up the X-tree for $\iota(\texttt{ShortLex}_k(\texttt{P})) - 1 = 2$ edges to visit node 6 and reach node 11. We read $T_X(\texttt{T}).r(11)$, but since $\texttt{rest}(\texttt{ShortLex}_k(\texttt{P})) = \lambda$, we stay at space position 11. By the definition of an X-tree, this is the smallest space position that satisfies $\texttt{rest}(\texttt{P}) \prec \texttt{rest}(w)$. From there on, we find the node of the Y-tree that has 11 in its child range. We find that node, which is 8, by following the thick arrow that starts at space position 11 and ends at node 8. Again, we traverse nodes 3 and 0 by repeatedly climbing the Y-tree for $\iota(\texttt{P}) - 1 = 2$ edges. The space positions that we have traversed are exactly the borders of X- and Y-arches for a minimal match candidate of P that starts at space position 0.

However, we need $\Sigma = \texttt{alph}(\texttt{P})$ to ensure that arches from T will line up with arches from P. Note that no substring w of T such that $\texttt{alph}(w) \neq \texttt{alph}(\texttt{P})$ will be a match of P. Thus, we can let $\Sigma = \texttt{alph}(\texttt{P})$ and slice T into maximal substrings $\texttt{T}'$ such that $\texttt{alph}(\texttt{T}') = \texttt{alph}(\texttt{P})$. Then, we can re-apply the same matching algorithm for each $\texttt{T}'$. Recall that arch links of w are determined by a pair of X-arch and Y-arch. Considering that X-vectors and Y-vectors can only be calculated in increasing or decreasing order of space positions, respectively, YX-links have two fixed vectors by Lemma 9. However, XY-links have no fixed vectors and thus we need a way to checkpoint and continue the application of the ShortLex normalization algorithm.

For a string w, let z be a length-minimal subsequence of w that is $\sim_k$-congruent to w. Also, let space positions i and j of w $(0 < i \leq j < |w|)$ be borders of some X- or Y-arch and map to space positions i' and j' of z such that

$$z \not\sim_k z[0 : i' - 1]z[i']z[i' - 1]z[i' + 1 : |z|],$$

$z \not\sim_k z[0:j'-1]z[j']z[j'-1]z[j'+1:|z|]$.

Then, we can decompose $z = v_1uv_2$ where $v_1 \prec w[0:i]$, $u \prec w[i:j]$, and $v_2 \prec w[j:|w|]$. Here, v_1 and v_2 do not share the same arches. Since Lemma 9 ensures that X-coordinate and Y-coordinate values of different arches are independent of each other, v_1 and v_2 can be independently obtained through the repeated removal procedure of the ShortLex normalization algorithm. Thus, we have $v_1w[i:j]v_2 \sim_k w$. Moreover, we have $\overrightarrow{X}(v_1w[i:j]v_2, |v_1|) = \overrightarrow{X}(z, |v_1|)$ as well as $\overrightarrow{Y}(v_1w[i:j]v_2, |v_1w[i:j]|) = \overrightarrow{Y}(z, |v_1u|)$. It follows that we can compute u if we surely know $\overrightarrow{X}(z, |v_1|)$ and $\overrightarrow{Y}(z, |v_1u|)$ without running the full ShortLex normalization algorithm on w. We use $\texttt{ShortLex}_k(w, i, j, \overrightarrow{X}, \overrightarrow{Y})$ to denote this checkpointed version of the ShortLex normalization algorithm. The second and third arguments are space positions i and j each at the border of some arch of w. They define the substring $w[i:j]$ that needs to be ShortLex normalized. The fourth and fifth arguments are X- and Y-vectors such that $\overrightarrow{X} = \overrightarrow{X}(z, |v_1|)$ and $\overrightarrow{Y} = \overrightarrow{Y}(z, |v_1u|)$ following our decomposition scheme from earlier.

Using our checkpointed algorithm, let strings w_1 and w_2 be substrings of $\texttt{T}$ that share some arch links. Let space positions i_1 and j_1 of w_1 and space positions i_2 and j_2 of w_2 mark the start and end of arch links that are shared between w_1 and w_2. For a length-minimal substring z of w_1 that is $\sim_k$-congruent to w_1, we let $z = v_1uv_2$ where $v_1 \prec w_1[0:i_1]$, $u \prec w_1[i_1:j_1]$, and $v_2 \prec w_1[j_1:|w_1|]$. Moreover, let arches $\texttt{ar}_{x_1}(w_1)$ and $\texttt{ar}_{y_1}(w_1^R)^R$ of w_1 and arches $\texttt{ar}_{x_2}(w_2)$ and $\texttt{ar}_{y_2}(w_2^R)^R$ be the X-arch and Y-arch that produces the arch link u.

First, while computing $\texttt{ShortLex}_k(w_1)$, we must checkpoint the progress of the ShortLex normalization algorithm for all borders of the arch links of w_1. If $w_1[i_1:j_1]$ is a YX-link, we associate u, $\overrightarrow{X}(z, |v_1u|) - \overrightarrow{U}(x_1)$, and $\overrightarrow{Y}(z, |v_1|) - \overrightarrow{U}(y_1)$ to the arch pair $(\texttt{ar}_x(w_1), \texttt{ar}_y(w_1^R)^R)$ using nodes in the X- and Y-tree. Later, when computing $\texttt{ShortLex}_k(w_2)$, we can skip the computation for $\texttt{ShortLex}_k(w_2, i_2, j_2, \overrightarrow{X}(w_2, i_2), \overrightarrow{Y}(w_2, j_2))$ and use u instead. Thus, we can decompose the length-minimal subsequence of w_2 that is $\sim_k$-congruent to w_2 as $v_1'uv_2'$, where

$$v_1' = \texttt{ShortLex}_k(w_2, 0, i_2, \overrightarrow{U}(1), \overrightarrow{Y}(z, |v_1|) - \overrightarrow{U}(y_1) + \overrightarrow{U}(y_2)),$$

$$v_2' = \texttt{ShortLex}_k(w_2, j_2, |w_2|, \overrightarrow{X}(z, |v_1u|) - \overrightarrow{U}(x_1) + \overrightarrow{U}(x_2), \overrightarrow{U}(1)).$$

On the other hand, if $w_1[i_1:j_1]$ is a XY-link, we only remember that the arch link defined by the arch pair $\texttt{ar}_x(w_1)$ and $\texttt{ar}_y(w_1^R)^R$ produces u, because $\overrightarrow{X}(w_2, j_2)$ and $\overrightarrow{Y}(z, i_2)$ are directly obtainable through Lemma 9. Thus, the length-minimal subsequence of w_2 that is $\sim_k$-congruent to w_2 is decomposed as

$$\texttt{ShortLex}_k(w_2, 0, i_2, \overrightarrow{U}(1), \overrightarrow{U}(y_2+1))u\texttt{ShortLex}_k(w_2, j_2, |w_2|, \overrightarrow{U}(x_2+1), \overrightarrow{U}(1)).$$

Returning to our running example on Figure 3, the minimal match candidates that start at space position 0 and space position 3 share the YX-link $w[6:8]$. This is because arches $T_X(\texttt{T}).\langle 6, 11\rangle$ and $T_Y(\texttt{T}).\langle 8, 3\rangle$ are traversed for both match candidates. This allows us to skip the computation for $w[6:8]$ if checkpoints for space positions 6 and 8 were saved beforehand. Note that further decomposition is possible if more checkpoints are available.

Now we are ready to efficiently tackle the Simon's congruence pattern matching problem. Let $\Sigma = \texttt{alph}(\texttt{P})$. If $\texttt{alph}(\texttt{T}) \not\subseteq \Sigma$, arches of $\texttt{T}$ will not align with arches of $\texttt{P}$. Thus, we first split $\texttt{T}$ into maximal substrings $\texttt{T}'$ of $\texttt{T}$ that satisfy $\texttt{alph}(\texttt{T}') \subseteq \Sigma$. Then, we repeat the following for every split substring $\texttt{T}'$ of $\texttt{T}$. We construct the X- and Y-trees for $\texttt{T}'$ as well as compute the X- and Y-ranker values beforehand. For each node n of the X-tree, we find the minimal match candidate by traversing the nodes of the X-tree and Y-tree. Then, we check whether the minimal match candidate is $\sim_k$-congruent to $\texttt{P}$ through the checkpointed

ShortLex normalization algorithm. Along the way, we store checkpoints for arch links that are not yet checkpointed to use in future computations. If the minimal match candidate is indeed congruent to P, then we extend the starting and ending points of the minimal match to find all matches of P which have n as the end space position of the first X-arch.

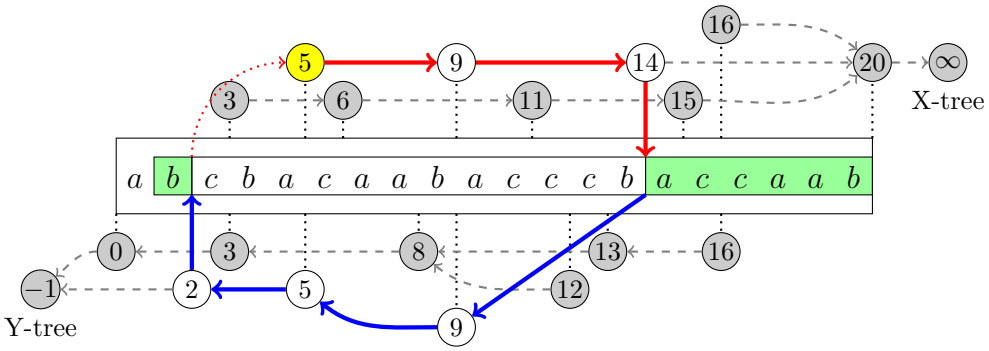


Figure 4 An illustration of the matching process where the current node is $i = 5$. If the minimal match candidate is a match of P, the front and back of the minimal match are extended to the colored boxes at the left and right of the minimal match.

We explain the matching procedure using our running example illustrated in Figure 4, highlighting the matching process for candidate strings for which the first arch ends at node $i = 5$ using Figure 4. Let $k = 3$. We have already computed sets A and B where $A = \{\sigma \mid \mathtt{P}\sigma \not\sim_k \mathtt{P}\}$ as well as $B = \{\sigma \mid \sigma\mathtt{P} \not\sim_k \mathtt{P}\}$. Since P is 3-universal, we have $A = B = \emptyset$. Recall that we are searching for all pairs (f, b) of starting positions such that $\mathtt{T}[f : b] \sim_k \mathtt{P}$. All substrings of T that start at space positions in $T_X(\mathtt{T}).\mathtt{chld}(i)$ will have its first arch end at node 5, using the same traversal path in the X- and Y-tree. Thus, we have space positions in $[1, 2]$ as our candidates for f.

Next, we use our traversal method explained with Figure 3 to obtain the arch link borders. Note that $T_Y(\mathtt{T}).r(2) = 2$, so we set our minimal string to start at space position 2, which is the maximum position in $T_X(\mathtt{T}).\mathtt{chld}(i) \cap [-1, T_Y(\mathtt{T}).r(2)]$. The -1 comes from $B = \emptyset$. Moreover, our minimal string ends at space position 14, which is the largest value observed during the X- and Y-tree traversal. The minimal match candidate is indicated by the box in the middle of Figure 4.

Now, we check whether $\mathtt{T}[2 : 14] \sim_k \mathtt{P}$. Since no checkpoints are saved at the moment, we compute the shortest subsequence of $\mathtt{T}[2 : 14]$ that is $\sim_k$-congruent to $\mathtt{T}[2 : 14]$ without using any checkpoints. By applying the first step of the ShortLex normalization algorithm, we obtain $z = cbacabacb$. With the subsequence z, we compute checkpoints for later use for edge pairs $(\langle 5, 9\rangle, \langle 5, 2\rangle)$, $(\langle 5, 9\rangle, \langle 9, 5\rangle)$, and $(\langle 9, 14\rangle, \langle 9, 5\rangle)$ as well as the edge-rest pair $(\langle 9, 14\rangle, \langle 14, 9\rangle)$. Popping all characters in z from the stack representation of $\mathtt{ShortLex}_k(\mathtt{P})$ verifies that $z \sim_k \mathtt{P}$.

Finally, using Lemma 7, we observe that all space positions no less than 14 are values for b that result in a match. Using Lemma 7 again on $\mathtt{T}^R$, we observe that space positions in $[-1, 2] \cap [1, 2]$ are values for f that result in a match. The extendable range for the starting and ending space positions are marked as the box on the left and right of Figure 4, respectively. Thus, we obtain the space position interval pair $([1, 2], [14, 20])$ for the iteration at node $i = 5$.

Now that we have an idea of how the algorithm works, we prove the algorithm solves MatchSimK in linear time in $|\mathtt{T}|$.

▶ **Theorem 14.** *Given a pattern* $\mathtt{P}$*, a text* $\mathtt{T}$*, and a number* k*, we can report all non-overlapping triples* $([f_1, f_2], [b_1, b_2], \text{offset})$ *such that for all space positions* $f \in [f_1, f_2]$ *and* $b \in [b_1, b_2]$*, we have* $\mathtt{T}[f + \text{offset} : b + \text{offset}] \sim_k \mathtt{P}$*. The computation takes* $O(|\mathtt{T}||\Sigma|(|\Sigma|^2 + k))$ *time.*

Note that we need an additional offset value that denotes the starting point of each $\mathtt{T}'$. This distinguishes matches of $\mathtt{P}$ from different $\mathtt{T}'$s and can be used to pinpoint the match from $\mathtt{T}$. Moreover, in practice, the size of an alphabet is regarded as constant. Thus, given a fixed k, MatchSimK can be solved in $O(|\mathtt{T}||\Sigma|^3) = O(|\mathtt{T}|)$ time.

3.2 Algorithms for Pattern Matching Variants

By altering the algorithm for MatchSimK to remember the length of the shortest or longest congruent substring while iterating every node of the X-tree, we can solve LCongStrK and SCongStrK.

▶ **Theorem 15.** *Given a pattern* $\mathtt{P}$*, a text* $\mathtt{T}$*, and a number* k*, we can report the longest and shortest substring of* $\mathtt{T}$ *that is* $\sim_k$*-congruent to* $\mathtt{P}$ *in* $O(|\mathtt{T}||\Sigma|(|\Sigma|^2 + k))$ *time.*

On the other hand, Proposition 4 shows that if a subsequence x of $\mathtt{T}$ is $\sim_k$-congruent to $\mathtt{P}$, then there must be a subsequence p' of x that is also $\sim_k$-congruent to $\mathtt{P}$ and has length $|\mathtt{ShortLex}_k(\mathtt{P})|$. Since there does not exist a shorter string that is $\sim_k$-congruent to $\mathtt{P}$, any algorithm that solves SCongSeqK must return a string of length $|\mathtt{ShortLex}_k(\mathtt{P})|$ if there exists a substring of $\mathtt{T}$ that is $\sim_k$-congruent to $\mathtt{P}$. In other words, the recognition of a shortest congruent subsequence of $\mathtt{T}$ can be done by popping characters of $\mathtt{T}$ from the stack representation of $\mathtt{P}$. Scanning $\mathtt{T}$ from left to right, we pop each character and mark the current space position if the corresponding character of the current space position is at the top of the stack. When the stack becomes empty, the marked positions yield a shortest subsequence of $\mathtt{T}$ that is $\sim_k$-congruent to $\mathtt{P}$.

▶ **Theorem 16.** *Given a pattern* $\mathtt{P}$*, a text* $\mathtt{T}$*, and a number* k*, we can report an instance of the shortest subsequence of* $\mathtt{T}$ *that is* $\sim_k$*-congruent to* $\mathtt{P}$ *in* $O(|\mathtt{P}| + |\mathtt{T}|)$ *time.*

Note that $\mathbb{S}_k(\mathtt{P}) \subseteq \mathbb{S}_k(\mathtt{T})$ if the search succeeds. Thus, one interesting idea is to use the algorithm for SCongSeqK in solving SubseqSetInclusion. However, the existence of an answer of SCongSeqK on strings $\mathtt{P} = w_1$ and $\mathtt{T} = w_2$ is only a sufficient condition for $\mathbb{S}_k(\mathtt{P}) \subseteq \mathbb{S}_k(\mathtt{T})$. Consider the example $w_1 = aabc$ and $w_2 = ccacbca$ and let $k = 2$. No character rearrangement of w_1 that is $\sim_k$-congruent to w_1 is a subsequence of w_2. Indeed, $\text{Closure}_k(w_1)$ is the singleton set $\{w_1\}$, while w_1 is not a subsequence of w_2. Thus, the algorithm for SCongSeqK will fail on this pair of strings. However, every element in the subsequence set $\mathbb{S}_2(w_1) = \{aa, ab, ac, bc, a, b, c, \lambda\}$ is a subsequence of w_2. This means that we need further characterization of the relation $\mathbb{S}_k(w_1) \subseteq \mathbb{S}_k(w_2)$ in order to solve SubseqSetInclusion. Although we conjecture that the solution for the Shortest Congruent Subsequence problem may be used in solving SubseqSetInclusion along with clever classification of w_1 and additional subprocedures, SubseqSetInclusion still remains open.

4 Conclusions

We have solved the open problem of finding all substrings of a text $\mathtt{T}$ that are $\sim_k$-congruent to a pattern $\mathtt{P}$ for an integer k proposed by Gawrychowski et al. [8]. We have devised tree data structures called X-trees and Y-trees to reuse results from previous computations and lower the asymptotic running time to be linear in the length of the text. Moreover, we have solved

two variants of the pattern matching problem using the efficient algorithm for MatchSimK as well as provided a linear algorithm that finds the shortest subsequence of the text that is $\sim_k$-congruent to the pattern. As future work, we plan to solve LCongSeqK, which is the remaining unsolved variant of MatchSimK. Moreover, we extend MatchSimK into an optimization problem, defined as the following:

▶ **Problem 17.** *Simon's Congruence Pattern Matching Optimization (ThreshMatchSimK): Given a pattern* `P`*, a text* `T`*, and a threshold* t*, find the maximum integer* k *for which there exist at least* t *congruent substrings of* `T` *that are* $\sim_k$*-congruent to* `P`*.*

Finally, we remark that the SubseqSetInclusion problem proposed by Gawrychowski et al. [8] is an interesting open problem to investigate.

References

1. Anadi Agrawal and Paweł Gawrychowski. A faster subquadratic algorithm for the longest common increasing subsequence problem. In *31st International Symposium on Algorithms and Computation*, volume 181 of *LIPIcs*, pages 4:1–4:12, 2020.
2. Laura Barker, Pamela Fleischmann, Katharina Harwardt, Florin Manea, and Dirk Nowotka. Scattered factor-universality of words. In *Developments in Language Theory – 24th International Conference*, volume 12086 of *Lecture Notes in Computer Science*, pages 14–28, 2020.
3. Richard Beal, Tazin Afrin, Aliya Farheen, and Don Adjeroh. A new algorithm for "the LCS problem" with application in compressing genome resequencing data. *BMC Genomics*, 17 (Supplement 4)(544):369–381, 2016.
4. Wun-Tat Chan, Yong Zhang, Stanley P. Y. Fung, Deshi Ye, and Hong Zhu. Efficient algorithms for finding a longest common increasing subsequence. *Journal of Combinatorial Optimization*, 13(3):277–288, 2007.
5. Lukas Fleischer and Manfred Kufleitner. Testing Simon's congruence. In *43rd International Symposium on Mathematical Foundations of Computer Science*, pages 62:1–62:13, 2018.
6. Pamela Fleischmann, Lukas Haschke, Annika Huch, Annika Mayrock, and Dirk Nowotka. Nearly k-universal words – investigating a part of Simon's congruence. In *Descriptional Complexity of Formal Systems – 24th International Conference, Proceedings*, volume 13439 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2022.
7. Emmanuelle Garel. Minimal separators of two words. In *4th Annual Symposium on Combinatorial Pattern Matching*, pages 35–53, 1993.
8. Paweł Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. In *38th International Symposium on Theoretical Aspects of Computer Science*, volume 187 of *LIPIcs*, pages 34:1–34:18, 2021.
9. Jean-Jacques Hébrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science*, 82(1):35–49, 1991.
10. James W. Hunt and M. Douglas McIlroy. An algorithm for differential file comparison. In *Computer Science Technical Reports 41*, 1975.
11. Sungmin Kim, Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. On Simon's congruence closure of a string. In *Descriptional Complexity of Formal Systems – 24th International Conference, Proceedings*, volume 13439 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2022.
12. Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.
13. Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Revised Papers from the 5th International Conference on Developments in Language Theory*, pages 239–250, 2001.

14 Jan Sedmidubský and Pavel Zezula. A web application for subsequence matching in 3d human motion data. In *19th IEEE International Symposium on Multimedia*, pages 372–373, 2017.

15 Imre Simon. Piecewise testable events. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pages 214–222, 1975.

16 Petra Surynková and Pavel Surynek. Application of longest common subsequence algorithms to meshing of planar domains with quadrilaterals. In *Mathematical Methods for Curves and Surfaces – 9th International Conference*, volume 10521 of *Lecture Notes in Computer Science*, pages 296–311, 2016.

17 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

18 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Logical Methods in Computer Science*, 5(3), 2009.

A Appendix

Algorithm 1 $O(|\mathtt{T}|^2)$–MatchSimK$(\mathtt{P}, \mathtt{T}, k)$ for Theorem 8.

Given: a pattern $\mathtt{P}$, a text $\mathtt{T}$, and a number k
Returns: all space position pairs (f, b) that satisfy $\mathtt{T}[f : b] \sim_k \mathtt{P}$
preprocess all possible X-ranker values
$s_p \leftarrow$ stack representation of $\mathtt{ShortLex}_k(\mathtt{P})$
$B \leftarrow \{b \in \Sigma \mid \mathtt{P} \not\sim_k \mathtt{P}b\}$
$M \leftarrow \emptyset$
for $i = 0, 1, \ldots, |\mathtt{T}|$ **do**
    $s'_p \leftarrow copy(s_p)$
    $m \leftarrow i$
    while s'_p is not empty **do**
        $\sigma \leftarrow \underset{a \in peek(s'_p)}{\arg\min}\ R_X(\mathtt{T}, m, a)$
        $m \leftarrow R_X(\mathtt{T}, m, \sigma)$
        pop σ from s'_p
    end while
    if $\mathtt{ShortLex}_k(\mathtt{P}) \sim_k \mathtt{T}[i : m]$ **then**
        find $\sigma \in B$ that minimizes $R_X(\mathtt{T}, m, \sigma)$
        for all space positions $j \in [m : R_X(\mathtt{T}, m, \sigma) - 1]$ **do**
            add (i, j) to M
        end for
    end if
end for
return M

Algorithm 2 X-tree construction for Lemma 13.

Given: preprocessed X-ranker values, $\texttt{ShortLex}_k(\texttt{P})$ s_p in stack form, and text $\texttt{T}$
Returns: an X-tree $T_X(\texttt{T})$ constructed from $\texttt{T}$
$T_X(\texttt{T}).\text{Nodes} \leftarrow \{\infty\}$
for $i = 1, 2, \ldots, \iota(\texttt{ShortLex}_k(\texttt{P}))$ **do**
    for $j = 0, 1, \ldots, |\texttt{ar}_i(\texttt{ShortLex}_k(\texttt{P}))| - 1$ **do**
        pop $\texttt{ar}_i(\texttt{ShortLex}_k(\texttt{P}))[j]$ from s_p
    end for
end for
for $i = 0, 1, 2, \ldots, |\texttt{T}| - 1$ **do**
    parent $\leftarrow \max_{\sigma:\sigma\in\Sigma}\{R_X(\texttt{T}, i, \sigma)\}$
    if parent $\notin T_X(\texttt{T}).\text{Nodes}$ **then**
        add parent to $T_X(\texttt{T}).\text{Nodes}$
        $s'_p \leftarrow copy(s_p)$
        $T_X(\texttt{T}).r(i) \leftarrow i$
        while s'_p is not empty **do**
            $S \leftarrow peek(s'_p)$
            $\sigma \leftarrow \arg\min_{c:c\in S}\{R_X(\texttt{T}, r(i), c)\}$
            $T_X(\texttt{T}).r(i) \leftarrow R_X(\texttt{T}, T_X(\texttt{T}).r(i), \sigma)$
            pop σ from s'_p
        end while
        $T_X(\texttt{T}).\texttt{chld}(\text{parent}) \leftarrow [i, i)$
    end if
    extend end point of $T_X(\texttt{T}).\texttt{chld}(\text{parent})$ by one
    if $i \in \text{Nodes}$ **then**
        $T_X(\texttt{T}).\texttt{prnt}(i) \leftarrow \text{parent}$
    end if
end for
return $T_X(\texttt{T})$

Algorithm 3 $O(|\mathtt{T}|)$–MatchSimK$(\mathtt{P}, \mathtt{T}, k)$ for Theorem 14.

Given: a pattern $\mathtt{P}$, a text $\mathtt{T}$, an integer k
Returns: a set S of triples where, for space positions f and b of $\mathtt{T}$, $\mathtt{T}[f : b] \sim_k \mathtt{P}$ if and only if there exists some element $e = ([f_1, f_2], [b_1, b_2], \text{offset})$ in S such that space positions $f - \text{offset} \in [f_1, f_2]$ and $b - \text{offset} \in [b_1, b_2]$
positions $\leftarrow \emptyset$
$s_p \leftarrow \mathtt{ShortLex}_k(\mathtt{P})$ in stack form
Slice $\mathtt{T}$ whenever $\mathtt{T}[i] \notin \mathtt{alph}(\mathtt{P})$
$A \leftarrow \{\sigma \mid \mathtt{P}\sigma \not\sim_k \mathtt{P}\}$
$B \leftarrow \{\sigma \mid \sigma\mathtt{P} \not\sim_k \mathtt{P}\}$
for all sliced substrings $\mathtt{T}'$ of $\mathtt{T}$ **do**
    offset $\leftarrow$ the start space position of $\mathtt{T}'$ in $\mathtt{T}$
    Map $\leftarrow$ empty map for saving vectors and substrings
    Preprocess X- and Y-ranker array
    Construct X-tree $T_X(\mathtt{T}')$ and Y-tree $T_Y(\mathtt{T}')$
    for all nodes $i \in T_X(\mathtt{T}')$.nodes **do**
        From i, go up the X-tree for $\iota(\mathtt{P}) - 1$ edges
        $j_1 \leftarrow T_X(\mathtt{T}').r(\text{current node})$
        if $j_1 = \infty$, break.
        From j_1, go up the Y-tree using $\iota(\mathtt{P})$ calls of $T_Y(\mathtt{T}').\mathtt{prnt}()$
        $n \leftarrow$ current node
        $j_2 \leftarrow \max(T_Y(\mathtt{T}').\mathtt{chld}(i) \cap [\max_{\sigma \in B} R_Y(\mathtt{T}', n, \sigma) + 1, n])$
        if no such value exists, continue.
        $z \leftarrow \mathtt{ShortLex}_k(\mathtt{T}'[j_2 : j_1])$ using the checkpoint mechanism and Map
        Save checkpoints for each arch link of $\mathtt{T}'[j_2 : j_1]$
        if $z \sim_k \mathtt{ShortLex}_k(\mathtt{P})$ **then**
            interval1 $\leftarrow T_X(\mathtt{T}').\mathtt{chld}(i) \cap [\max_{\sigma \in B} R_Y(\mathtt{T}', j_2, \sigma) + 1, j_2]$
            interval2 $\leftarrow [j_1, \min_{\sigma \in A} R_X(\mathtt{T}', j_1, \sigma) - 1]$
            add (interval1, interval2, offset) to positions
        end if
    end for
end for
return positions

Algorithm 4 Shortest Congruent Subsequence Problem for Theorem 16.

Given: a pattern $\mathtt{P}$, a text $\mathtt{T}$, an integer k
Returns: an instance of the shortest subsequence of $\mathtt{T}$ that is $\sim_k$-congruent to $\mathtt{P}$
$s_p \leftarrow \mathtt{ShortLex}_k(\mathtt{P})$ in stack form
sseq $\leftarrow \lambda$
for all indices $i = 0, 1, \ldots, |\mathtt{T}| - 1$ of $\mathtt{T}$ **do**
    if $\mathtt{T}[i] \in peek(s_p)$ **then**
        pop $\mathtt{T}[i]$ from s_p
        append $\mathtt{T}[i]$ to sseq
        if s_p is empty **then**
            return sseq
        end if
    end if
end for
return None

Simple Order-Isomorphic Matching Index with Expected Compact Space

Sung-Hwan Kim[1]
Pusan National University, Busan, South Korea

Hwan-Gue Cho[2]
Pusan National University, Busan, South Korea

Abstract

In this paper, we present a novel indexing method for the order-isomorphic pattern matching problem (also known as order-preserving pattern matching, or consecutive permutation matching), in which two equal-length strings are defined to match when $X[i] < X[j]$ iff $Y[i] < Y[j]$ for $0 \leq i, j < |X|$. We observe an interesting relation between the order-isomorphic matching and the insertion process of a binary search tree, based on which we propose a data structure which not only has a concise structure comprised of only two wavelet trees but also provides a surprisingly simple searching algorithm. In the average case analysis, the proposed method requires $\mathcal{O}(R(T))$ bits, and it is capable of answering a count query in $\mathcal{O}(R(P))$ time, and reporting an occurrence in $\mathcal{O}(\lg |T|)$ time, where T and P are the text and the pattern string, respectively; for a string X, $R(X)$ is the total time taken for the construction of the binary search tree by successively inserting the keys $X[|X|-1], \cdots, X[0]$ at the root, and its expected value is $\mathcal{O}(|X| \lg \sigma)$ where σ is the alphabet size. Furthermore, the proposed method can be viewed as a generalization of some other methods including several heuristics and restricted versions described in previous studies in the literature.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Compact Data Structure, String Matching, Order-Preserving Matching, Suffix Array, FM-index, Binary Search Tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.61

Acknowledgements The authors would like to thank anonymous reviewers for their helpful comments.

1 Motivation

In string matching, it is asked to find all the substrings of a given text string that match a pattern string. Instead of the standard setting where two strings are said to match if they are exactly the same, we can define a match of two strings in a different way depending on the target application. For example, parameterized string matching [2] and structural pattern matching [23] define the matching using bijective functions on the alphabet Σ, satisfying certain conditions to address pattern matching problems on program source codes and RNA sequences, respectively. In another variant of the matching problem [21], Cartesian tree is used to determine the matching of strings. Especially, in order-isomorphic pattern matching [19, 15], which is of interest in this paper, the relative ordering of characters is used; more formally, two equal-length strings X and Y are said to match, which we denote by $X =_o Y$, if $X[i] < X[j] \Leftrightarrow Y[i] < Y[j]$ for $0 \leq i, j < |X|$.

Despite the standard string matching, for which there are many space-efficient data structures such as [9] that require only a small amount of space, indexing strings for variant problems in a compact space has been considered quite challenging. Until Ganguly et al. [13]

[1] Currently working at Ca' Foscari University of Venice, Italy.
[2] Corresponding author

presented the first succinct index for the parameterized string matching problem, it had not been revealed whether there exists any data structure that is capable of efficiently processing string matching queries for such problems using asymptotically less than $\Theta(n \lg n)$ bits; only the indexing methods that exploit the conventional suffix tree and suffix array had been usually considered. After this breakthrough method was successfully invented, several succinct and compact data structures [12, 11, 16, 18, 17] for these problems have started to be actively developed.

One of the main concerns in developing a space-efficient index for such problems is suffix representation. Conventionally, we transform the suffixes into a certain form so that indexing these encoded suffixes enables us to search for pattern strings efficiently. Since the encoded suffixes should have several required properties to be effectively indexed, finding an appropriate representation is an important goal. Let $E(\cdot)$ be such an encoding function. When exploiting the conventional suffix tree and suffix array, it is sufficient to hold the property that $E(X \circ Y)$ has a prefix $E(X)$ for any strings X and Y where $\circ$ is the concatenation operator. However, when developing an index that is more space-efficient, we need more than that. Many space-efficient indexes rely on the compact representation of the relation between adjacent suffixes. For a text string $T[0..n-1]$, adjacent suffixes $T[i..n-1]$ and $T[i+1..n-1]$ are associated using this relation, which refers to the so-called *LF-mapping* or Ψ *function* in the literature, and they are used to access the sampled suffix array stored in a compact space. To do this, we must have a space-efficient way to characterize the operation that prepends a character; i.e. it is necessary to compactly represent the relation between the encoded strings $E(X)$ and $E(x \circ X)$ for any character x and string X.

It is more complicated for the order-isomorphic matching. As mentioned in the literature [11, 16], there are possibly many positions in which encoded characters differ when comparing $E(X)$ with $E(x \circ X)$. Moreover, an encoded character that has been already changed possibly changes again after performing prepending operations successively. This is quite different from much simpler problems such as parameterized string matching [2, 13], in which at most one character in an encoded string can change after prepending a character to the original string; and once an encoded character changes, it never changes anymore. Due to this complicated nature of the order-isomorphic matching, the ordering of the encoded strings is severely jumbled by prepending characters. Consequently, it is sophisticated to characterize the relation between adjacent suffixes, which makes its indexing problem much challenging.

For this reason, there has been a limited progress on developing indexes for the order-isomorphic matching that are space-efficient than the suffix tree-based method [6, 7]. Some methods used a filtering technique, in which two strings have the same signature only if they are order-isomorphic; e.g. up-down signature [3] and the rank information within a window of a restricted length [8]. However, these techniques may produce false positives in the filtering step, so the verification step must follow, which possibly involves substantial costs. It was also shown that one can report an occurrence of a pattern, if any, using an $\mathcal{O}(n \lg \lg n)$-bit data structure when the pattern length is restricted within a logarithmic size [10]. However, it had been an open problem whether there exists a compact index that does not restrict the searching capabilities until Ganguly et al. [11] presented the first $\mathcal{O}(n \lg \sigma)$-bit compact index for this problem, where n and σ is the text length and the alphabet size. The idea is to sample not only the suffix array but also the LF-mapping function, and the notion of LF-successor has been introduced to support this two-level sampling method. The underlying observation is the changing positions on the encoded string can be characterized by the rightmost position in which the change is made. They showed that this property can be

used to represent the relations between suffixes whose LF-mappings are adjacent. However, representing these relations were still rather sophisticated, which involved the classification of suffixes into four types according to the behavior of their LF-mapping, and an intricate structure including the topology of the suffix tree and a number of its subgraphs as well as many auxiliary bitvectors and wavelet trees storing the required information.

1.1 Contributions

In this paper, we develop a novel index with a concise structure and a simpler searching algorithm. More specifically, the main contributions can be summarized as follows.

1. **Novel suffix representation (Section 2):** We show how the order-isomorphic pattern matching can be described in terms of binary search trees, based on which we propose a novel suffix representation for this problem. With the new representation, each encoded character is a set, thereby not entirely replaced with another one but an integer is added to it when it changes by a prepending operation. This resolves many complications that arise in representing the relations between adjacent suffixes compactly.
2. **Simplicity (Sections 3 and 4):** The proposed data structure has a simple structure, which basically consists of only a pair of wavelet trees [9, 20] (plus a sampled suffix array if reporting each occurrence is needed); each wavelet tree can be implemented as a single bitvector. The searching algorithm is also surprisingly simple; the process can be done just by (properly) walking down on one wavelet tree, and then tracing up on the other wavelet tree.
3. **Expected compact space (Section 5):** The proposed data structure requires $\mathcal{O}(R(T))$ bits, where $R(T)$ is the time taken by root insertion of the characters of the input string T into a binary search tree in the backward fashion, which is $\mathcal{O}(n \lg \sigma)$ on average [24] where n and σ is the text length and the alphabet size, respectively. It has the same bound as that of the previous work in the average case analysis. Since the deviation of $R(T)$ seems small [24], many input strings are likely to be indexed within this bound.
4. **Fully-searchable index:** As far as we know, for this problem, this is the first fully-searchable index that does not require to keep the text string separately. Although adding the text string itself does not increase the space requirement asymptotically, we believe the proposed method can be a hint towards a succinct self-index for this problem.
5. **Extensibility and generalizability (Section 6):** The proposed method also has interesting connections with other related methods and problems. It can be viewed as a generalized method from existing methods addressing this problem including: (i) window-based order-isomorphic testing [8], (ii) filtering with up-down signatures [3], and (iii) indexing for length-restricted queries [10]. We can also effectively derive index structures for many variations of the order-isomorphic matching problem.

2 Binary Search Tree and Order-isomorphic Matching

In this section, we observe the relation between binary search trees and order-isomorphic matching. For brevity, until Section 5, we assume that all the characters of the string are distinct. We show that order-isomorphic matching of two strings can be represented as a match of their corresponding binary search trees. In this perspective, prepending a character at the beginning of a string is equivalent to inserting a node into the corresponding binary search tree as the root node. We establish some notations for characterizing a root insertion operation on a binary search tree (see Figure 1).

2.1 Binary Search Tree Generated from a String

Let $X[0..|X|-1]$ be a string. Consider a binary search tree $\mathsf{BST}(X)$ constructed by inserting a node with key $X[i]$ and value i for $0 \leq i < |X|$ in order; in other words, for each insertion $i = 0, \cdots, |X|-1$, we perform the conventional search for key $X[i]$ on the current binary search tree until we reach a leaf node, then we insert the new node having key $X[i]$ and value i as a child. For a node v of $\mathsf{BST}(X)$, we denote its value, left child and right child by $\mathsf{value}(v)$, $\mathsf{left}(v)$, and $\mathsf{right}(v)$, respectively. We also denote the subtree rooted at node v by $\mathsf{subtree}(v)$.

Consider two strings X and Y, and their corresponding binary search trees $\mathsf{BST}(X)$ and $\mathsf{BST}(Y)$. We say two binary search trees $\mathsf{BST}(X)$ and $\mathsf{BST}(Y)$ are value-identical, denoted by $\mathsf{BST}(X) =_v \mathsf{BST}(Y)$, if:

1. the number of nodes (tree sizes) are equal,
2. $\mathsf{value}(r_X) = \mathsf{value}(r_Y)$,
3. $\mathsf{subtree}(\mathsf{left}(r_X)) =_v \mathsf{subtree}(\mathsf{left}(r_Y))$, and
4. $\mathsf{subtree}(\mathsf{right}(r_X)) =_v \mathsf{subtree}(\mathsf{right}(r_Y))$.

where r_X and r_Y are the root node of $\mathsf{BST}(X)$ and $\mathsf{BST}(Y)$, respectively. To be well-defined, two empty binary search trees are also considered to be value-identical.

It is easy to see that X and Y are an order-isomorphic match iff $\mathsf{BST}(X)$ and $\mathsf{BST}(Y)$ are value-identical.

▶ **Lemma 1.** *$X =_o Y$ iff $\mathsf{BST}(X) =_v \mathsf{BST}(Y)$.*

Proof. We prove by induction. If $|X| = |Y| = 1$, it is trivial. Let us assume that $X =_o Y$ and $\mathsf{BST}(X) =_v \mathsf{BST}(Y)$. Consider two strings $X \circ x$ and $Y \circ y$ for some characters x and y. We can have $X \circ x =_o Y \circ y \Leftrightarrow \mathsf{BST}(X \circ x) =_v \mathsf{BST}(Y \circ y)$ immediately from the observation that, when a node with key x is inserted into a binary search tree $\mathsf{BST}(X)$ as its leaf node to obtain $\mathsf{BST}(X \circ x)$, the locus of the new leaf node is uniquely determined by the rank of x among $\{X[0], \cdots, X[|X|-1], x\}$. ◀

2.2 Root Insertion and Prepending operation

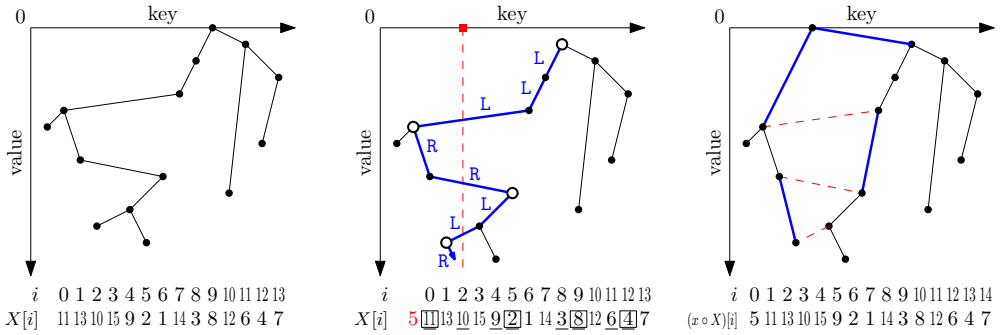
Let X be a string and x be a character. Consider a binary search tree $\mathsf{BST}(X)$. Suppose we prepend x at the beginning of X, and we want to have $\mathsf{BST}(x \circ X)$. We can observe that $\mathsf{BST}(x \circ X)$ can be obtained from $\mathsf{BST}(X)$ as follows:

1. Increment $\mathsf{value}(v)$ by 1 for every node v.
2. Insert a new node with key x and value 0 as the root node (as described in [24]).

Because the increment operation is applied to all nodes equally, the change can be characterized by how the root insertion is performed. Let us define $\mathsf{vpath}(x, X)$ to be the sequence of values of the nodes visited by searching x on $\mathsf{BST}(X)$; in other words, $\mathsf{vpath}(x, X)$ indicates the positions of the characters of X that correspond to the visited nodes when searching with the key x. We also define $\mathsf{branch}(x, X)$ to be a string over $\{\mathtt{L}, \mathtt{R}\}$ that indicates the branch taken at each node. More specifically, for $0 \leq i < |\mathsf{vpath}(x, X)|$,

$$\mathsf{branch}(x, X) = \begin{cases} \mathtt{L} & \text{if } x < X[\mathsf{vpath}(x, X)[i]] \\ \mathtt{R} & \text{if } x > X[\mathsf{vpath}(x, X)[i]] \end{cases} \tag{1}$$

Here, note that $X[\mathsf{vpath}(x, X)[i]]$ is the key of the node having the value $\mathsf{vpath}(x, X)[i]$.



(a) BST(X) (b) Prepending $x = 5$ (c) BST($x \circ X$)

Figure 1 Examples of a string X =11 13 10 15 9 2 1 14 3 8 12 6 4 7, and its corresponding binary search tree. (a) BST(X). Each dot is a node. x-axis represents the key, and y-axis represents the value of the nodes. (b) A character $x = 5$ is being prepended. Red dashed line indicates $x = 5$, and blue thick edges are the paths visited by searching for 5 on the tree. $\mathsf{vpath}(x, X)$=0 2 4 5 8 9 11 12 (positions in which $X[i]$ are underlined), $\mathsf{branch}(x, X) = \texttt{LLLRRLLR}$, and $\mathsf{turnpoint}(x, X) = \{0, 5, 9, 12\}$ (positions in which $X[i]$ are boxed). (c) BST($x \circ X$). The red dashed edges are removed ones, blue thick edges are newly established edges.

We also define $\mathsf{turnpoint}(x, X)$ to be the set of vertices at which the branching direction is switched. If the left branch (L) is taken at the root node, the root node is also included.

$$\mathsf{turnpoint}(x, X) = \{\mathsf{vpath}(x, X)[i] : 0 \le i < p, \mathsf{branch}(x, X)[i-1] \neq \mathsf{branch}(x, X)[i]\} \quad (2)$$

where $p = |\mathsf{branch}(x, X)|$ and for convenience we assume $\mathsf{branch}(x, X)[-1] = \texttt{R}$.

3 Data Structure

In this section, we describe how to organize and implement the proposed data structure. We transform the suffixes into a certain form based on their corresponding binary search trees, after which we sort these encoded suffixes. After computing two arrays F and L of binary strings containing information required for the searching tasks, we build wavelet trees [9, 20] on them.

3.1 Suffix Representation

First, we define an encoding function $E(X)$ to transform a string into a certain form in which an order-isomorphic match can be easily determined. We use the fact that (i) two strings are order-isomorphic iff their corresponding binary search trees are value-identical, and (ii) two value-identical binary search trees are still value-identical after performing a root insertion with the same branching sequence.

We define $E(X)$ as a length-$|X|$ string over $2^{\mathbb{N}\cup\{\infty\}}$ where $\mathbb{N}$ is the set of natural numbers; i.e. each character of $E(X)$ is a set. If X is an empty string, $E(X)$ is also an empty string. If X is a non-empty string, then $E(X)$ can be defined recursively as follows: for $0 \le i \le |X|$,

$$E(x \circ X)[i] = \begin{cases} \{\infty\} & \text{if } i = 0, \\ E(X)[i-1] \cup \{i\} & \text{if } i - 1 \in \mathsf{turnpoint}(x, X) \\ E(X)[i-1] & \text{otherwise.} \end{cases} \quad (3)$$

We transform all the suffixes $T[i..n-1]$ of a given text string $T[0..n-1]$ into its encoded form $E(T[i..n-1])$ for $0 \leq i \leq n$; here, we define $T[n..n-1]$ to be an empty string. Before sorting them, we need to define the ordering on sets because characters of the encoded suffixes are sets. We compare two sets by treating each set as a string that consists of the elements in the ascending order: i.e., for two sets A and B, we define $A < B$ iff:

1. $\min A < \min B$, or
2. $\min A = \min B$ and $A - \{\min A\} < B - \{\min B\}$.

Now we can sort all the suffixes to determine the lexicographic ranks of (encoded) suffixes; note that each encoded suffix is a string of sets. Let $\mathsf{SA}[0..n]$ be an integer array such that $\mathsf{SA}[i] = j$ iff there are i encoded suffixes that are lexicographically smaller than $E(T[j..n-1])$: i.e. $\mathsf{SA}[i] = j \Leftrightarrow i = |\{0 \leq k \leq n \mid E(T[k..n-1]) < E(T[j..n-1])\}|$. This array is the so-called *suffix array*. Because $\mathsf{SA}[0..n]$ is a permutation of $0, \cdots, n$, we can also define its inverse $\mathsf{SA}^{-1}[0..n]$ such that $\mathsf{SA}^{-1}[\mathsf{SA}[i]] = i$ for $0 \leq i \leq n$.

3.2 Associating Adjacent Suffixes Using F and L Arrays

In this subsection, we consider the relation between two adjacent (encoded) suffixes $E(T[i..n-1])$ and $E(T[i+1..n-1])$. In the literature of compact indexing, it is common to use the so-called *LF-mapping*, which is defined as: for $0 \leq i \leq n$,

$$\mathsf{LF}(i) = \mathsf{SA}^{-1}[(\mathsf{SA}[i] + n) \mod (n+1)] \tag{4}$$

It maps the lexicographical rank of a suffix $E(T[\mathsf{SA}[i]..n-1])$ into the lexicographical rank of its positionally previous suffix $E(T[\mathsf{SA}[i]-1..n-1])$ for $0 < i \leq n$. For the rank of the suffix $E(T[0..n-1])$ starting at position 0, it maps to the rank of the empty suffix $E(T[n..n-1])$. Our goal is to implement $\mathsf{LF}(i)$ in a compact space, which will be done here by defining two length-$(n+1)$ arrays F and L of bitstrings.

To do this, we define an array $C[0..n]$ of bitstrings such that $C[n] = 1$, $C[n-1] = 001$, and, for $0 \leq i < n-1$,

$$C[i] = (00)^{l_0}01(00)^{l_1}01 \cdots (00)^{l_{k-1}}1 \tag{5}$$

where $\mathsf{branch}(T[i], T[i+1..n-1]) = \mathtt{R}^{l_0}\mathtt{L}^{1+l_1}\mathtt{R}^{1+l_2} \cdots d^{1+l_{k-1}}$; d is either $\mathtt{R}$ or $\mathtt{L}$, and $l_0, \cdots, l_{k-1} \geq 0$. In other words, $C[i]$ is a bitstring that indicates the branching direction when we insert a node with key $T[i]$ into the binary search tree $\mathsf{BST}(T[i+1..n-1])$ as the root node. It is obvious that the sum of the length of bitstrings $C[i]$ over $0 \leq i \leq n$ is bounded by $\mathcal{O}(R(T))$ where $R(T)$ is the total time taken by performing the root insertion of a node with key $T[n-1], \cdots, T[0]$ in order, starting with an empty binary search tree.

▶ **Lemma 2.** *For a string $T[0..n-1]$,*

$$\sum_{0 \leq i \leq n} |C[i]| = \mathcal{O}(R(T)) \tag{6}$$

Proof. Immediate from the fact that the length of $C[i]$ is proportional to the length of $\mathsf{branch}(\cdot, \cdot)$. ◀

The bits at even positions $(0, 2, 4, \cdots)$ indicate the length of the code; 1-bit at an even position means the code ends. The bits at odd positions represent the unary code of $l_0, l_1, \cdots, l_{k-1}$ (except that 1-bit does not follow 0-bits for the last run) which characterize the prepending character $T[i]$ for a suffix $T[i+1..n-1]$ in terms of its encoded form.

i	$\mathsf{SA}[i]$	$\mathsf{LF}[i]$	$F[i]$	$L[i]$	$E(T[\mathsf{SA}[i]..n-1])$
0	12	1	1	001	empty string
1	11	11	001	001	$\{\infty\}$
2	7	8	011	00011	$\{\infty\}$ $\{1,\infty\}$ $\{1,\infty\}$ $\{1,\infty\}$ $\{\infty\}$
3	8	2	011	011	$\{\infty\}$ $\{1,\infty\}$ $\{1,\infty\}$ $\{\infty\}$
4	1	12	0101001	0000011	$\{\infty\}$ $\{1,\infty\}$ $\{2,\infty\}$ $\{1,2,\infty\}$ $\{3,\infty\}$ $\{1,2,\infty\}\cdots$
5	9	3	01001	011	$\{\infty\}$ $\{1,\infty\}$ $\{\infty\}$
6	5	10	011	0001001	$\{\infty\}$ $\{1,\infty\}$ $\{\infty\}$ $\{1,2,\infty\}$ $\{1,\infty\}$ $\{1,\infty\}$ $\cdots$
7	3	9	011	0001011	$\{\infty\}$ $\{1,\infty\}$ $\{\infty\}$ $\{1,2,\infty\}$ $\{\infty\}$ $\{1,2,\infty\}\cdots$
8	6	6	00011	011	$\{\infty\}$ $\{\infty\}$ $\{1,2,\infty\}$ $\{1,\infty\}$ $\{1,\infty\}$ $\{\infty\}$
9	2	4	0001011	0101001	$\{\infty\}$ $\{\infty\}$ $\{1,2,\infty\}$ $\{3,\infty\}$ $\{1,2,\infty\}$ $\{\infty\}$ $\cdots$
10	4	7	0001001	011	$\{\infty\}$ $\{\infty\}$ $\{1,2,\infty\}$ $\{\infty\}$ $\{1,2,\infty\}$ $\{1,\infty\}$ $\cdots$
11	10	5	001	01001	$\{\infty\}$ $\{\infty\}$
12	0	0	0000011	1	$\{\infty\}$ $\{\infty\}$ $\{1,\infty\}$ $\{2,3,\infty\}$ $\{1,2,\infty\}$ $\{3,\infty\}$ $\cdots$

Figure 2 Example of sorted (encoded) suffixes and related information for T =5 3 4 1 6 2 8 7 9 10 12 11.

Now we define a length-$(n+1)$ array $F[0..n]$ as a permuted array of $C[0..n]$ via $\mathsf{SA}[0..n]$: for $0 \le i \le n$,

$$F[i] = C[\mathsf{SA}[i]] \tag{7}$$

We also define another length-$(n+1)$ array $L[0..n]$ such that: for $0 \le i \le n$,

$$L[i] = F[\mathsf{LF}(i)] \tag{8}$$

See Figure 2 for an example of how the (encoded) suffixes are sorted and how these arrays are computed.

3.3 Operations on F and L

Obviously, $\{C[i]\}$ is a prefix code; i.e. there exists no $0 \le i, j \le n$ such that $C[i]$ is a prefix of $C[j]$ unless $C[i] = C[j]$. Therefore, we can build a wavelet tree with a prefix-coding (as described in Section 3.2 in [20]) for F (and L, respectively).

For the completeness, we describe the detail. Let us consider the wavelet tree WT_L of $L[0..n]$. We define it recursively. For $0 \le i \le n$, we write $L[i][0]$ into a single bitmap B of length $n+1$. B is stored in the root node. We divide $I = \{0, \cdots, n\}$ into two disjoint subsets $I_0 = \{0 \le i \le n : L[i][0] = 0\}$, and $I_1 = \{0 \le i \le n : L[i][0] = 1\}$. Let L_0 be a sequence of bitmaps that can be obtained by writing for $L[i][1..]$ for $i \in I_0$ in order. Similarly, we define L_1 using L and I_1. Then we construct the wavelet tree for L_0 (and L_1) recursively, and make the tree as the left (and right, resp.) subtree of the current node. The recursion is repeated until all the bits are processed.

For a node w of the wavelet tree WT_L for L, a bit $b \in \{0,1\}$, and an integer $0 \le i, j \le n$, we define the following operation:

- $\mathsf{WT}_L.\mathsf{down}_w(b,i,j)$ returns a triplet (w', i', j') of a node w', integers i' and j' such that:
  - If $b = 0$, w' is the left child node of w, otherwise w' is the right child of w,
  - $i' = B_w.\mathsf{rank}(b, i-1)$, and
  - $j' = B_w.\mathsf{rank}(b, j) - 1$,

  where B_w is the bitmap of node w, and $B_w.\mathsf{rank}(b,i)$ is the number of b-bits in $B_w[0..i]$.

We construct the wavelet tree WT_F for F in the same way except, at this time, we build a select dictionary on the bitmap of each node. For a node w of the wavelet tree WT_F of F, and an integer $0 \leq i, j \leq n$, we define the following operation:

- $\mathsf{WT}_F.\mathsf{up}_w(i, j)$ returns a triplet (w', i', j') of a node w', integers i' and j' such that:
  - w' is the parent node of w,
  - $i' = B_{w'}.\mathsf{select}(b, i + 1)$, and
  - $j' = B_{w'}.\mathsf{select}(b, j + 1)$,

  where $B_{w'}$ is the bitmap of node w' where $b = 0$ if w is the left child of w', $b = 1$ otherwise; and $B_{w'}.\mathsf{select}(b, i)$ is the position of the i-th occurrence of b-bit in $B_{w'}$ (note: defined for $i \geq 1$, and $i = 1$ indicates the leftmost occurrence).

Note that L is a permuted array of F. Therefore, the topology of their wavelet trees are the same, and there is a one-to-one correspondence between the nodes of WT_L and the nodes of WT_F. For a node w in WT_L, we denote by $\mathsf{paired}(w)$ its associated node w' in WT_F.

Note also that the wavelet trees WT_L (and WT_F) can be emulated with a single bitmap equipped with the rank (and select) dictionary. This can be done by concatenating all the bitmaps in the level order. When we go down in WT_L, the bitmap boundary at the next level can be computed by counting the number of 0- or 1-bits within the boundary at the current level and node. As described later, going up in WT_F will be always followed by going down in WT_L, so we can reused the bitmap boundaries.

▶ **Lemma 3.** *The wavelet trees* WT_L *and* WT_F *require* $\mathcal{O}(R(T))$ *bits supporting the operations described above in* $\mathcal{O}(1)$ *time.*

Proof. It is well-known that rank and select operations on a length-n bitmap can be performed in $\mathcal{O}(1)$ time using $n + o(n)$ bits [4, 14]. The total number of bits stored in arrays F and L is $\mathcal{O}(R(T))$ according to Lemma 2. ◀

4 Searching Algorithm

In this section, we present the searching algorithm on the proposed data structure implemented in the previous section. As other searching methods based on suffix arrays, our searching algorithm represents the occurrences of a pattern string as an interval (p_s, p_e) on the suffix array. For a pattern string $P[0..m-1]$, we call a pair (p_s, p_e) of integers the *suffix range* for P if $p_s \leq i \leq p_e \Leftrightarrow P =_o T[\mathsf{SA}[i]..\mathsf{SA}[i] + m - 1]$. We give a simple algorithm to compute the suffix range for a pattern string. The number of occurrences can be immediately obtained by $p_e - p_s + 1$ once the suffix range (p_s, p_e) is computed. To report each occurrences in $\mathcal{O}(\lg n)$ time, we also present a new method to sample the suffix array.

4.1 Computing the Suffix Range

Given a length-m pattern, its suffix range is computed in the backward fashion. We start from the last character of the pattern. The encoded string of any length-1 string is always $\{\infty\}$, thus we have its suffix range as $(1, n)$ where n is the text length. We also have the binary search tree with a single node at this moment. For each character to be prepended, we perform the root insertion accordingly to obtain the bitstring representation of the branching directions in a similar way described in Equation 5. Then we walk down on WT_L according to the computed bitstring, followed by jumping to the corresponding node on WT_F, and tracing up to the root of the wavelet tree, which completes the updated suffix range. Figure 3 describes an example of an iteration that updates the suffix range by prepending a character at the currently searched pattern, Algorithm 1 describes the procedure to compute the suffix range of a given pattern.

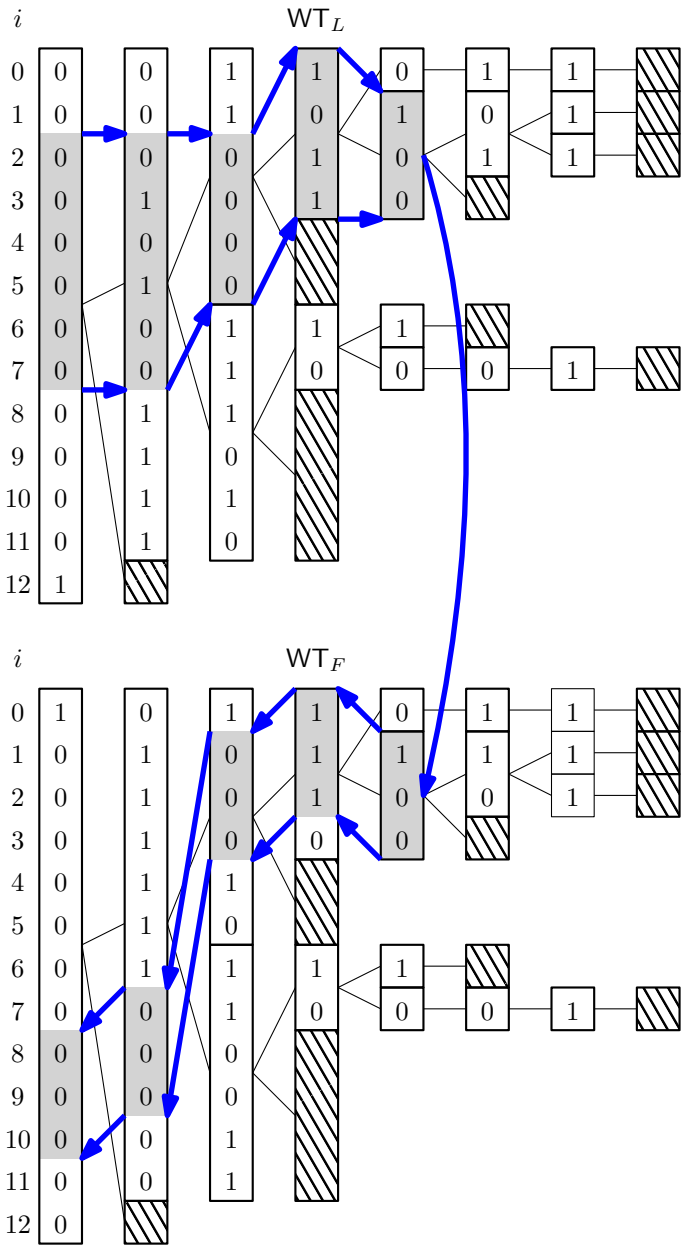


Figure 3 An iteration for searching for the pattern P =3 1 4 on the text string T described in Figure 2. For each wavelet tree, upward (downward) branches correspond to 0-bits (1-bits, resp.). Given a suffix range $(p_s, p_e) = (2, 7)$ for $P[1..2] = 14$, $P[0] = 3$ is to be prepended. Since $\mathsf{branch}(P[0], P[1..2]) = \texttt{RL}$, $c = \texttt{00011}$. We walk down on WT_L according to $c[0..|c| - 2]$, then jump to the corresponding locus on WT_F, then trace up to the root. After the iteration the suffix range is updated to $(8, 10)$. Note that $\mathsf{SA}[8] = 6$, $\mathsf{SA}[9] = 2$, $\mathsf{SA}[10] = 4$, and $T[6..8]$ =8 7 9, $T[2..4]$ =4 1 6, and $T[4..6]$ =6 2 8, which are all order-isomorphic to $P[0..2]$=3 1 4.

Algorithm 1 Computing the suffix range (p_s, p_e) of $P[0..m-1]$.

```
1: procedure computeSuffixRange(P[0..m − 1]: a non-empty string)
2:     (p_s, p_e) ← (1, n)    ▷ The suffix range for a length-1 string P[m − 1] is always (1, n).
3:     for i = m − 2, ··· , 0 do
4:         c ← bitstring as defined in Eq. (5) w.r.t. P[i] and P[i + 1..m − 1].
5:         w ← the root of WT_L
6:         for j = 0, ··· , |c| − 2 do        ▷ Process the code except the 1-bit at the end
7:             (w, p_s, p_e) ← WT_L.down_w(c[j], p_s, p_e)
8:         end for
9:         w′ ← paired(w)                                          ▷ Jump to WT_F
10:        for j = |c| − 2, ··· , 0 do        ▷ Trace up until reaching the root node
11:            (w′, p_s, p_e) ← WT_F.up_w′(p_s, p_e)
12:        end for
13:    end for
14:    return (p_s, p_e)
15: end procedure
```

It is easy to show its running time. It is proportional to the sum of the code length over all iterations, thereby proportional to the time required for performing the root insertion of nodes keyed by $P[m-1], \cdots, P[0]$ to the empty binary search tree.

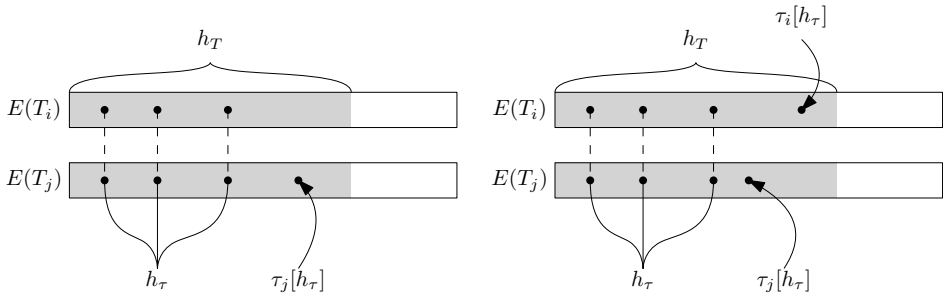
▶ **Lemma 4.** *Algorithm 1 runs in $\mathcal{O}(R(P))$ time.*

Proof. The time taken for a single iteration for i is proportional to $|c|$ because other basic operations take $\mathcal{O}(1)$ time. $|c|$ is proportional to the length of the path retrieved in the binary search tree. Therefore, the total time is $\mathcal{O}(R(P))$ by its definition. ◀

Now we show the correctness of this algorithm. Suppose we have a suffix range (p_s, p_e) and a bitstring c that characterizes the next character $P[i]$ to be prepended to the currently searched pattern $P[i+1..m-1]$. Among the suffixes within the current suffix range (for $P[i+1..m-1]$), we need to determine which suffix should be included in the updated suffix range (for $P[i..m-1]$). Let us define $\mathbf{b}_j = \mathsf{branch}(T[\mathsf{SA}[\mathsf{LF}(j)]], T[\mathsf{SA}[j]..n-1])$ for $p_s \le j \le p_e$, and let $\mathbf{b}_P = \mathsf{branch}(P[i], P[i+1..m-1])$. If $\mathsf{LF}(j)$ is included in the update suffix range, then $\mathbf{b}_j$ must have a prefix $\mathbf{b}_P$, which is equivalent to that $L[j]$ has a prefix $c[0..|c|-2]$. Otherwise, the root insertion for the suffix would take a different branching direction within the common prefix region, which result in being excluded from the updated suffix range. As a result, when we walk down in WT_L according to $c[0..|c|-2]$, only the suffixes that are to be included in the updated suffix range remain.

The remaining task is to show that the suffixes within the interval (p_s, p_e) at w' after executing Line 9 correspond to the correct target suffixes. In order to show this, we establish the following lemma, which claims that the ordering of two (encoded) suffixes inverts via LF-mapping iff the larger suffix has at least one turn points (1) beyond the common turn points, (2) within the common prefix, (3) before any non-common turn point for the smaller suffix within the common prefix.

▶ **Lemma 5.** *Let i and j be integers such that $0 \le i < j \le n$. Let $T_i = T[\mathsf{SA}[i]..n-1]$ and $t_i = T[\mathsf{SA}[\mathsf{LF}(i)]]$ be the suffix whose rank is i and its previous character on T. Similarly, we define $T_j = T[\mathsf{SA}[j]..n-1]$ and $t_j = T[\mathsf{SA}[\mathsf{LF}(j)]]$. Consider $\tau_i = \mathsf{turnpoint}(t_i, T_i)$ and $\tau_j = \mathsf{turnpoint}(t_j, T_j)$. Let h_T be the length of the longest common prefix of $E(T_i)$ and $E(T_j)$, and let h_τ be the length of the longest prefix of τ_i and τ_j. Then $\mathsf{LF}(i) > \mathsf{LF}(j)$ iff:*



(a) $|\tau_i| = h_\tau$ (b) $|\tau_i| > h_\tau$ and $\tau_j[h_\tau] < \tau_i[h_\tau]$

Figure 4 Example of order-inverted cases described in Lemma 5. Suffixes (T_i and T_j), the length of the common prefix of the encoded suffixes (h_T), turn points (τ_i and τ_j), and the length of the common prefix of the turn point sequences (h_τ) are defined as in the lemma. Shaded area indicates the common prefix of the two encoded suffixes. Dots indicate turn points where the changes are made when the corresponding characters are prepended. In order to invert the ordering of the two suffixes via LF-mapping, there must be a turn point for T_j (indicated by $\tau_j[h_\tau]$) between the position of the last common turn points and the earliest position of the end of the longest common prefix and the next non-common turn point for suffix T_i.

1. $|\tau_j| > h_\tau$,
2. $\tau_j[h_\tau] < h_T$, *and*
3. $|\tau_i| = h_\tau$ *or* ($|\tau_i| > h_\tau$ *and* $\tau_j[h_\tau] < \tau_i[h_\tau]$).

Proof. ($\Rightarrow$) We prove by contrapositive. Note that the changes are made only at turn points. For each case, when we assume it is false, then the changes made in $E(T_j)$ are either (i) also made in $E(T_i)$ at the same position or (ii) out of the common prefix, so the ordering is not affected.

1. Let us assume that $|\tau_j| \leq h_\tau$. Then, for all positions $q \in \tau_j$ in which $E(T_j)[q]$ changes, $E(T_i)[q]$ also changes. Therefore $E(t_j \circ T_j)[q+1] = E(t_i \circ T_i)[q+1]$ for such q, which means the ordering is not inverted.
2. Let us assume that $\tau_j[h_\tau] \geq h_T$. Then the first position only $E(T_j)$ changes is out of the common prefix. If $\tau_j[h_\tau] > h_T$, the ordering remains the same because it is determined at position h_T. Let us assume $\tau_j[h_\tau] = h_T$. We have $E(t_i \circ T_i)[h_T + 1] = E(T_i)[h_T] < E(T_j)[h_T]$. Because $h_T + 1 \notin E(T_i)[h_T]$, we still have $E(T_i)[h_T] < E(T_j)[h_T] \cup \{h_T + 1\} = E(t_j \circ T_j)[h_T + 1]$.
3. Let us assume that $|\tau_i| \neq h_\tau$ and ($|\tau_i| \leq h_\tau$ or $\tau_j[h_\tau] \geq \tau_i[h_\tau]$). Since h_τ cannot be greater than $|\tau_i|$, we can rewrite it as: $|\tau_i| > h_\tau$ and ($|\tau_i| = h_\tau$ or $\tau_j[h_\tau] \geq \tau_i[h_\tau]$)$\Leftrightarrow$$|\tau_i| > h_\tau$ and $\tau_j[h_\tau] \geq \tau_i[h_\tau]$. However if $\tau_j[h_\tau] \geq \tau_i[h_\tau]$, the position on $E(T_i)$ where a change is made is earlier than that on $E(T_j)$, therefore $E(T_j)$ cannot become smaller after prepending the corresponding character.

($\Leftarrow$) We can easily see that, for $0 \leq q \leq \tau_j[h_\tau]$, $E(t_i \circ T_i)[q] = E(t_i \circ T_i)[q]$. Note that, for any q, $E(T_j)[q] > E(T_j)[q] \cup \{q+1\}$ because each encoded character, which is a set, contains ∞. And we have $E(t_i \circ T_i)[\tau_j[h_\tau]+1] = E(T_i)[\tau_j[h_\tau]] = E(T_j)[\tau_j[h_\tau]] > E(T_j)[\tau_j[h_\tau]] \cup \{\tau_j[h_\tau]+1\} = E(t_j \circ T_j)[\tau_j[h_\tau] + 1]$. See Figure 4. Therefore $E(t_i \circ T_i) > E(t_j \circ T_j)$, which means $\mathsf{LF}(i) > \mathsf{LF}(j)$. ◀

Let us look at the interval (p_s, p_e) at node w of WT_L. The suffixes corresponding to indices $j < p_s$ or $p_e < j$ do not have a common prefix long enough to invert the lexicographical ordering. Therefore, by Lemma 5, the ordering does not change. Each suffix on the left (and

right, resp.) side of the interval (p_s, p_e) at node w of WT_L corresponds to a suffix on the left (and right, resp.) side of the interval (p_s, p_e) at node $w' = \mathsf{paired}(w)$ of WT_F. Therefore, the interval at node w' of WT_F that correspond to the interval (p_s, p_e) at node w is still (p_s, p_e). Although the lexicographical ranks of the suffixes within the interval are possibly jumbled, all of them are the suffixes that need to be included in the updated suffix range.

Finally, by the similar reason, during tracing up to the root node of WT_F the interval length does not change. Because, otherwise it means the lexicographical rank is inverted by the turn point out of the common prefix.

4.2 Reporting the Occurrences: Suffix Sampling along Codes

In order to reduce the space requirement to store the suffix array, the existing succinct and compact indexes usually use the sampling method. The standard sampling method is to sample the entries whose value is a multiple of δ. Then we can retrieve the suffix array value by calling $\mathsf{LF}(i)$ at most δ times until reaching any sampled entry. However, this method is no longer efficient for our method. Note that we can compute $\mathsf{LF}(i)$ by walking down from position i in the root of WT_L until we reach the leaf, then tracing up to the root of WT_F; we can easily see its correctness by showing, for any $0 \leq i < j \leq n$ such that $L[i] = L[j]$, $\mathsf{LF}(i) < \mathsf{LF}(j)$ as a corollary of Lemma 5 because these two suffixes share all the turn points within the common prefix of their encoded strings. This operation takes $\Theta(|L[i]|)$ time, and the length $|L[i]|$ of the bitstring $L[i]$ can be $\Theta(n)$ in the worst case.

Rather, we propose to sample the suffix array based on the actual time taken during the computation of $\mathsf{LF}(i)$. Consider the computation of $\mathsf{LF}(\cdots(\mathsf{LF}(0))) = \mathsf{LF}^n(0)$. Starting at the position 0 at the root node of WT_L, and calling $\mathsf{LF}(\cdot)$ is equivalent to walking down to a leaf node, jumping to WT_F, followed by tracing up in WT_F. We repeat it until we arrive back the starting position, which eventually forms a cycle along the bits of the wavelet tree. We sample the suffix array entries for every δ-th bit along this cycle (see Figure 5). Note that the same suffix array value can be sampled multiple times when the length of its corresponding L-value (bitstring) is longer than δ. Note also that each wavelet tree can be linearized as a single bitvector, so we can store the sampled values from each wavelet tree using an array with a bitvector indicating marked locus. When we retrieve a suffix array value, we repeatedly call $\mathsf{LF}(\cdot)$ until we arrive any locus at which the suffix array is sampled. Then we can get the desired value by adding the retrieved sampled value by the number of jumps from WT_L to WT_F performed during the $\mathsf{LF}(\cdot)$ calls.

It is easy to see that $\mathcal{O}(\delta)$ navigating operations on the wavelet trees are needed until any of marked locus is met. By setting $\delta = \Theta(\lg n)$, we can have the following result.

▶ **Lemma 6.** *For $0 \leq i \leq n$, $\mathsf{SA}[i]$ can be computed in $\mathcal{O}(\lg n)$ time, and the required space is $\mathcal{O}(R(T))$ bits.*

Proof. The total number of bits to mark the sampled positions is $\mathcal{O}(R(T))$. By setting $\delta = \Theta(\lg n)$, the number of sampled entries of the suffix tree is $\mathcal{O}(R(T)/\lg n)$ and each entry requires $\lceil \lg n \rceil$ bits to be stored. The space required to store the sampled entries of the suffix array is $\mathcal{O}(R(T)/\lg n) \cdot \lceil \lg n \rceil = \mathcal{O}(R(T))$ bits. ◀

5 Non-distinct Case

In this section, we consider the case in which the characters of strings are drawn from an alphabet of size $\sigma (< n)$ so that the same character can appear more than once in a string. We can deal with non-distinct characters by making a slight modifications in order to consider

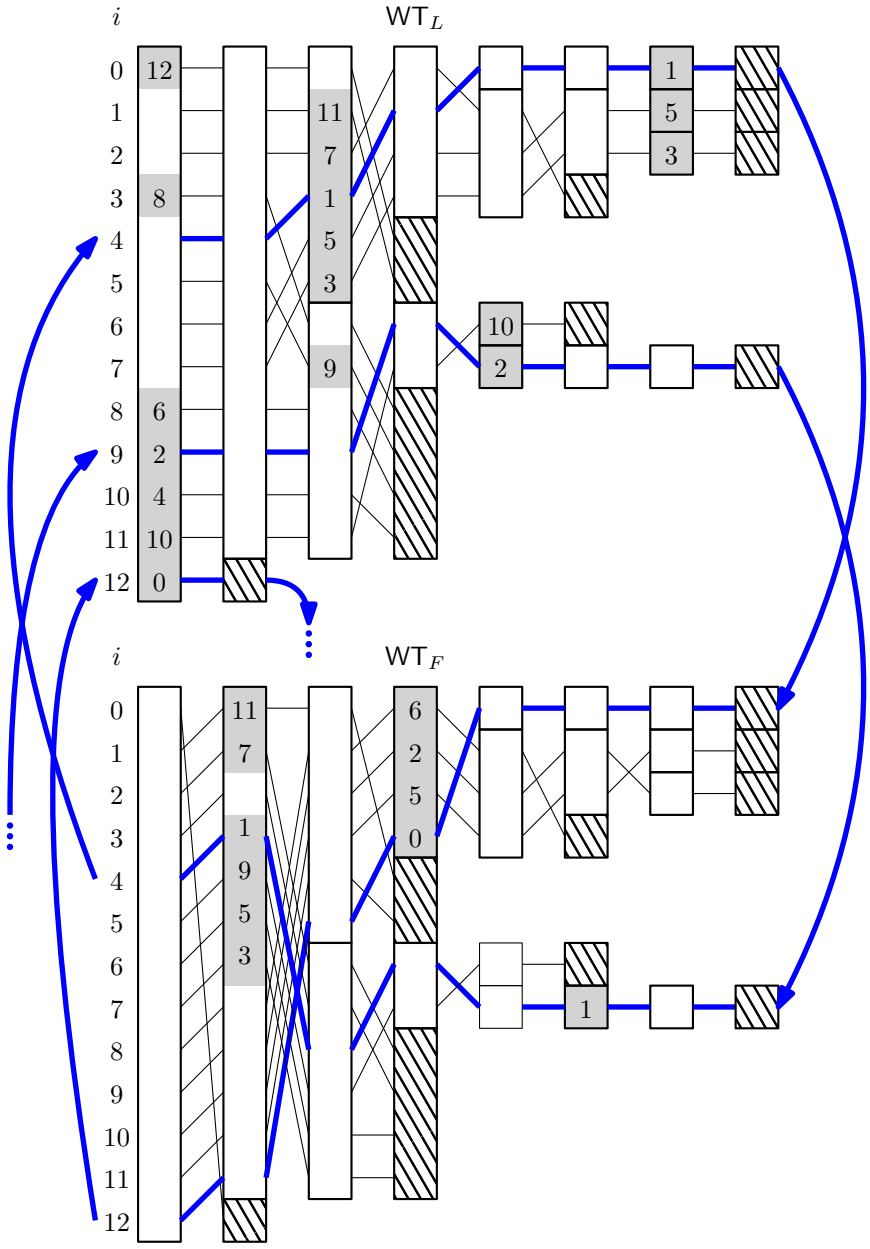


Figure 5 Suffix array sampling along the cycle formed by the visited order of bits by successive calls of LF-mapping ($\delta = 4$). Blue thick lines indicate a part of this cycle. Shaded cells are the marked locus where the sampled suffix array values are stored.

the equality case. Each node has another pointer, which we call a *middle pointer*, in addition to pointers to its left and right child. This middle pointer forms a linked list connecting the nodes with the same key. Then the following modifications are applied:

1. Root insertion: if we find a node with the same key x, we cut the links to its left and right child, if any. Then we remove the existing node, after which the new node is inserted as the root node and links are reestablished properly. Then the removed node is connected to the new node via the middle pointer.
2. Checking $\mathsf{BST}(X) =_v \mathsf{BST}(Y)$: for checking value-identical BSTs, we also consider the node connected via middle pointers.
3. $\mathsf{branch}(x, X)$ in Eq. (1): we add the case of $x = X[\mathsf{vpath}(x, X)[i]]$, in which the corresponding value is `E`.
4. $C[i]$ in Eq. (5): we replace it with the following to deal with the case of `E`. More specifically, for every three bits, the code ends if the first bit is 1, otherwise the following two bits represent the content of $\mathsf{branch}(x, X)$; 00: same direction, 01: direction switched, 11: equality.

$$C[i] = \begin{cases} (000)^{l_0}001(000)^{l_1}001\cdots(000)^{l_1}001\cdots(000)^{l_{k-1}}1 & \mathsf{branch}(\cdot) \text{ does not end with } \texttt{E}. \\ (000)^{l_0}001(000)^{l_1}001\cdots(000)^{l_1}001\cdots(000)^{l_{k-2}}011\ 1 & \mathsf{branch}(\cdot) \text{ ends with } \texttt{E}. \end{cases} \tag{9}$$

5. $E(x \circ X)$ in Eq. (3): we define it as a string of multisets, and we add the equality case in which we insert the corresponding index twice as follows.

$$E(x \circ X)[i] = \begin{cases} \cdots \\ E(X)[i-1] \cup \{i, i\} & \text{if } x = X[i-1] \\ \cdots \end{cases} \tag{10}$$

After applying the above modifications, we can reuse the searching algorithm. Note that `E` can appear only at the end of $\mathsf{branch}(\cdot, \cdot)$ so it does not involve many complications and the useful properties still hold.

For the time and space complexity analysis, we need to know the expected value of $R(X)$, the time taken by performing root insertion with keys $X[|X|-1], \cdots, X[0]$ in order. Note that we do not follow the middle pointer in this insertion process; middle pointers are (conceptually) used only when comparing trees. Hence, we can consider only the nodes connected via pointers to left and right subtrees, so the number of nodes of the binary search tree that are of concern in this insertion process is bounded by σ. Therefore, averaging over all possible binary search trees with σ nodes, we can see that the expected length of the path from the root to a node is $\mathcal{O}(\lg \sigma)$ as in [24, 22]. Therefore, the expected value of $R(X)$ is $\mathcal{O}(|X| \lg \sigma)$. Combining it with the previous results, we can have the following theorem.

▶ **Theorem 7.** *There exists a data structure that uses $\mathcal{O}(n \lg \sigma)$ bits on average, and is capable of counting the number of occurrences in $\mathcal{O}(m \lg \sigma)$ time on average, and reporting each occurrence in $\mathcal{O}(\lg n)$ time.*

Proof. Immediate from Lemmas 3, 4 and 6, and the fact that, for a string X randomly drawn over an alphabet of size σ, the expected value of $R(X)$ is $\mathcal{O}(|X| \lg \sigma)$. ◀

6 Connections to Other Methods

The encoded string is represented based on the retrieved path through the corresponding binary search tree. Deriving a variant by adding some restrictions to the binary search tree can naturally come up. For example, we can restrict the number of nodes by keeping the binary search tree have nodes having a value at most τ where τ is a parameter. In this case, we remove the (leaf) node with a value greater than τ, if any, after inserting a new node. This method is equivalent to checking the order-isomorphism with a size-τ window, as similar to [8]. If $\tau = 1$, it is identical to a heuristic filtering method that uses the up-down signature used in [3]. If we set $\tau = \mathcal{O}(\lg^c n)$ for some constant c, then we can have the required space to be $\mathcal{O}(n \lg \lg n)$ bits on average, which is related to the space requirement of the data structure for order-isomorphic matching with length-restricted patterns described in [10], which $\mathcal{O}(n \lg \lg n)$ in bits (in the worst case). Similarly, we can restrict other measures; e.g. the number of nodes and the maximum depth of the tree. We can also restrict the number of turn points used to encode the suffixes. If we restrict the number of turn points to be 1, the matching problem becomes analogous to Cartesian tree matching [21] because the first turn point indicates the leftmost element that is greater than the prepending character. The proposed method can also be viewed as an extension of pointer sequence matching [16]. We can view each node has pointers to its corresponding turn points. From this perspective, restricting the number of nodes of the binary search tree can be viewed as restricting the length of the pointers; and the restricting the number of turn points is equivalent to restricting the out-degree of a node in the pointer sequence viewpoint.

7 Conclusion

Developing a space-efficient index for order-isomorphic matching has been considered challenging. In this paper, we have presented a new method to index a string regarding this problem. The new suffix representation has been introduced, based on which two arrays L and F are computed using the sorted suffixes and the relation among them. The proposed searching algorithm is quite simple as it just traverses down through the wavelet tree for L and then traces up along the wavelet tree for F. We also presented a new suffix sampling method based on the time taken during the computation of $\mathsf{LF}(\cdot)$, which also resolves an open problem in [17], which asked if reporting an occurrence can be improved when L-values are of variable lengths, possibly as long as $\Theta(n)$. We leave the following open problems:

1. **Suffix array construction:** it has not been shown whether we can efficiently sort the suffixes that are encoded as described in this paper. Even if we use a generalized fast indexing method such as [5, 1], it is not trivial because a single character of an encoded suffix can be a set of $\Theta(n)$ integers in the worst case, so comparing characters may take too much time.
2. **Reducing the space requirement further:** although we achieved the compactness in the average case analysis, the space requirement can be $\mathcal{O}(n\sigma)$ in the worst case. Using a different representation for $C[i]$'s may allow to reduce the worst case space complexity; however, further investigation and analysis are needed. Ultimately, it is interesting to know whether there exists any $n \lg \sigma + o(n \lg \sigma)$-bit index.
3. **Discovering new string matching problems:** in this paper, we define the matching using the binary search tree. In a similar fashion, we can discover new string matching problems using appropriate discrete structures. For example, we can think of a trajectory matching problem, in which two trajectories are defined to be a match if their corresponding triangulations constructed by inserting the points in order are equivalent.

References

1. Amihood Amir and Eitan Kondratovsky. Sufficient Conditions for Efficient Indexing Under Different Matchings. In *Proceedings of the 30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 6:1–6:12, 2019. doi:10.4230/LIPIcs.CPM.2019.6.
2. Brenda S. Baker. Parameterized Pattern Matching: Algorithm and Applications. *Journal of Computer and System Sciences*, 52:28–42, 1996. doi:10.1006/jcss.1996.0003.
3. Tamanna Chhabra, M. Oğuzhan Külekci, and Jorma Tarhio. Alternative Algorithms for Order-Preserving Matching. In *Proceedings of the Prague Stringology Conference*, pages 36–46, 2015.
4. David Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1996.
5. Richard Cole and Ramesh Hariharan. Faster Suffix Tree Construction with Missing Suffix Links. In *Proceedings of the 32nd Annual Symposium on Theory of Computing (STOC)*, pages 407–415, 2000. doi:10.1145/335305.335352.
6. Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, and Solon P. Pissis. Order-Preserving Incomplete Suffix Trees and Order-Preserving Indexes. In *Proceedings of the 20th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 84–95, 2013. doi:10.1007/978-3-319-02432-5_13.
7. Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Order-preserving Indexing. *Theoretical Computer Science*, 628:122–135, 2016. doi:10.1016/j.tcs.2015.06.050.
8. Gianni Decaroli, Travis Gagie, and Giovanni Manzini. A Compact Index for Order-preserving Pattern Matching. *Software: Practice and Experience*, 46(6):1041–1051, 2019.
9. Paolo Ferragina and Giovanni Manzini. Opportunistic Data Structures with Applications. In *Proceedings of the 41st Annual Symposium on Foundation of Computer Science (FOCS)*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.
10. Travis Gagie, Giovanni Manzini, and Rossano Venturini. An Encoding for Order-Preserving Matching. In *Proceedings of the 25th European Symposium on Algorithms (ESA)*, pages 38:1–38:15, 2017. doi:10.4230/LIPIcs.ESA.2017.38.
11. Arnab Ganguly, Dhrumil Patel, Rahul Shah, and Sharma V. Thankachan. LF Successor: Compact Space Indexing for Order-Isomorphic Pattern Matching. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 71:1–71:19, 2021. doi:10.4230/LIPIcs.ICALP.2021.71.
12. Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Structural Pattern Matching - Succinctly. In *Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC)*, pages 35:1–35:13, 2017. doi:10.4230/LIPIcs.ISAAC.2017.35.
13. Arnab Ganguly, Rahul Shah, and SharmaV. Thankachan. pBWT: Achieving Succinct Data Structures for Parameterized Pattern Matching and Related Problems. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 397–407, 2017. doi:10.1137/1.9781611974782.25.
14. G. Jacobson. Space-efficient Static Trees and Graphs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989. doi:10.1109/SFCS.1989.63533.
15. Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving Matching. *Theoretical Computer Science*, 525:68–79, 2014. doi:10.1016/j.tcs.2013.10.006.
16. Sung-Hwan Kim and Hwan-Gue Cho. Indexing Isodirectional Pointer Sequences. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC)*, pages 35:1–35:15, 2020. doi:10.4230/LIPIcs.ISAAC.2020.35.
17. Sung-Hwan Kim and Hwan-Gue Cho. A Compact Index for Cartesian Tree Matching. In *Proceedings of the 32nd Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 18:1–18:19, 2021. doi:10.4230/LIPIcs.CPM.2021.18.

18 Sung-Hwan Kim and Hwan-Gue Cho. Simpler FM-index for Parameterized String Matching. *Information Processing Letters*, page 106026, 2021. doi:10.1016/j.ipl.2020.106026.

19 M. Kubica, T. Kulczyński, J. Radoszewski, W. Rytter, and T. Waleń. A Linear Time Algorithm for Consecutive Permutation Pattern Matching. *Information Processing Letters*, 113(12):430–433, 2013. doi:10.1016/j.ipl.2013.03.015.

20 Gonzalo Navarro. Wavelet Trees for All. *Journal of Discrete Algorithms*, 25:2–20, 2014. doi:10.1016/j.jda.2013.07.004.

21 Sung Gwan Park, Amihood Amir, Gad M. Landau, and Kunsoo Park. Cartesian Tree Matching and Indexing. In *Proceedings of the 30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 16:1–14, 2019. doi:10.4230/LIPIcs.CPM.2019.16.

22 Bruce Reed. The Height of a Random Binary Search Tree. *Journal of the ACM*, 50(3):306–332, 2003. doi:10.1145/765568.765571.

23 Tetsuo Shibuya. Generalization of a Suffix Tree for RNA Structural Pattern Matching. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 393–406, 2000. doi:10.5555/645900.672451.

24 C. J. Stephenson. A Method for Constructing Binary Search Trees by Making Insertions at the Root. *International Journal of Computer & Information Science*, 9:15–29, 1980. doi:10.1007/BF00995807.

Space-Efficient Graph Coarsening with Applications to Succinct Planar Encodings

Frank Kammer
THM, Technische Hochschule Mittelhessen, Giessen, Germany

Johannes Meintrup
THM, Technische Hochschule Mittelhessen, Giessen, Germany

Abstract

We present a novel space-efficient graph coarsening technique for n-vertex planar graphs G, called *cloud partition*, which partitions the vertices $V(G)$ into disjoint sets C of size $O(\log n)$ such that each C induces a connected subgraph of G. Using this partition $\mathcal{P}$ we construct a so-called *structure-maintaining minor* F of G via specific contractions within the disjoint sets such that F has $O(n/\log n)$ vertices. The combination of $(F, \mathcal{P})$ is referred to as a *cloud decomposition*.

For planar graphs we show that a cloud decomposition can be constructed in $O(n)$ time and using $O(n)$ bits. Given a cloud decomposition $(F, \mathcal{P})$ constructed for a planar graph G we are able to find a balanced separator of G in $O(n/\log n)$ time. Contrary to related publications, we do not make use of an embedding of the planar input graph. We generalize our cloud decomposition from planar graphs to H-minor-free graphs for any fixed graph H. This allows us to construct the succinct encoding scheme for H-minor-free graphs due to Blelloch and Farzan (CPM 2010) in $O(n)$ time and $O(n)$ bits improving both runtime and space by a factor of $\Theta(\log n)$.

As an additional application of our cloud decomposition we show that, for H-minor-free graphs, a tree decomposition of width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ can be constructed in $O(n)$ bits and a time linear in the size of the tree decomposition. A similar result by Izumi and Otachi (ICALP 2020) constructs a tree decomposition of width $O(k\sqrt{n}\log n)$ for graphs of treewidth $k \leq \sqrt{n}$ in sublinear space and polynomial time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases planar graph, H-minor-free, space-efficient, separator, tree decomposition

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.62

Related Version *Full Version*: `https://arxiv.org/abs/2205.06128`

Funding *Johannes Meintrup*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 379157101.

1 Introduction

Graphs are used to model a multitude of systems that can be expressed via entities and relationships between these entities. Many real-world problems operate on very large graphs for which standard algorithms and data structures use too much space. This has spawned an area of research with the aim of reducing the required space. Examples include large road-networks [50] or social graphs arising from interactions between users of large internet communities [19]. Therefore, it is of high interest to find compact representations of such graphs, space-efficient algorithms and other space-efficient or succinct data structures. In the following we denote by n the number of vertices of a graph under consideration and m the number of edges. An algorithm is called space-efficient if it has (almost) the same asymptotic runtime as a standard algorithm for the same problem, but uses asymptotically fewer bits. Examples include space-efficient graph searching algorithms such as depth-first search and breadth-first search, which run in linear time, but use $O(n \log n)$ bits with standard methods. Space-efficient solutions lower the space requirement to $O(n)$ bits and

keep the runtime asymptotically (almost) the same. For a data structure or algorithm to be called *succinct* the space used must be $Z + o(Z)$ bits with Z being the information theoretic minimum to store the data. One of the most researched topics regarding space-efficiency are graph-traversal algorithms such as depth-first search (DFS). It is still an open research question if a linear-time DFS exists that uses $O(n)$ bits, with the current best bound of $O(n + m + \min\{n, m\} \log^* n)$ due to Hagerup [24], with $\log^* n$ being the iterated logarithm. This result is the work of gradual improvements over the span of seven years spanning multiple publications by multiple research groups, with some of the first results providing a runtime $O(m \log n)$ time due to Asano et al. [6] and a runtime of $O((n + m) \log \log n)$ due to Elmasry et al. [17]. Other typical problems that can be easily solved in settings where space is of no concern, such as storing spanning trees or simple mappings between vertices, require new problem-specific strategies when considered in the space-efficient setting.

Another interesting setting is aiming for arbitrary polynomial time runtime, but using only $o(n)$, or even $O(\log n)$ bits. In such settings information can be recomputed in polynomial time, and for problems such as mappings this is often trivial, while other problems such as deciding if two vertices are connected by a path in a graph requires the very involved algorithm due to Reingold [48], but is quite easy in a space-efficient setting. Thus, our space-efficient setting differentiates itself quite strongly from both, the well-researched sublinear-space settings and the common settings that do not regard space as limited.

Many large graphs that arise in practice have some known structural property that allows the design of specialized techniques that make use of these properties. A common such structural property is the existence of small separators. Such a *separator* is a small subset of vertices whose removal disconnects the graph if it was connected previously, or increases the number of connected components if it was not connected. For the graphs of interest to this work the separators are so-called balanced separators. For now the intuition suffices that such balanced separators split the graph in somewhat equally parts. Section 2 contains precise definitions of these terms and also defines what we consider small in regard to separators. Arguably the most well-known graphs that contain such small balanced separators are planar graphs, which are graphs that can be drawn in the plane without overlapping edges. All planar graphs contain a balanced separator of size $O(\sqrt{n})$ [42]. Similar separator theorems exist for almost-planar graphs [20] (which includes road networks) and well-formed meshes such as nearest neighbor graphs [44] and H-minor-free graphs for some fixed graph H [36]. For arbitrary separable graphs there exists a polylogarithmic approximation algorithm for finding balanced separators due to Leighton and Rao [40].

We present a novel partitioning scheme called *cloud partition* for planar graphs that partitions the vertices of a connected input graph G into connected subsets called *clouds* that induce a connected subgraph and are of size $O(\log n)$. For a cloud partition $\mathcal{P}$ constructed for G we construct a so-called *structure-maintaining minor* F of G induced by $\mathcal{P}$. For easier reading comprehension we call vertices of such a minor *nodes*. Intuitively, a node v of a structure-maintaining minor F is mapped to one or more clouds $C \in \mathcal{P}$ such that $|V(F)| = O(n/\log n)$ – the exact mapping is outlined in Section 3. A solution for some problems such as finding small separators can be found in F and then translated to an approximate solution for G. As F contains $O(n/\log n)$ nodes, the time and space bounds of linear or superlinear algorithms can be decreased by a factor of $\Omega(\log n)$ when being executed on F instead of G. We show how this speedup is especially helpful for recursive algorithms such as the recursive separator search used during the computation of the succinct representation of separable graphs due to Blelloch and Farzan [11]. Additionally, we show that small modifications of this recursive separator search can be used to find a tree decomposition

of width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ for planar graphs in $O(n)$ bits and a time linear in the size of the tree decomposition. Finally, we generalize our partitioning scheme from planar graphs to H-minor-free graphs for any fixed graph H.

One of the key points of our novel scheme is that we do not make use of a (planar) embedding of the input graph, as there is no known way to construct such an embedding with $O(n)$ bits and in linear time. It is often implied that an embedding is given alongside a planar graph in many of the publications regarding planar graphs, as it is computable in linear time [28] when space is of no concern. Additionally, it is often required that the planar graph is maximal. Both of these properties are, for example, required for the major result of the $O(\sqrt{n})$-separator theorem [42] or the so-called r-partition of Klein et al. [39], which similarly to our result partitions the input graph into regions of size $O(n/r)$. Again, if the graph is not maximal, it can be easily made maximal in linear time with the help of an embedding. In a space-efficient context we can not make use of either of these properties and are thus quite limited. For sublinear space settings there exist algorithms that produce an embedding in polynomial time with a rather large polynomial degree [5, 16]. Thus, the $O(n)$-bit setting provides a unique challenge due to the additional goal of matching the runtime of non space-efficient algorithms.

Succinct and space-efficient representations of planar graphs is a highly researched topic partially due to the practical applications. For compressing planar graphs without regards to providing fast access operations refer to the work of Keeler et al. [37] for an $O(n)$ bits representation and for a compression within the information theoretic lower bound refer to He et al. [26]. Due to Munro and Raman [45] there exists an encoding using $O(n)$ bits that allows constant-time queries which has subsequently been improved by Chiang et al. [14] to use a constant factor less space. We use the succinct representation due to Blelloch and Farzan, which allows encoding arbitrary separable graphs and subsequently allows constant-time adjacency-, neighborhood- and degree-queries [11], which builds on the work of Blanford et al. [10]. For H-minor-free graphs for a fixed graph H, their encoding takes $\Theta(n \log n)$ time and $\Theta(n \log n)$ bits using their described technique of recursive separator searches. We are able to improve it to $O(n)$ time and $O(n)$ bits. For planar graphs we only improve the space-requirement from $\Theta(n \log n)$ to $O(n)$ due to the algorithm of Goodrich [22]. Note that all mentioned publications above regarding succinct representations of planar graphs use $\Theta(n \log n)$ bits during the construction. Additionally, they assume a planar embedding is given explicitly or implicitly by making direct use of it, or referring to other publications as sub-routines that require them. For maximal planar graphs there exist special encodings. Some major research in this field is due to Aleardi et al. [1, 2, 3, 4].

The general technique of graph coarsening has been heavily used in the past for practical and theoretical algorithms [13, 18, 27, 34, 49]. A typical approach is to find a matching $E' \subset E(G)$ for a graph G and contract the edges in E' to find a minor F of G. The problem at hand is then solved on F and the solution is translated to an approximate solution for G [34]. Highly specialized practical algorithms using sophisticated data structures are implemented in the METIS [35] and SCOTCH [46] library. Other approaches such as graph coarsening on the GPU [7, 8] or graph coarsening via neural networks [12] have been developed.

For finding tree decompositions in a space-efficient manner refer to the work of Kammer et al. [32] and the work of Izumi and Otachi [30]. Izumi and Otachi showed that for a given graph G with treewidth $k \leq \sqrt{n}$ there exists an algorithm that obtains a tree decomposition of width $O(k\sqrt{n} \log n)$ in polynomial time and $O(k\sqrt{n} \log^2 n)$ bits. Note that the polynomial degree in the runtime is rather large due to the use of the well known s-t reachability algorithm of Reingold [48]. Izumi and Otachi additionally mention that for planar graphs

there exists a polynomial-time and sublinear-space algorithm for finding a tree decomposition of a planar graph due to a simple recursive separator search with the result of Imai et al. [29], which present an algorithm for finding balanced separators of size $O(\sqrt{n})$ in planar graphs with sublinear space and polynomial runtime. Note that the algorithm of Imai et al. makes use of an embedding of the input graph and also uses the algorithm of Reingold. In contrast, we present an algorithm that computes a tree decomposition of width $O(n^{1/2+\epsilon})$ in time linear in the size of the tree decomposition using $O(n)$ bits. As the tree decomposition has $O(n)$ bags of size $O(n^{1/2+\epsilon})$ this results in a runtime of $O(n^{3/2+\epsilon})$.

In Section 2 we present concepts needed to understand the subsequent sections. In Section 3 we present our space-efficient graph-coarsening framework. Following that, in Section 4 and 5 we show how this scheme is used for finding separators, tree decompositions and constructing succinct encodings of planar graphs. In Section 6 we generalize our work to H-minor-free graphs. For space reasons most proofs can be found in the full version.

2 Background

We operate in the standard word RAM model of computation with word size $w = \Omega(\log n)$. This assumes the existence of read-only input memory, read/write working memory and write-only output memory. When we talk about space-usage we focus on bits used in the read/write working memory. This is a common setting for space-efficient algorithms.

We make use of common graph theoretic notations and terminology. Refer to Diestel [15] for more information. When using space-efficient graph algorithms the exact representation of the input graph is important. We assume that any input graph is given via *adjacency arrays* or an equivalent interface. Given a vertex u and index $i \leq \texttt{degree}(u)$ this allows us to determine in constant time the ith edge $\{u, v\}$ out of u. All our input graphs are assumed to be undirected with the vertices labeled from $1, \ldots, n$. Such a labeling is given implicitly by the order of the adjacency arrays. Given the use of adjacency arrays we store each undirected edge $\{u, v\}$ as two directed *arcs* uv and vu. For each arc uv we call vu the *co-arc of* uv. We assume the existence of so-called *crosspointers* that allow us to find the co-arc of an arc uv in constant time. Typically, this is realized by storing an index i in addition to the vertex v in u's adjacency array. The index i then indicates the position of u in v's adjacency array. We assume w.l.o.g. that every given graph G is connected as otherwise all our techniques can be done iteratively for each connected component of G. We assume that a given input graph is in read-only memory.

A *separator* S of a graph $G = (V, E)$ is a subset of V such that its removal from V divides V into non-empty sets $A \subset V$ and $B \subset V$ so that $\{A, S, B\}$ is a partition of V with the constraint that all paths from a vertex $u \in A$ to a vertex $v \in B$ contain at least one vertex of S. If $|A| < \alpha n$ and $|B| < \alpha n$ for some $\alpha < 1$, then S is called a *α-balanced separator* or simply *balanced separator*.

A family of graphs $\mathcal{G}$ that is closed under taking vertex-induced subgraphs satisfies the *$f(\cdot)$-separator theorem* exactly if, for constants $\alpha < 1$ and $\beta > 0$, each member $G \in \mathcal{G}$ has an α-separator S of size $|S| < \beta f(n)$. We say a family of graphs $\mathcal{G}$ is *separable* exactly if it satisfies the *n^c-separator theorem* for some $c < 1$. We say a graph is separable if it belongs to a separable family of graphs. For planar graphs there exist an $O(\sqrt{n})$-separator theorem with $\alpha = 2/3$ that runs in linear time [42], later extended to graphs of bound genus with the same runtime [21]. For minor-closed graph classes excluding the complete graph K_t on t vertices for some constant t there exists an $O(n^{(2-\epsilon)/3})$-separator theorem with runtime $O(n^{1+\epsilon} + m)$ [36]. Note that this includes H-minor-free graphs for any fixed graph H.

The definition of the separable graph families are not closed under taking minors, but vertex-induced subgraphs [11]. As we specifically construct minors F of a separable graph G at various points in this publication, we require the more restrictive property that F also belongs to the same family of separable graphs as G, which holds for all graph classes explicitly mentioned in the previous paragraph.

Due to Lipton et al. [41] is it known that a separable graph class $\mathcal{G}$ has *bounded density* d for some constant d. This means that each $G \in \mathcal{G}$ contains at most dn edges. In particular, we use the well-known fact, by Euler's Formula, that for a planar graph $G = (V, E)$ it holds that $|E| \leq 3|V| - 6$. For planar bipartite graphs a stronger bound $|E| \leq 2|V| - 4$ holds.

A *tree decomposition* of a graph $G = (V, E)$ consists of a tree T and a family $\mathcal{X}$ of subsets X_w (called *bags*) of V, one for each $w \in V(T)$, such that:

1. $\bigcup_{w \in V(T)} X_w = V$
2. for all $\{u, v\} \in E$ there exists $w \in V(T)$ such that $u, v \in X_w$
3. for all $w_1, w_2, w_3 \in V(T)$, if there is a path from w_1 to w_3 that contains w_2, then $X_{w_1} \cap X_{w_3} \subseteq X_{w_2}$.

The *width* of a tree decomposition is the size of the largest bag minus one. The treewidth of a graph G is the smallest width amongst all possible tree decompositions of G.

We make use of *indexable dictionaries*, which is a structure that supports constant-time rank-select queries over a bitvector. The $\texttt{rank}(i)$ operation counts the number of occurrences of bits set to 1 before the ith index and the $\texttt{select}(i)$ operation returns the index of the i*th* bit set to 1.

▶ **Lemma 1** ([47]). *Given a bitvector S of length ℓ there is an indexable dictionary on S that requires $o(\ell)$ additional bits, supports rank-select queries in constant time and can be constructed in $O(\ell)$ time.*

What can be thought of a dynamic alternative to indexable dictionaries is the so-called *choice dictionary* [23]. A choice dictionary is initialized for a universe $1, \ldots, \ell$ and supports constant time `insert`, `delete`, and `contains` operations, with the latter returning `true` exactly if an element of the universe is contained in the choice dictionary. Additionally, the choice dictionary offers the operation `choice` that returns an arbitrary member and iteration over its members. The iteration outputs all members of a choice dictionary in constant time per member. All operations run in constant time and the iteration over its members is linear in the number of members. The following lemma has been adapted from [33], with some minor rephrasing to make it more clear in the context of this paper.

▶ **Lemma 2.** *There is a succinct choice dictionary initialized for the universe $1, \ldots, \ell$ that occupies $\ell + o(\ell)$ bits and provides constant-time `insert`, `delete`, `contains` and `choice` operations and constant-time (per member) iteration. The choice dictionary can be initialized in $O(\ell)$ time.*

We make use of a folklore technique called *static space allocation* allowing us to store ℓ items of varying size compactly. The following description is adapted from [31]. Each item B_k occupies d_k bits for $k \in \{1, \ldots, \ell\}$. Denote by L the amount of bits all these items totally occupy. We want to store all these items with $L + o(L)$ bits such that we can access each item B_k in $O(1)$ time for $k \in \{1, \ldots, \ell\}$. In $O(\ell + L)$ time compute the sums $s_k = k + \sum_{j=1}^{k-1} d_j$ and an indexable dictionary over a bitvector S of size $\ell + L$ with $S[i] = 1$ exactly if $i = s_k$ for $k \in \{1, \ldots, \ell\}$. The location of B_k is then equivalent to $\texttt{select}_B(k) - k$. Refer to [9, 25, 31] for more information and detailed descriptions.

For traversing graphs we use standard breadth-first search (BFS), which puts a start vertex in a first queue, then the unprocessed neighbors of the first queue in a second queue, swaps the queues and repeats. It can easily be seen that space requirement is $O(n \log n)$ bits. More precise, the BFS uses $O(\ell \log n)$ bits where ℓ is the maximum number of vertices in a queue at any point of the BFS.

In this work we need to store subgraphs G' of a given graph $G = (V, E)$. Similar to the techniques used by Hagerup et al. [25] we are able to store G' using $O(n + m)$ bits. We store a subgraph G' of G via n choice dictionaries D_v for each vertex $v \in V(G)$. Each D_v has length d_v with d_v being the degree of v in G. The choice dictionaries are stored using static space allocation. A member i in D_v then indicates the existence of the ith arc out of v's adjacency array. Another choice dictionary D' of length n can be used to mark vertices of degree > 0. Iterating over the neighborhood of a vertex $v \in V(G')$ can be done in linear time of the degree of v in G'. This additionally allows dynamic insertion and deletion of edges as long as G' remains a subgraph of G, i.e., no new edges can be added. Note that we do not directly allow the deletion of vertices. Instead, when a vertex has degree 0 in G' (due to the deletion of all incident edges) it is marked in D'. This allows to iterate over all vertices $V' = \{v \in V(G') | d_v > 0\}$ in $O(|V'|)$ time with d_v being the degree of v in G'. Additionally, an arbitrary vertex $v \in V(G')$ with $d_v > 0$ can be obtained via $D'.\texttt{choice}()$. We refer to such a structure as a *dynamic subgraph G' of G*. Note that a dynamic subgraph G' can be used to direct an edge $\{u, v\} \in E(G')$ by deleting only the arc uv or vu in G'. Using this we are able to implement the next lemma in $O(n)$ time and $O(n)$ bits using the same algorithm as described in [10].

▶ **Lemma 3.** *Let G be a separable graph. We can obtain the directed graph G' of G such that each vertex of G' has bounded in-degree (out-degree) in $O(n)$ time and $O(n)$ bits.*

3 Graph Coarsening Framework for Planar Graphs

In this section we outline a strategy for coarsening a planar graph $G = (V, E)$. The idea is to create a specific type of partition $\mathcal{P}$ of V, which we call *cloud partition* that induces a unique minor F of G, defined later. We refer to each $C \in \mathcal{P}$ as a *cloud*. Thus, a cloud is a set of vertices. In the following we describe the exact specifications that define such a cloud partition. For each $C \in \mathcal{P}$, C induces a connected subgraph of G and $|C| \leq \lceil c \log n \rceil$ for an arbitrary, but fixed constant c. We differentiate between different types of clouds $C \in \mathcal{P}$, which we define after introducing some terminology. Let $C_1, C_2 \in \mathcal{P}$ with $C_1 \neq C_2$. We call an edge $\{u, v\} \in E$ a *border edge* exactly if $u \in C_1$ and $v \in C_2$. We then refer to C_1 and C_2 as *adjacent* or *neighbors* and as *incident to* $\{u, v\}$. Furthermore, we call C a *big cloud* if $|C| = \lceil c \log n \rceil$ and a *small cloud* otherwise. We call a small cloud C a *leaf cloud* if C is adjacent to one cloud, a *bridge cloud* if it is adjacent to two clouds and a *critical cloud* if it is adjacent to at least three clouds. We call two clouds C_1, C_2 adjacent to a bridge cloud B *connected by* B. A cloud partition is created by starting with the following scheme: Initially, mark all vertices as unvisited. Run a BFS from an arbitrary unvisited vertex v. The BFS only traverses vertices marked unvisited. Each time an unvisited vertex is traversed, it is marked visited. The BFS runs until either $\lceil c \log n \rceil$ vertices are marked visited or no unvisited vertices can be reached by the BFS. In the first case a big cloud is found and in the second case a small cloud is found. Repeat this until no unvisited vertices remain, always starting a new BFS at an arbitrary unvisited vertex. Only a partition created in this way is referred to as a cloud partition. By fixing the search for unvisited vertices and the BFS algorithm, each graph has a fixed cloud partition. The next two observations are derived directly from the process of creating a cloud partition.

▶ **Observation 4.** *Let $\mathcal{P}$ be an arbitrary cloud partition created for a graph G. Then no two small clouds $C_1, C_2 \in \mathcal{P}$ are adjacent.*

▶ **Observation 5.** *Let $\mathcal{P}$ be an arbitrary cloud partition created for a graph G. Since each big cloud has size $\lceil c \log n \rceil$ for a constant c, $\mathcal{P}$ contains at most $n/\lceil c \log n \rceil$ big clouds.*

For planar graphs, the number of critical clouds can be bounded similarly. The proof is based on the fact that a planar graph has $O(n)$ edges.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a cloud partition created for a planar graph G containing k big clouds. Then $\mathcal{P}$ contains $O(k)$ critical clouds.*

Proof. Consider the graph $F = (V', E')$ constructed by contracting the vertices of each cloud $C \in \mathcal{P}$ to a single vertex $v \in C$. The graph F is a minor of G with $|V'| = |\mathcal{P}|$ and each vertex $v \in V'$ represents a single cloud $C_v \in \mathcal{P}$. Two vertices $v, u \in V'$ are adjacent in F exactly if C_v and C_u are adjacent. For brevity's sake we refer to vertices of F introduced for critical clouds as critical vertices, and vertices introduced for big clouds as big vertices. Assume that all vertices that are not big vertices or critical vertices are removed in F and all edges between big vertices are removed as well. This turns F into a bipartite planar graph due to Observation 4, with big vertices on one side of the bipartition, and small and critical on the other. It is well-known that $|E'| \leq 2|V'| - 4$ for bipartite planar graphs. Denote by ℓ the number of critical vertices and k the number of big vertices in F. Note that for each critical vertex $v \in V'$ it holds $\texttt{degree}(v) \geq 3$. As such, $|E'| \geq 3\ell$. Using this the number of critical vertices in F is bounded via $3\ell \leq 2|V'| - 4 = 2k + 2\ell - 4$ and thus $\ell \leq 2k - 4$. (Note that the addition of any edges between big vertices in F can only tighten this bound.) As the number of critical vertices in F is equal to the number of critical clouds in $\mathcal{P}$ we have shown that there are at most $2k - 4$ critical clouds in $\mathcal{P}$. ◀

The next corollary follows together with Obs. 5.

▶ **Corollary 7.** *Let $\mathcal{P}$ be a cloud partition created for a planar graph G. Then $\mathcal{P}$ contains $O(n/\log n)$ critical clouds.*

Note that for leaf or bridge clouds no such bound exists as a cloud partition can contain $O(n)$ leaf and bridge clouds. Extreme examples include a cloud partition $\mathcal{P}$ with one big cloud C and $n - |C|$ leaf clouds, all adjacent to C and containing only one vertex. For bridge clouds a similar example partition $\mathcal{P}$ can be constructed, with two big clouds $C_1, C_2 \in \mathcal{P}$ and $n - (|C_1| + |C_2|)$ bridge clouds, all adjacent to both C_1 and C_2.

Next we focus on the construction of a specific weighted minor F of G with weights $w(v)$ assigned to each node $v \in V(F)$. Intuitively, F represents a minor of G that is constructed by repeatedly contracting the vertices in one or more clouds, and the weights keep track of the number of vertices that have been contracted. We call such an F a *structure-maintaining minor* of G, with the formal definition outlined shortly. As each such minor is constructed specifically for a cloud partition P we say that F is *induced by* P. We now define the properties of such a minor and follow with a sketch of a construction. Let F be a structure-maintaining minor induced by a cloud partition $\mathcal{P}$. Denote by $\mathcal{C}_u$ the set of clouds contracted to a node $u \in V(F)$. Each node $u \in V(F)$ is assigned a weight $w(u) = \sum_{C \in \mathcal{C}_u} |C|$. For each u one of the following two properties holds: (1) $|\mathcal{C}_u| = 1$, $G[C]$ is connected for $C \in \mathcal{C}_u$ and $w(u) = \lceil c \log n \rceil$ for some fixed constant c or (2) each $C \in \mathcal{C}_u$ is adjacent to exactly the same clouds, $G[\bigcup_{C \in \mathcal{C}_u} C]$ contains $|\mathcal{C}_u|$ connected components. Additionally, $\{v, w\} \in E(F)$ exactly if there exist adjacent clouds C_v and C_w with $C_v \in \mathcal{C}_v$ and $C_w \in \mathcal{C}_w$.

We now outline the construction of F. Initially, $F = (V' = \emptyset, E' = \emptyset)$. For each cloud C that is big or critical, add a node v to V' with $w(v) = |C|$. Add edges between $u, v \in V'$ to E' exactly if the clouds u and v represent are adjacent. We call v a *big node* if it was added to V' for a big cloud and *critical node* if it was added for a critical cloud. For each pair of big clouds $C_1, C_2 \in \mathcal{P}$ connected via one or more bridge clouds, let $\mathcal{B}_{C_1,C_2} \subset \mathcal{P}$ be the set of all such bridge clouds. Add a single node v to V' adjacent to the nodes u, w added to V' for C_1, C_2 and set $w(v) = \sum_{B \in \mathcal{B}_{C_1,C_2}} |B|$. We refer to such a node v as a *meta-bridge node*. For each big cloud $C \in \mathcal{P}$ adjacent to one or more leaf clouds, denote by $\mathcal{L}_C \subset \mathcal{P}$ the set of all leaf clouds adjacent to C. For each such C add a single node v to V' adjacent to the node u added to V' for C with $w(v) = \sum_{L \in \mathcal{L}_C} |L|$. We call such a node v a *meta-leaf node*.

Since we have only $O(n/\log n)$ big nodes, we also have only $O(n/\log n)$ critical, meta-bridge and meta-leaf nodes, expressed explicitly in the following lemma. Note that the weight of a meta-bridge or meta-leaf node can be bounded only by n. We next bound the number of nodes and edges of F.

▶ **Lemma 8.** *Let $\mathcal{P}$ be a cloud partition constructed for a planar graph G and $F = (V', E')$ the structure-maintaining minor induced by $\mathcal{P}$. Then $|V'| = O(n/\log n)$ and $O(|E'|) = O(|V'|)$.*

Proof. As F is a minor of G it is planar as well and so it holds that $O(|E'|) = O(|V|')$. From Obs. 5 we know that the number of big nodes in F is $k = O(n/\log n)$. From the construction of meta-leaf nodes it directly follows that each meta-leaf node is adjacent to exactly one big node and vice-versa. Therefore there are at most k meta-leaf nodes in V'. For meta-bridge nodes the proof follows analogous, consider two big nodes u, v that are adjacent to the same meta-bridge node w. Then there is no meta-bridge node $w \neq w$ adjacent to both u and v, as per the definition of meta-bridge nodes, there exist at most one meta-bridge node between two big nodes. By the fact that F is planar it follows that there are at most $O(k) = O(n/\log n)$ meta-bridge nodes in V'. Putting it all together we arrive at $|V'| = O(n/\log n)$. ◀

In the following we describe a data structure for a cloud partition and show how it can be constructed and stored in $O(n)$ time and $O(n)$ bits. We refer to this data structure as *cp-structure*. A cp-structure constructed for a planar connected graph G admits the following operations:

- `type`(v): Given a vertex v outputs the type of cloud v belongs to (big, small, critical, bridge or leaf) in $O(1)$ time.
- `cloud`(v): Given a vertex v returns all vertices of the cloud C in which v is contained in $O(|C|)$ time and $O(|C| \log n)$ space.
- `border`(v, k): Outputs if the k-th arc out of v's adjacency array is part of a border edge in $O(1)$ time, with $v \in V(G)$.

Additionally, the structure allows access to the subgraph G' of G induced by all non-border edges of $\mathcal{P}$. The graph G' admits adjacency array access to its vertices and edges. The following lemma describes the runtime and space usage of constructing a cloud partition and a cp-structure.

▶ **Lemma 9.** *Let G be a planar connected graph. We can compute a cloud partition $\mathcal{P}$ of G and a cp-structure of $\mathcal{P}$ in $O(n)$ time and $O(n)$ bits.*

Next we show how to construct a structure-maintaining minor F induced by a cloud partition $\mathcal{P}$ of a graph G in $O(n)$ time and $O(n)$ bits of space. By Lemma 8, F can be stored in $O(n)$ bits by adjacency lists. We additionally store a bi-directional mapping from each big/critical cloud C to the node $v \in V(F)$ added to $V(F)$ to represent C. This mapping

is stored by a pointer at v to a single vertex $v' \in C$ and a pointer at v' to v. As there are $O(n/\log n)$ pointers to store for this bi-directional mapping we use standard pointers of size $\Theta(\log n)$. To store the pointers from the direction of a vertex $v \in V(G)$ we use static-space allocation. Note that the choice of the vertex to store these pointers (per cloud) is arbitrary, but should be fixed. We choose the vertex with the lowest label. For meta-bridge and meta-leaf nodes $v \in V(F)$ we store a pointer from v to the lowest labeled vertex v' amongst all clouds represented by the meta-bridge or meta-leaf node. The details are described in the proof of the next lemma. Next, we define a special operation on F.

- `expand`(v): Given a node $v \in V(F)$ with weight $w(v)$, first determine the cloud $\mathcal{C}$ mapped to v and then return iteratively all vertices part of the clouds $C \in \mathcal{C}$.

Later we make use of the `expand` operation when translating solutions for problems such as finding separators from F to a solution to G. The key difficulty is implementing the `expand` operation for a meta bridge v. As the clouds represented by v may induce up to $n - O(\log n)$ connected components in G, and we store only a pointer to the lowest labeled vertex $u \in V(G)$ amongst all such clouds represented by v, we are unable to simply output each cloud without additional information, or we must traverse (in the worst case) the entire graph G. The idea is to store a spanning tree that spans all vertices in clouds represented by v. As mentioned, since these clouds each induces a connected component, a spanning tree with these clouds does not exist. Therefore, we create a spanning tree by additionally using the vertices of one adjacent big cloud. Traversing all vertices represented by v can now be done by traversing the respective spanning tree. The key observation for this is that we can direct the edges of F in such a way that each vertex has up to $c = O(1)$ outgoing edges (Lemma 3). By this we are able to store all spanning trees in $O(n)$ bits due to the fact each big cloud is used only by c spanning trees. Concerning the problem of having c spanning trees that use the vertices of one big cloud, i.e., that overlap, our approach is to assign each spanning tree a color from a set of c colors. These spanning trees are then stored in c dynamic subgraphs, with each subgraph storing all spanning trees of one color.

▶ **Lemma 10.** *Let G be a connected planar graph and $\mathcal{P}$ a cloud partition of G given as a cp-structure. We can construct the structure-maintaining minor F induced by $\mathcal{P}$ in $O(n)$ time and bits such that the* ***expand*** *operation on $v \in V(F)$ runs in $O(w(v) + \log n)$ time.*

We refer to the combined data structure of Lemma 9 and Lemma 10 as a *cloud decomposition $(F, \mathcal{P})$ of G*.

4 Applications: Succinctly Encoded Planar Graphs

In this section we show how a cloud decomposition $(F, \mathcal{P})$ constructed for a planar graph G can be used to find a balanced separator of size $O(\sqrt{n \log n})$ in $O(n/\log n)$ time and $O(n)$ bits, which in turn can be used to construct a succinct encoding of planar graphs, described later. Afterwards we show how such a cloud decomposition can be used together with the search for balanced separators to compute a tree decomposition with width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ of a planar graph G. We make use of an $O(n)$-time $O(n \log n)$-bits algorithm for finding 2/3-balanced separators of size $O(\sqrt{n})$ in weighted planar graphs as long as no vertex has a weight more than 1/3 times the total weight [38, 42, 43]. The next theorem shows how a balanced separator can be constructed for a planar graph with the help of a cloud decomposition. The idea is to run a slightly modified standard algorithm for finding balanced separators S on the structure-maintaining minor F and translating S to a separator S' of G.

▶ **Theorem 11.** *Let $G = (V, E)$ be a planar graph and $(F = (V', E'), \mathcal{P})$ a cloud decomposition of G. We can construct a balanced separator S of G with size $O(\sqrt{n \log n})$ in $O(n/\log n)$ time using $O(n)$ bits.*

Due to Blelloch and Farzan [11] there exists a succinct encoding of an arbitrary separable graph $G = (V, E)$ that provides constant-time adjacency, degree and neighborhood queries. The encoding is constructed via recursive application of a separator algorithm such that V is split into two sets A and B via a separator S. This recursion is continued for $G_a = G[A \cup S]$ and $G_b = G[B \cup S]$ until the remaining graphs are of size at most $\log^\delta n$ for a graph class specific constant δ, which they refer to as *mini graphs*. In several papers, what is constructed here is referred to as a *separator hierarchy*. For technical reasons the edges between vertices of S are only included in G_a and not G_b. The graph G is then encoded by a combination of these mini graphs, which in turn are further decomposed by the same recursive separator search into *micro graphs*, which are then small enough to be handled via lookup tables encoding every possible micro graph and in turn are used to encode the mini graphs. Constructing these mini and micro graphs can be done in $O(n)$ time for a planar graph G using the techniques described by Blelloch and Farzan combined with the algorithm of Goodrich for constructing a separator hierarchy [22], but we want to use only $O(n)$ bits and maintain the runtime. The main result of [11] is summed up by the next theorem.

▶ **Theorem 12** ([11])**.** *Any family of separable graphs with entropy $\mathcal{H}(n)$ can be succinctly encoded in $\mathcal{H}(n) + o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time.*

One important point of the construction is the fact that the number of *duplicate vertices* in each mini graph is bounded. A duplicate is a vertex that is contained in more than one mini graph. A vertex becomes a duplicate any time it is part of a separator S, as it is then contained in both G_a and G_b. The exact bound is defined by the next lemma.

▶ **Lemma 13** ([11])**.** *The number of mini graphs is $\Theta(n/\log^\delta n)$. The total number of duplicates among mini graphs is $O(n/\log^2 n)$. The sum of number of vertices of mini graph together is $n + O(n/\log^2 n)$.*

Our goal is to execute the recursive decomposition of the graph G via a cloud decomposition $(F, \mathcal{P})$ by recursively searching for separators in F until the entire weight of the remaining graphs is less than $\log^\delta n$. These small subgraphs of F then are expanded to be exactly the mini graphs of G we aim to find. For each mini graph a new cloud decomposition is constructed, and the recursive separator search is repeated for each mini graph to construct the micro graphs.

▶ **Lemma 14.** *Let G be a planar graph and $(F, \mathcal{P})$ a cloud decomposition of G. We can output all mini graphs of G in $O(n)$ time.*

The bound on the number of duplicates due to Lemma 13 can be upheld with some care.

▶ **Lemma 15.** *Let G be a planar graph and $(F, \mathcal{P})$ a cloud decomposition of G. The total number of duplicate vertices among the mini graphs of the cloud decomposition is $O(n/\log n)$.*

The full encoding in $O(n)$ time can be done by using the algorithm of Lemma 14 to output all mini graphs. For every mini graph G_m we again construct a cloud decomposition in linear time and use the algorithm of Lemma 14 to find the micro graphs of G_m. All micro graphs are encoded using the table lookup scheme outlined in [11]. All other structures

needed for the encoding can be constructed in $O(n)$ time and $O(n)$ bits easily, as they are simple indexable dictionaries over vertices of the micro graphs and mini graphs or small lookup tables which can be initialized in time linear in their size, which is $O(n)$. From this description the next theorem follows.

▶ **Theorem 16.** *A planar graph G with entropy $\mathcal{H}(n)$ can be succinctly encoded in $\mathcal{H}(n)+o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time. The encoding can be constructed in $O(n)$ time using $O(n)$ bits.*

5 Applications: Planar Tree Decompositions

We next present a simple modification of the recursive separator search of Lemma 14 using standard techniques to output a tree decomposition of G. Let $(F, \mathcal{P})$ be a cloud decomposition of G. Each balanced separator S of F induces a bag in a tree decomposition of F via the following recursive relation. Let F' be the input graph used in the recursive calls, initially $F' = F$. Additionally, maintain a set X containing vertices contained in separators found in previous recursive calls, initially $X = \emptyset$. When a separator S is found for F that partitions $V(F)$ into three sets $\{A, S, B\}$ output the next bag of the tree decomposition of F as $S \cup X$. Continue the recursion for the input $F'[S \cup A]$ and $X = (X \cup S) \cap A$ and the input $F'[S \cup B]$ and $X = (X \cup S) \cap B$. Note that this tree decomposition of F has width $O(\sqrt{|V(F)|})$. This tree decomposition can easily be expanded to a tree decomposition of G by expanding all vertices of a bag B to their respective clouds. The width of this expanded tree decomposition is $O(\sqrt{|V(F)|}\log n) = O(n^{1/2+\epsilon})$ for any $\epsilon > 0$. Each bag can be output in $O(n^{1/2+\epsilon})$ time and there are $O(n)$ bags. This description allows the next corollary.

▶ **Corollary 17.** *Let G be a planar graph. We can compute and output the bags of a tree decomposition with width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ in time linear in the size of the tree decomposition $O(n^{3/2+\epsilon})$ using $O(n)$ bits.*

6 Generalizing to H-Minor-Free Graphs

To generalize the previous results to H-minor-free graphs from planar graphs we must generalize cloud partitions and their induced minors. Recall that a cloud partition $\mathcal{P}$ for a planar graph G contains $O(n/\log n)$ critical clouds. Critical clouds are exactly those of size $< c \log n$ and with ≥ 3 adjacent big clouds. We define a ϕ-critical cloud as a small cloud adjacent to $\geq \phi$ big clouds. The idea of the proof is the same as for the proof of Lemma 6, but we now have a bound ϕ that depends on the graph class. The existence of the bound ϕ is due to the face that a separable graph $G = (V, E)$ has bound density d, i.e., $|E| \leq d|V|$ for some fixed constant d. In particular we show that setting ϕ as $d + 1$ gives us the desired properties outlined in the following.

▶ **Lemma 18.** *Let $\mathcal{G}$ be a H-minor-free family of graphs for some fixed graph H and let d be the maximal density of a graph in $\mathcal{G}$. There exists a cloud partition $\mathcal{P}$ for each graph $G \in \mathcal{G}$ such that $\mathcal{P}$ contains $O(n/\log n)$ ϕ-critical clouds for $\phi = d + 1$.*

It remains to show how small clouds adjacent to $< \phi$ big clouds are to be handled. For planar graphs such clouds are exactly the bridge and leaf clouds. To generalize from planar graphs to H-minor-free graphs we must in turn generalize bridge clouds similarly to the generalization from critical to ϕ-critical clouds. We call such generalized bridge clouds *ϕ-bridge clouds*, which we define as small clouds adjacent to $< \phi$, but > 2 big clouds.

Analogous to regular bridge clouds, there can be $O(n)$ such clouds in a cloud partition for an H-minor-free graph. We refer to cloud partitions with the additional labeling of clouds as ϕ-critical and ϕ-bridge as a *generalized cloud partition* $\mathcal{P}$.

To construct the structure-maintaining minor F induced by a generalized cloud partition $\mathcal{P}$ we can use the same strategy for bridge clouds and leaf clouds. For ϕ-critical clouds the strategy also remains the same as for critical clouds in regular cloud partitions. For ϕ-bridge clouds we introduce a generalized version of the meta-bridge node to F, which we call ϕ-meta-bridge node. Note that each ϕ-meta-bridge node has degree $< \phi$ in F. For adding all ϕ-meta bridge nodes we iterate over $i \in (3, \dots, \phi - 1)$. For each i we add all ϕ-meta bridges with degree i to $V(F)$. Adding all degree-i ϕ-meta bridge nodes v takes $O(in)$ time as for each neighbor of v the respective clouds must be explored. Thus, an overall time of $O(\phi^2 n)$ is needed during the construction. As described above Lemma 10, we store spanning trees for meta-bridge nodes and meta-leaf nodes. We store and construct the exact same spanning trees for the ϕ-meta-bridge nodes. These spanning trees are stored in c different dynamic subgraphs, with c being some constant, as described in Lemma 3. Storing these dynamic subgraphs takes $O(cn) = O(\phi n)$ bits of space.

▶ **Lemma 19.** *Let G be a H-minor-free graph for some fixed graph H with a generalized cloud partition $\mathcal{P}$. We can construct the minor induced by $\mathcal{P}$ in $O(\phi^2 n)$ time and $O(\phi n)$ bits with ϕ being a graph class dependent constant.*

Analogous for planar graphs, we call the combination of a generalized cloud partition $\mathcal{P}$ and a minor induced by $\mathcal{P}$ a *ϕ-cloud decomposition*. This allows us to generalize Theorem 11, Theorem 12 and Corollary 17 to H-minor-free graphs using such generalized cloud partitions. For generalizing the proofs of these theorems and the corollary we only must handle the new case of ϕ-meta-bridge nodes exceeding the weight threshold during the separator search, as all other cases work analogous. Recall that we must handle the case when a node $v \in V(F)$ has too large of a weight, and thus no balanced separator can be found. A new additional case now arises when a ϕ-meta-bridge node v exceeds the weight threshold which is handled exactly the same as meta-bridge and meta-leaf nodes. In detail, the neighborhood S of v in F is a separator that separates v from $V(F) \setminus S$. Expanding S to a separator S' of G then separates all vertices in clouds represented by v from the rest of the graph. The balanced separator S' then contains $O(\phi \log n)$ vertices. For the following theorems we make use of the linear time $O(n^{2/3})$-separator theorem for H-minor-free graphs [36].

▶ **Theorem 20.** *Let G be a H-minor-free graph for some fixed graph H, let ϕ be a graph class dependent constant and $(F, \mathcal{P})$ a generalized cloud partition of G. We can compute a balanced separator S of G with size $O(n^{2/3+\epsilon})$ for any $\epsilon > 0$ in $O(n/\log n)$ time using $O(n)$ bits.*

▶ **Theorem 21.** *A H-minor-free graph for some fixed graph H with entropy $\mathcal{H}(n)$ can be succinctly encoded in $\mathcal{H}(n) + o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time. The construction of the encoding of takes $O(\phi^2 n)$ time and uses $O(\phi n)$ bits with ϕ a graph class dependent constant.*

▶ **Corollary 22.** *Let G be a H-minor-free graph for some fixed graph H, ϕ a graph class dependent constant and $(F, \mathcal{P})$ a generalized cloud partition of G. We can compute a tree decomposition of G with width $O(n^{2/3+\epsilon})$ for any $\epsilon > 0$ in $O(n^{5/3+\epsilon})$ time using $O(n)$ bits.*

References

1 Luca Castelli Aleardi and Olivier Devillers. Array-based compact data structures for triangulations: Practical solutions with theoretical guarantees. *J. Comput. Geom.*, 9(1):247–289, 2018. doi:10.20382/jocg.v9i1a8.

2 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Dynamic updates of succinct triangulations. In *Proceedings of the 17th Canadian Conference on Computational Geometry, CCCG'05, University of Windsor, Ontario, Canada, August 10-12, 2005*, pages 134–137, 2005. URL: http://www.cccg.ca/proceedings/2005/45.pdf.

3 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Succinct representation of triangulations with a boundary. In *Algorithms and Data Structures*, pages 134–145. Springer, 2005.

4 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Succinct representations of planar maps. *Theor. Comput. Sci.*, 408(2-3):174–187, 2008. doi:10.1016/j.tcs.2008.08.016.

5 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Inf. Comput.*, 189(1):117–134, 2004. doi:10.1016/j.ic.2003.09.002.

6 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $o(n)$ bits. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation*, pages 553–564, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-13075-0_44.

7 Bas Fagginger Auer and Rob H Bisseling. Graph coarsening and clustering on the GPU. *Graph Partitioning and Graph Clustering*, 588:223, 2012.

8 Bas O Fagginger Auer and Rob H Bisseling. A GPU algorithm for greedy graph matching. In *Facing the Multicore-Challenge II*, pages 108–119. Springer, 2012.

9 Niranka Banerjee, Sankardeep Chakraborty, and Venkatesh Raman. Improved space efficient algorithms for BFS, DFS and applications. In *Computing and Combinatorics*, pages 119–130. Springer International Publishing, 2016.

10 Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. Compact representations of separable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 679–688. Society for Industrial and Applied Mathematics, 2003.

11 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *Proceedings of the 21st Annual Conference on Combinatorial Pattern Matching*, CPM 2010, pages 138–150, 2010.

12 Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks, 2021. arXiv:2102.01350.

13 Cédric Chevalier and Ilya Safro. Comparison of coarsening schemes for multilevel graph partitioning. In Thomas Stützle, editor, *Learning and Intelligent Optimization, Third International Conference, LION 3, Trento, Italy, January 14-18, 2009. Selected Papers*, volume 5851 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2009. doi:10.1007/978-3-642-11169-3_14.

14 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 506–515. Society for Industrial and Applied Mathematics, 2001.

15 Reinhard Diestel, Alexander Schrijver, and Paul D. Seymour. Graph theory, 2007.

16 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 383–392, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591865.

17 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient Basic Graph Algorithms. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 288–301, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2015.288.

18 Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. Faster graph embeddings via coarsening. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2953–2963. PMLR, 13–18 July 2020. URL: https://proceedings.mlr.press/v119/fahrbach20a.html.

19 Volodymyr Floreskul, Konstantin Tretyakov, and Marlon Dumas. Memory-efficient fast shortest path estimation in large social networks. *Proceedings of the International AAAI Conference on Web and Social Media*, 8:91–100, 2014. URL: https://ojs.aaai.org/index.php/ICWSM/article/view/14532.

20 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

21 John R Gilbert, Joan P Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5(3):391–407, 1984. doi:10.1016/0196-6774(84)90019-1.

22 Michael T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, December 1995. doi:10.1006/jcss.1995.1076.

23 Torben Hagerup. A constant-time colored choice dictionary with almost robust iteration. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 64:1–64:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.64.

24 Torben Hagerup. Space-efficient dfs and applications to connectivity problems: Simpler, leaner, faster. *Algorithmica*, 82(4):1033–1056, 2020. doi:10.1007/s00453-019-00629-x.

25 Torben Hagerup, Frank Kammer, and Moritz Laudahn. Space-efficient Euler partition and bipartite edge coloring. *Theoretical Computer Science*, 754:16–34, 2019. Algorithms and Complexity. doi:10.1016/j.tcs.2018.01.008.

26 Xin He, Ming-Yang Kao, and Hsueh-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM J. Comput.*, 30(3):838–846, 2000. doi:10.1137/S0097539799359117.

27 Emilie Hogan, John R. Johnson, and Mahantesh Halappanavar. Graph coarsening for path finding in cybersecurity graphs. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13. Association for Computing Machinery, 2013. doi:10.1145/2459976.2459984.

28 John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, October 1974. doi:10.1145/321850.321852.

29 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N Variyam Vinodchandran, and Osamu Watanabe. An $O(n^{1/2+\epsilon})$-space and polynomial-time algorithm for directed planar reachability. In *2013 IEEE Conference on Computational Complexity*, pages 277–286. IEEE, 2013. doi:10.1109/CCC.2013.35.

30 Taisuke Izumi and Yota Otachi. Sublinear-Space Lexicographic Depth-First Search for Bounded Treewidth Graphs and Planar Graphs. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.67.

31 Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. *Algorithmica*, 81(3):1180–1204, 2019. doi:10.1007/s00453-018-0464-z.

32 Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *CoRR*, abs/1907.00676, 2019. arXiv:1907.00676.

33 Frank Kammer and Andrej Sajenko. Simple 2^f-Color Choice Dictionaries. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2018.66.

34 G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Supercomputing '95:Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, pages 29–29, 1995. doi:10.1109/SUPERC.1995.242800.

35 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, 20(1):359–392, 1998.

36 Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 153–162, 2010. doi:10.1109/FOCS.2010.22.

37 Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Appl. Math.*, 58(3):239–252, 1995. doi:10.1016/0166-218X(93)E0150-W.

38 Philip Klein and Shay Mozes. Separators in planar graphs, 2021. URL: http://planarity.org/Klein_separators_in_planar_graphs.pdf.

39 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 505–514, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488672.

40 T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988. doi:10.1109/SFCS.1988.21958.

41 Richard J. Lipton, Donald J. Rose, and Robert E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16:346–358, 1977.

42 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

43 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

44 Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, January 1997. doi:10.1145/256292.256294.

45 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

46 François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer Berlin Heidelberg, 1996.

47 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43–es, November 2007. doi:10.1145/1290672.1290680.

48 Omer Reingold. Undirected st-connectivity in log-space. *Electronic Colloquium on Computational Complexity (ECCC)*, January 2004. doi:10.1145/1060590.1060647.

49 Peter Sanders and Christian Schulz. Advanced multilevel node separator algorithms. In *Experimental Algorithms*, pages 294–309. Springer International Publishing, 2016.

50 Ben Strasser, Dorothea Wagner, and Tim Zeitz. Space-Efficient, Fast and Exact Routing in Time-Dependent Road Networks. In *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81:1–81:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.81.

Subquadratic Weighted Matroid Intersection Under Rank Oracles

Ta-Wei Tu
National Taiwan University, Taipei, Taiwan

Abstract

Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over an n-element integer-weighted ground set V, the weighted matroid intersection problem aims to find a common independent set $S^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing the weight of S^*. In this paper, we present a simple deterministic algorithm for weighted matroid intersection using $\widetilde{O}(nr^{3/4} \log W)$ rank queries, where r is the size of the largest intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$ and W is the maximum weight. This improves upon the best previously known $\widetilde{O}(nr \log W)$ algorithm given by Lee, Sidford, and Wong [FOCS'15], and is the first subquadratic algorithm for polynomially-bounded weights under the standard independence or rank oracle models. The main contribution of this paper is an efficient algorithm that computes shortest-path trees in weighted exchange graphs.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Matroids and greedoids

Keywords and phrases Matroids, Weighted Matroid Intersection, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.63

Acknowledgements I would like to thank Prof. Hsueh-I Lu for advising the project and helpful suggestions on the writing and notation. I also thank Min-Yen Tsai for proof-reading an initial draft of this paper and the anonymous reviewers of ISAAC 2022 for their useful comments.

1 Introduction

Matroid Intersection. A matroid is an abstract structure that models the notion of independence on a given ground set V. In particular, a subset $S \subseteq V$ is either *independent* or *dependent*, such that the family of independent sets is well-structured (see Section 2 for a complete definition). Matroids model many fundamental combinatorial objects, and examples of independent sets of a matroid include acyclic subgraphs of an undirected graph and linearly independent rows of a matrix. One of the most important optimization problems related to matroids is *matroid intersection*: Given two matroids, we would like to find a set with the largest cardinality that is independent in both matroids. Similarly, in the weighted case, each element in the ground set is associated with an integer weight, and the weighted matroid intersection problem is to find the maximum-weight common independent set. These problems have been extensively studied in the past since they capture many combinatorial optimization problems such as bipartite matching and colorful spanning trees.

Oracle Model. Since we are dealing with general matroids without additional constraints, we have to specify a way of reading the description of the two matroids. One way is to express them directly by reading the truth table of independence. However, that would require an exponentially-sized input. Instead, we are given oracle access to the matroids, which gives us information about a queried set $S \subseteq V$. Standard oracles include the *independence oracle*, which returns whether S is independent in $O(\mathcal{T}_{\mathsf{ind}})$ time, and the *rank oracle*, which returns the rank, i.e., the size of the largest independent subset, of S in $O(\mathcal{T}_{\mathsf{rank}})$ time. In this paper, we focus on the stronger rank oracle model.

Prior Work. Polynomial-time algorithms for both the weighted and unweighted matroid intersection problems have long been designed and improved. For the unweighted case, Edmonds [10, 11, 12], Lawler [17], and also Aigner and Dowling [1] gave algorithms that run in $O(nr^2 \cdot \mathcal{T}_{\mathsf{ind}})$ time. Here, n denotes the number of elements in V and r denotes the size of the largest intersection of the two matroids. Cunningham [7] obtained an $O(nr^{3/2} \cdot \mathcal{T}_{\mathsf{ind}})$ algorithm using the "blocking-flow" idea. Lee, Sidford, and Wong [18] gave quadratic algorithms using the cutting-plane method, running in $\widetilde{O}(nr \cdot \mathcal{T}_{\mathsf{rank}} + n^3)$ and $\widetilde{O}(n^2 \cdot \mathcal{T}_{\mathsf{ind}} + n^3)$ times[1], respectively. This also gives rise to the "quadratic barrier" of matroid intersection: most previous algorithms involve building exchange graphs that contain $\Theta(nr)$ edges explicitly and therefore cannot go beyond quadratic time. Chakrabarty, Lee, Sidford, Singla, and Wong [4] were the first to partially break the barrier. They obtained $(1-\epsilon)$-approximation algorithms running in $\widetilde{O}(n/\epsilon \cdot \mathcal{T}_{\mathsf{rank}})$ and $\widetilde{O}(n^{3/2}/\epsilon^{3/2} \cdot \mathcal{T}_{\mathsf{ind}})$ times and an $\widetilde{O}(n\sqrt{r} \cdot \mathcal{T}_{\mathsf{rank}})$ exact algorithm. One of the major components of Chakrabarty et al.'s improvements is to show that edges in exchange graphs can be efficiently discovered using binary search (this was discovered independently by Nguyễn [19]). This technique also allows them to obtain improved $\widetilde{O}(nr \cdot \mathcal{T}_{\mathsf{ind}})$ exact algorithms. Combining the approximation algorithm and a faster augmenting-path algorithm, Blikstad, van den Brand, Mukhopadhyay, and Nanongkai [3] broke the quadratic barrier completely by giving an $\widetilde{O}(n^{9/5} \cdot \mathcal{T}_{\mathsf{ind}})$ exact algorithm. This result was later optimized by Blikstad [2] to $\widetilde{O}(nr^{3/4} \cdot \mathcal{T}_{\mathsf{ind}})$ by improving the approximation algorithm to run in $\widetilde{O}(n\sqrt{r}/\epsilon \cdot \mathcal{T}_{\mathsf{ind}})$ time.

For the weighted case, the blocking flow idea does not seem to apply anymore. Frank [13] obtained an $O(nr^2 \cdot \mathcal{T}_{\mathsf{ind}})$ algorithm by characterizing the optimality of a common independent set using weight splitting. Fujishige and Zhang [14] improved the running time to $\widetilde{O}(nr^{3/2} \log W \cdot \mathcal{T}_{\mathsf{ind}})$ by solving a more general *independent assignment* problem using a scaling framework. The same bound was achieved by Shigeno and Iwata [23] and also by Gabow and Xu [15]. Lee, Sidford, and Wong's [18] algorithms work for the weighted case as well, albeit with an extra factor of $\operatorname{polylog} W$, in $\widetilde{O}(n^2 \log W \cdot \mathcal{T}_{\mathsf{ind}} + n^3 \operatorname{polylog} W)$ and $\widetilde{O}(nr \log W \cdot \mathcal{T}_{\mathsf{rank}} + n^3 \operatorname{polylog} W)$ times. Huang, Kakimura, and Kamiyama [16] obtained a generic framework that transforms any algorithm that solves the unweighted case into one that solves the weighted case with an extra $O(W)$ factor. Plugging in the state-of-the-art algorithms of [4] and [2], we get $\widetilde{O}(n\sqrt{r} \cdot W \cdot \mathcal{T}_{\mathsf{rank}})$ and $\widetilde{O}(nr^{3/4} \cdot W \cdot \mathcal{T}_{\mathsf{ind}})$ algorithms. Chekuri and Quanrud [6] also gave an $\widetilde{O}(n^2/\epsilon^2 \cdot \mathcal{T}_{\mathsf{ind}})$ approximation algorithm which, according to [3], can be improved to subquadratic by applying more recent techniques. A similar $\widetilde{O}(nr^{3/2}/\epsilon \cdot \mathcal{T}_{\mathsf{ind}})$ approximation algorithm was obtained independently by Huang et al. [16].

Our Result. The question of whether weighted matroid intersection can be solved exactly in subquadratic time with polylogarithmic dependence on W under either oracle model remained open. We obtain the first subquadratic algorithm for exact weighted matroid intersection under rank oracles. The formal statement of Theorem 1 is presented as Theorem 5 in Section 2.

▶ **Theorem 1.** *Weighted matroid intersection can be solved in* $\widetilde{O}(nr^{3/4} \log W \cdot \mathcal{T}_{\mathsf{rank}})$ *time.*

Our algorithm relies on the framework of Fujishige-Zhang [14] and Shigeno-Iwata [23], where they first obtain an approximate solution by adjusting weights of some elements (similar to the "auction" algorithms for bipartite matching [20]) and then refine it by augmenting the solution iteratively.

[1] For function $f(n)$, $\widetilde{O}(f(n))$ denotes $O(f(n) \operatorname{polylog} f(n))$.

We obtain efficient algorithms for these two phases, leading to the final subquadratic algorithm.

2 Preliminaries

Notation. For a set S, let $|S|$ denote the cardinality and 2^S the power set of S. Let $S \setminus R$ consist of elements of S which are not in R. Let $e = (u, v, w)$ denote a weighted directed edge directing from u to v with weight $w = w_E(e)$ and (u, v) be its unweighted counterpart.[2] Let $\mathrm{head}(e) = v$ and $\mathrm{tail}(e) = u$. For an edge set E, let $\mathrm{head}(E) = \{\mathrm{head}(e) \mid e \in E\}$ and $\mathrm{tail}(E) = \{\mathrm{tail}(e) \mid e \in E\}$. For functions f, g mapping from a set V to $\mathbb{R}$, let $f + g$, $f - g$, and $f + c$ for $c \in \mathbb{R}$ denote functions from V to $\mathbb{R}$ with $(f + g)(x) = f(x) + g(x)$, $(f - g)(x) = f(x) - g(x)$, and $(f + c)(x) = f(x) + c$ for each $x \in V$. We often abuse notation and use f to denote the function from 2^V to $\mathbb{R}$ with $f(S) = \sum_{x \in S} f(x)$ for each $S \subseteq V$.

Matroid. Let V be a finite set and $w : V \to \mathbb{Z}$ be a given weight function. For $S \subseteq V$, let $\overline{S} = V \setminus S$. Let $n = |V|$ and $W = \max_{x \in V} |w(x)|$. An ordered pair $\mathcal{M} = (V, \mathcal{I})$ with *ground set* V and a non-empty family $\emptyset \in \mathcal{I} \subseteq 2^V$ is a *matroid* if

M1. for each $S \in \mathcal{I}$ and $R \subseteq S$, it holds that $R \in \mathcal{I}$, and

M2. for each $R, S \in \mathcal{I}$ with $|R| < |S|$, there exists an $x \in S \setminus R$ such that $R \cup \{x\} \in \mathcal{I}$.

Sets in $\mathcal{I}$ are *independent*; sets not in $\mathcal{I}$ are *dependent*. A *basis* is a maximal independent set. A *circuit* is a minimal dependent set. It is well-known from the definition of matroid that all bases are of the same cardinality. For an independent set S and $x \notin S$, $S \cup \{x\}$ contains at most one circuit C and if it does, then $x \in C$ (see [21, Lemma 1.3.3]). The *rank* of $S \subseteq V$, denoted by $\mathsf{rank}(S)$, is the size of the largest $S' \subseteq S$ such that $S' \in \mathcal{I}$. The rank of $\mathcal{M}$ is the rank of V, i.e., the size of the bases of $\mathcal{M}$. Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set, the *weighted matroid intersection* problem is to find an $S^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing $w(S^*)$. Let $r = \max_{S \in \mathcal{I}_1 \cap \mathcal{I}_2} |S|$. In this paper, the two matroids are accessed through *rank oracles*, one for each matroid. Specifically, let $\mathsf{rank}_1(\cdot)$ and $\mathsf{rank}_2(\cdot)$ denote the rank functions of $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. We assume that given pointers to a linked list containing elements of S (see, e.g., [5]), the rank oracles compute $\mathsf{rank}_1(S)$ and $\mathsf{rank}_2(S)$ in $O(\mathcal{T}_{\mathsf{rank}})$ time. With the $O(n\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ unweighted matroid intersection algorithm of Chakrabarty et al. [4], we also assume that $\mathcal{M}_1$ and $\mathcal{M}_2$ are of the same rank and share a common basis $S^{(0)}$ of size r by adjusting the given rank oracles properly.[3] By adding r zero-weight elements to V, we may also assume that each common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ is contained in a common basis of the same weight.[4] Therefore, it suffices to find a common basis S^* maximizing $w(S^*)$. Note that elements with negative weights can be safely discarded from V.

[2] We use $w_E(\cdot)$ to denote edge weights so that they can be unambiguously distinguished from the element weights $w(\cdot)$ introduced later.

[3] We can compute r via the unweighted matroid intersection algorithm and regard all sets of size greater than r as dependent.

[4] In particular, let $Z = \{z_1, \ldots, z_r\}$ be the set of newly added zero-weight elements. For each $i \in \{1, 2\}$, instead of working with $\mathcal{M}_i$, we now work with $\widetilde{\mathcal{M}}_i = (V \cup Z, \widetilde{\mathcal{I}}_i)$ such that for each $\widetilde{S} \subseteq V \cup Z$, $\widetilde{S} \in \widetilde{\mathcal{I}}_i$ if and only if $|\widetilde{S}| \leq r$ and $\widetilde{S} \setminus Z \in \mathcal{I}_i$. This change is reflected in the new rank function $\widetilde{\mathsf{rank}}_i(\widetilde{S}) = \min(\mathsf{rank}_i(\widetilde{S} \setminus Z) + |\widetilde{S} \cap Z|, r)$, which can be implemented via the given oracle rank_i.

Weight-Splitting. For weight function $f : V \to \mathbb{R}$, a basis S of matroid $\mathcal{M}$ is *f-maximum* if $f(S) \geq f(R)$ holds for each basis R of $\mathcal{M}$. Let $w^\epsilon = (w_1^\epsilon, w_2^\epsilon)$ with $w_i^\epsilon : V \to \mathbb{R}$ being a weight function for each $i \in \{1, 2\}$. Let $w^\epsilon(x) = w_1^\epsilon(x) + w_2^\epsilon(x)$. We say that w^ϵ is an *ϵ-splitting* (see, e.g., [23]) of w with $\epsilon > 0$ if $w(x) \leq w^\epsilon(x) \leq w(x) + \epsilon$ holds for each $x \in V$. If w^ϵ is an ϵ-splitting of w and S_i is a w_i^ϵ-maximum basis of $\mathcal{M}_i$ for each $i \in \{1, 2\}$, then we call (w^ϵ, S) with $S = (S_1, S_2)$ an *ϵ-partial-solution* of w. Note that by M1, $S_1 \cap S_2$ is a common independent set. If $S_1 = S_2$, then (w^ϵ, S) is an *ϵ-solution* of w. In this case, we may abuse notation and refer to S_1 as simply S.

Matroid Algorithms. The unweighted version of the following lemma was shown in [4] (it was also mentioned in [19]), and it was extended to the weighted case implicitly in [3].

▶ **Lemma 2** ([4, Lemma 13], [19], and [3]). *For $i \in \{1, 2\}$, given $S \in \mathcal{I}_i$, $B \subseteq S$ (respectively, $B \subseteq \overline{S}$), $x \in \overline{S}$ (respectively, $x \in S$), and weight function $f : V \to \mathbb{R}$, it takes $O(\log |B| \cdot \mathcal{T}_{\mathsf{rank}})$ time to either obtain a $b \in B$ minimizing/maximizing $f(b)$ such that $(S \setminus \{b\}) \cup \{x\} \in \mathcal{I}_i$ (respectively, $(S \setminus \{x\}) \cup \{b\} \in \mathcal{I}_i$) or report that such an element does not exist in B.*

The main idea of Lemma 2 is to perform binary search on B ordered by $f(\cdot)$. Throughout this paper, we will maintain such an ordered set in a balanced binary search tree where each element holds pointers to its successor and predecessor and each node holds pointers to the first and the last elements in its corresponding subtree. This allows us to perform binary search on the tree and obtain pointers to the linked list containing elements in a consecutive range efficiently.

The following greedy algorithm for finding a maximum-weight basis is folklore.

▶ **Lemma 3** (See, e.g., [9]). *It takes $O(n \log n + n\mathcal{T}_{\mathsf{rank}})$ time to obtain a f-maximum basis S of a given matroid $\mathcal{M}$ and weight function $f : V \to \mathbb{R}$.*

2.1 The Framework

The core of our algorithm is the following subroutine.

▶ **Theorem 4.** *Given a 2ϵ-solution $(w^{2\epsilon}, S')$ of w, it takes $O(nr^{3/4} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time to obtain an ϵ-solution (w^ϵ, S).*

With Theorem 4, the weighted matroid intersection algorithm follows from the standard weight-scaling framework (see, e.g, [14, 23]). Recall that our goal is to find a maximum-weight common basis.

▶ **Theorem 5** (Weighted Matroid Intersection). *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$, it takes $O(nr^{3/4} \log n \log (rW) \cdot \mathcal{T}_{\mathsf{rank}})$ time to obtain an $S^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing $w(S^*)$.*

Proof. Let $w^W = (w_1^W, w_2^W)$ with $w_i^W(x) = \frac{W}{2}$ for each $x \in V$ and the initial common basis $S^{(0)}$ obtained via the unweighted matroid intersection algorithm be a W-solution of w. Repeatedly apply Theorem 4 for $O(\log rW)$ iterations to obtain a $\frac{1}{2r}$-solution $(w^{\frac{1}{2r}}, S^*)$. For each $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, we have

$$w(S) \leq w^{\frac{1}{2r}}(S) \leq w^{\frac{1}{2r}}(S^*) \leq w(S^*) + r \cdot \frac{1}{2r} < w(S^*) + 1.$$

Since $w(S)$ and $w(S^*)$ are integers, S^* is a maximum-weight common basis. The algorithm runs in $O(nr^{3/4} \log n \log (rW) \cdot \mathcal{T}_{\mathsf{rank}})$ time. The theorem is proved. ◀

The rest of the paper proves Theorem 4.

3 The Algorithm

As in [14] and [23], the algorithm of Theorem 4 consists of the following two parts.

3.1 Weight Adjustment

The first part of the algorithm is the following subroutine which computes two bases S_1 and S_2 with a large enough intersection. This part is essentially the same as Shigeno and Iwata's algorithm [23], except that we replace the fundamental (co-)circuit queries in it with calls to Lemma 2.

▶ **Lemma 6.** *Given a 2ϵ-solution $(w^{2\epsilon}, S')$ and a parameter $1 \le k \le r$, it takes $O(nk \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time to obtain an ϵ-partial-solution (w^ϵ, S) with $|S_1 \cap S_2| \ge \left(1 - \frac{O(1)}{k}\right) r$.*

Since the algorithm and analysis are essentially the same as in [23], here we only describe how we can obtain S_1 and S_2 in the desired time bound. Please refer to [23] or Lemma 11 in Appendix A for the proof of $|S_1 \cap S_2| \ge \left(1 - \frac{O(1)}{k}\right) r$.

Algorithm of Lemma 6. Let $w^\epsilon = (w_1^{2\epsilon}, w - w_1^{2\epsilon} + \epsilon)$ be the initial ϵ-splitting and S_i be the w_i^ϵ-maximum basis of $\mathcal{M}_i$ obtained by Lemma 3 in $O(n \log n + n\mathcal{T}_{\mathsf{rank}})$ time for each $i \in \{1, 2\}$. Let $p(x) = 0$ for each $x \in V$. Repeat the following *weight adjustment* for an arbitrary $x \in S_1 \setminus S_2$ with $p(x) < k$ until such an x becomes non-existent.

- If $w^\epsilon(x) = w(x) + \epsilon$, then set $w_1^\epsilon(x) \leftarrow w_1^\epsilon(x) - \epsilon$. Apply Lemma 2 to obtain a $y \in V \setminus S_1$ maximizing $w_1^\epsilon(y)$ such that $(S_1 \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1$. If $w_1^\epsilon(x) < w_1^\epsilon(y)$, then set $S_1 \leftarrow (S_1 \setminus \{x\}) \cup \{y\}$.
- Otherwise, set $p(x) \leftarrow p(x) + 1$ and $w_2^\epsilon(x) \leftarrow w_2^\epsilon(x) + \epsilon$. Apply Lemma 2 to obtain a $y \in S_2$ minimizing $w_2^\epsilon(y)$ such that $(S_2 \setminus \{y\}) \cup \{x\} \in \mathcal{I}_2$. If $w_2^\epsilon(x) > w_2^\epsilon(y)$, then set $S_2 \leftarrow (S_2 \setminus \{y\}) \cup \{x\}$.

Since $p(x)$ is only incremented when $x \in S_1 \setminus S_2$, we have $p(x) \le k$ for each $x \in V$ when the procedure terminates. Apparently, $w^\epsilon(x)$ oscillates between $w(x)$ and $w(x) + \epsilon$, and thus the number of weight adjustments for x is bounded by $2p(x)$. We also have that S_i remains w_i^ϵ-maximum for each $i \in \{1, 2\}$ due to the potential exchange of x and y after the adjustment. Each weight adjustment takes $O(\mathcal{T}_{\mathsf{rank}} \log n)$ time by Lemma 2, hence the total running time is $O(nk \log n \cdot \mathcal{T}_{\mathsf{rank}})$.

3.2 Augmentation

With S_1 and S_2 obtained from Lemma 6, we then run "few" augmentations to make these two bases equal. To do so, we need the following notion of exchange graphs, which is slightly different compared to previous algorithms for unweighted matroid intersection (e.g., [3, 4, 7, 17]).

Exchange Graph. Let (w^ϵ, S) be an ϵ-partial-solution of w with $S_1 \ne S_2$. The *exchange graph* with respect to (w^ϵ, S) is a weighted directed multi-graph $G_{w^\epsilon, S} = (V \cup \{s, t\}, E)$ with $s, t \notin V$ and $E = E_1 \cup E_2 \cup E_s \cup E_t$, where

$$E_1 = \{(x, y, w_1^\epsilon(x) - w_1^\epsilon(y)) \mid x \in S_1, y \notin S_1, \text{ and } (S_1 \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1\},$$
$$E_2 = \{(y, x, w_2^\epsilon(x) - w_2^\epsilon(y)) \mid x \in S_2, y \notin S_2, \text{ and } (S_2 \setminus \{x\}) \cup \{y\} \in \mathcal{I}_2\},$$
$$E_s = \{(s, x, 0) \mid x \in S_1 \setminus S_2\}, \text{ and}$$
$$E_t = \{(x, t, 0) \mid x \in S_2 \setminus S_1\}.$$

Since S_i is w_i^ϵ-maximum for each $i \in \{1, 2\}$, all edge weights are non-negative. Note that this definition of exchange graph is a simplified version of the *auxiliary graph* defined by Fujishige and Zhang [14] to solve the more generalized *independent assignment* problem[5]. We have the following properties of the exchange graph, for which we also provide simplified and more direct proofs for self-containedness in Appendix A.

▶ **Lemma 7** ([14]; See Appendix A). *$G_{w^\epsilon,S}$ admits an st-path.*

Let $d(x)$ be the sx-distance in $G_{w^\epsilon,S}$ for each $x \in V$ (set $d(x)$ to a large number if x is unreachable from s; see Section 3.3 for the exact value) and P be the shortest st-path with the least number of edges.

Let $\widehat{S}_1 = (S_1 \setminus \mathrm{tail}(P \cap E_1)) \cup \mathrm{head}(P \cap E_1)$ and $\widehat{S}_2 = (S_2 \setminus \mathrm{head}(P \cap E_2)) \cup \mathrm{tail}(P \cap E_2)$ be S_1 and S_2 augmented by P. Let $\widehat{w}_1^\epsilon(x) = w_1^\epsilon(x) + d(x)$ and $\widehat{w}_2^\epsilon(x) = w_2^\epsilon(x) - d(x)$. We have that $(\widehat{w}^\epsilon, \widehat{S})$ with $\widehat{w}^\epsilon = (\widehat{w}_1^\epsilon, \widehat{w}_2^\epsilon)$ and $\widehat{S} = (\widehat{S}_1, \widehat{S}_2)$ is a better ϵ-solution (note that $\widehat{w}^\epsilon$ is indeed an ϵ-splitting). In other words, we have the following.

▶ **Lemma 8** ([14]; See Appendix A). *It holds that $(\widehat{w}^\epsilon, \widehat{S})$ is an ϵ-solution with $|\widehat{S}_1 \cap \widehat{S}_2| > |S_1 \cap S_2|$.*

With the above properties and Lemma 6, we finish our algorithm with the following shortest-path procedure. Note that in order to make the algorithm subquadratic, we do not construct the exchange graphs explicitly. Nevertheless, we show that a partial construction suffices to compute the shortest-path trees in them.

▶ **Lemma 9.** *It takes $O(n\sqrt{r}\log n \cdot \mathcal{T}_{\mathsf{rank}})$ time to obtain $d(x)$ for each $x \in V$ and the shortest st-path with the least number of edges in $G_{w^\epsilon,S}$.*

We are now ready to prove Theorem 4.

Proof of Theorem 4. Apply Lemma 6 with $k = r^{3/4}$ to obtain an ϵ-partial-solution (w^ϵ, S) of w such that $|S_1 \cap S_2| \geq r - O(r^{1/4})$ in $O(nr^{3/4}\log n \cdot \mathcal{T}_{\mathsf{rank}})$ time. For $O(r^{1/4})$ iterations, apply Lemmas 8 and 9 to obtain $(\widehat{w}^\epsilon, \widehat{S})$ with $\widehat{S}_1$ and $\widehat{S}_2$ having a larger intersection than S_1 and S_2 do, and set $(w^\epsilon, S) \leftarrow (\widehat{w}^\epsilon, \widehat{S})$ until $S_1 = S_2$. This takes overall $O(nr^{3/4}\log n \cdot \mathcal{T}_{\mathsf{rank}})$ time as well. Note that $w^\epsilon(x) = w_1^\epsilon(x) + w_2^\epsilon(x)$ remains the same, completing the proof. ◀

The remainder of this section proves Lemma 9. For the ease of notation, we abbreviate $G_{w^\epsilon,S}$ as G.

Intuitively, we would like to run Dijkstra's algorithm [8] on G to build a shortest-path tree. However, naïve implementation takes $O(nr)$ time since we might need to relax $O(nr)$ edges. This is unlike the BFS algorithm of Chakrabarty et al. [4] for the unweighted case, where we can immediately mark all out-neighbors of the current vertex as "visited", leading to a near-linear running time. To speed things up, note that using Lemma 2, for a vertex x, we can efficiently find the vertex which is "closest" to x. Let F denote the set of visited vertices whose exact distances are known. The closest unvisited vertex to F must be closest to some $x \in F$. Therefore, in each iteration, it suffices to only relax the "shortest" edge from

[5] Specifically, given a bipartite graph $G = (V_1 \cup V_2, E)$ with V_1 and V_2 being copies of V and two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$, $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on V, the independent assignment problem aims to find the largest $S_1 \in \mathcal{I}_1$ and $S_2 \in \mathcal{I}_2$ such that G admits a perfect matching between $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$. Analogously, the weighted version of the problem wants to find S_1 and S_2 such that the weight of the maximum-weight perfect matching between S_1 and S_2 is maximized. Clearly, the (weighted) matroid intersection problem is a special case of the (weighted) independence assignment problem with $E = \{(v, v) \mid v \in V\}$.

each $x \in F$. This can be done efficiently by maintaining a set of "recently visited" vertices B of size roughly $\sqrt{n}$ and computing the distance estimate from $F \setminus B$ to all unvisited vertices[6]. In each iteration, we relax the shortest edge from each $x \in B$, and now the vertex with the smallest distance estimate is closest to F and therefore we include it into B (and thus F). When B grows too large, we clear B and recompute the distance estimates from F in $\widetilde{O}(n)$ queries. This leads to a subquadratic algorithm. We now prove the lemma formally.

Proof of Lemma 9. The algorithm builds a shortest-path tree of G using Dijkstra's algorithm. We maintain a distance estimate $\widehat{d}(x)$ for each $x \in V \cup \{s,t\}$. Initially, $\widehat{d}(x) = 0$ for each $x \in (S_1 \setminus S_2) \cup \{s\}$ and $\widehat{d}(x) = \infty$ for other vertices. Edge set E_t is only for the convenience of defining an st-path and thus we may ignore it here. Let F be the set of *visited* vertices whose distance estimates are correct, i.e., $d(x) = \widehat{d}(x)$ holds for each $x \in F$. Initially, $F = \{s\}$. The algorithm runs in at most n iterations, and in the t-th iteration, we visit a new vertex v_t such that $d(v_t) = \widehat{d}(v_t)$ and $d(v_t) \leq d(v)$ for each $v \notin F$. We maintain two *buffers* $B_1 \subseteq F \cap S_1$ and $B_2 \subseteq F \cap \overline{S_2}$ containing vertices in S_1 and $\overline{S_2}$ that are visited "recently". That is, after the t-th iteration, we have $B_1 = \{v_i, \dots, v_t\} \cap S_1$ or $B_1 = \emptyset$ and $B_2 = \{v_j, \dots, v_t\} \cap \overline{S_2}$ or $B_2 = \emptyset$ for some $i, j \leq t$. Recall that E_1 and E_2 are the edges in G that correspond to exchange relations in $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. For each $i \in \{1,2\}$ and edge $e = (x,y)$, let $w_{E_i}(x,y) = |w_i^\epsilon(x) - w_i^\epsilon(y)|$ be the edge weight of e in E_i if $e \in E_i$ and $w_{E_i}(x,y) = \infty$ otherwise. Let $E(B_i) = \{(x,y) \in E_i \mid x \in B_i \text{ and } y \notin F\}$ be edges in E_i directing from B_i to $V \setminus F$ and $E(B) = E(B_1) \cup E(B_2)$. Let $E(F) = \{(x,y) \in E_1 \cup E_2 \mid x \in F \text{ and } y \notin F\}$. For $v \in V \setminus F$ and edge set E' such that $\mathrm{tail}(E') \subseteq F$, let $\widetilde{d}(v, E') = \min_{(x,v) \in E'} \{d(x) + w_E(x,v)\}$ be the shortest distance to v "relaxed" by edges in E' (recall that $w_E(x,v)$ is the weight of the edge (x,v)). We maintain the following invariants after each iteration of the algorithm except the last one.

- **(i)** $d(v) \leq \widehat{d}(v) \leq \widetilde{d}(v, E(F) \setminus E(B))$ holds for each $v \in V \setminus F$.
- **(ii)** There exists a $v \in V \setminus F$ such that $\widehat{d}(v) = d_F^* := \min_{u \in V \setminus F} \{\widetilde{d}(u, E(F))\}$.

Intuitively, Invariant (i) asserts that all edges in $E(F) \setminus E(B)$ are "relaxed" while Invariant (ii) ensures that the distance estimate of the target vertex, i.e., one with the shortest distance from s, is correct. Initially, both invariants are satisfied since $\widehat{d}(x) = 0$ holds for each $x \in S_1 \setminus S_2$. We maintain a priority queue Q containing vertices in $V \setminus F$ ordered by $\widehat{d}(\cdot)$. In the t-th iteration, let v_t be the vertex v with the smallest $\widehat{d}(v)$. By Invariants (i) and (ii), we have $\widehat{d}(v_t) = d_F^*$ and thus $d(v_t) \leq d(v')$ holds for each $v' \in V \setminus F$ according to Dijkstra's algorithm. As such, we push v_t into F and update B_1, B_2 appropriately by checking if v_t belongs to S_1 and $\overline{S_2}$. Now, we would like to modify $\widehat{d}(v)$ for some $v \in V \setminus F$ so that both invariants remain true. For each $i \in \{1,2\}$, depending on the size of B_i, we perform one of the following.

1. If $|B_i| \geq \sqrt{r}$, then we compute $\widetilde{d}_i(v) = \widetilde{d}(v, E(B_1))$ and set $\widehat{d}(v) \leftarrow \min(\widehat{d}(v), \widetilde{d}_i(v))$ for each $v \in V \setminus F$ using Lemma 10 below. For $i = 1$, by definition of G, $\mathrm{head}(E_1) \subseteq V \setminus S_1$ and thus we only need to compute $\widetilde{d}_i(v)$ for $v \in V \setminus S_1$, and therefore Lemma 10 takes $O(n \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time. For $i = 2$, similarly, $\mathrm{head}(E_2) \subseteq S_2$ and thus we only need to compute $\widetilde{d}_i(v)$ for $v \in S_2$, taking $O(r \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time. Then, we set $B_i \leftarrow \emptyset$, and the above modification ensures that Invariant (i) holds since $\widetilde{d}(v, E(F)) = \min(\widetilde{d}(v, E(F) \setminus E(B)), \widetilde{d}(v, E(B)))$.

[6] In the actual algorithm, we maintain two buffers instead of one to further improve the running time to $\widetilde{O}(n\sqrt{r})$ from $\widetilde{O}(n\sqrt{n})$. This makes our weighted matroid intersection algorithm $o(nr)$ as opposed to just $o(n^2)$.

2. If $|B_i| < \sqrt{r}$, then we do not clear B_i, and therefore Invariant (i) trivially holds. For each $b \in B_i$, we find a $v_b \in V \setminus F$ minimizing $d(b) + w_{E_i}(b, v_b)$ via Lemma 2 as follows. If $i = 1$, then we have $w_{E_1}(b, v_b) = w_1^\epsilon(b) - w_1^\epsilon(v_b)$, and thus we find the v_b maximizing $w_1^\epsilon(v_b)$ such that $(S_1 \setminus \{b\}) \cup \{v_b\} \in \mathcal{I}_1$. If $i = 2$, then $w_{E_2}(b, v_b) = w_2^\epsilon(v_b) - w_2^\epsilon(b)$, and thus we find the v_b minimizing $w_2^\epsilon(v_b)$ such that $(S_2 \setminus \{v_b\}) \cup \{b\} \in \mathcal{I}_2$. Then, we set $\widehat{d}(v_b) \leftarrow \min(\widehat{d}(v_b), d(b) + w_{E_i}(b, v_b))$ and update v_b's position in Q appropriately. This takes $O(\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time.

In both cases, as argued above, Invariant (i) holds. We argue that Invariant (ii) holds after the iteration as well. Let $B_1^{(t)}$ be B_1 after the t-th iteration and define $B_2^{(t)}$ and $F^{(t)}$ similarly. Let $E(B^{(t)})$ denote $E(B_1^{(t)}) \cup E(B_2^{(t)})$. Let $v^* = \arg\min_{v \in V \setminus F^{(t)}} \{\widetilde{d}(v, E(F^{(t)}))\}$ be an unvisited vertex after the t-th iteration with the smallest distance from s and let $e^* = (u, v^*)$ be the edge such that $u \in F^{(t)}$ and $d(v^*) = d(u) + w(e^*)$. That is, e^* is the edge connecting v^* and its parent in the shortest-path tree. If $e^* \in E(F^{(t-1)}) \setminus E(B^{(t-1)})$, then Invariant (ii) trivially follows from the end of the $(t-1)$-th iteration. Otherwise, we must have either $e^* \in E(B_1^{(t-1)})$ or $e^* \in E(B_2^{(t-1)})$. Without loss of generality, let's assume $e^* \in E(B_1^{(t-1)})$. If $|B_1^{(t-1)}| + 1 \geq \sqrt{r}$ (i.e., Case 1), then after setting $B_1^{(t)} \leftarrow \emptyset$, Invariant (ii) follows from the fact the Invariant (i) holds for v^* and $\widetilde{d}(v^*, E(F^{(t)}) \setminus E(B^{(t)})) \leq d(u) + w_E(e^*)$ since $e^* \in E(F^{(t)}) \setminus E(B^{(t)})$. If $|B_1^{(t-1)}| + 1 < \sqrt{r}$ (i.e., Case 2), then there must exists a $b \in B_1^{(t)}$ such that $d(b) + w_{E_1}(b, v^*) = \min_v \{d(b) + w_{E_1}(b, v)\}$ and thus we have at least one $v_b \in V \setminus F^{(t)}$ such that $\widehat{d}(v_b) \leq d(b) + w_{E_1}(b, v_b) = d(v^*)$. This shows that Invariant (ii) indeed holds after the t-th iteration. The correctness of the algorithm follows from the two invariants and the analysis of Dijkstra's algorithm.

To bound the total running time, observe that for B_1, Case 1 happens at most $O(r/\sqrt{r}) = O(\sqrt{r})$ times since $|S_1| = r$. Thus, it takes $O(n\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time in total. Similarly, for B_2, Case 1 happens at most $O(n/\sqrt{r})$ time, taking $O(n/\sqrt{r} \cdot r \log n \cdot \mathcal{T}_{\mathsf{rank}}) = O(n\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time in total as well. For Case 2, each iteration takes $O(\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time, contributing a total of $O(n\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time. As a result, the algorithm runs in $O(n\sqrt{r} \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time, as claimed.

Finally, it is easy to maintain balanced binary search trees of elements ordered by w_1^ϵ, w_2^ϵ, $\widehat{d} + w_1^\epsilon$, and $\widehat{d} - w_2^\epsilon$ in $O(n \log n)$ time throughout the procedure so that Lemma 2 can be applied without overhead. The shortest st-path can also be easily recovered by maintaining the optimal parent in the shortest-path tree for each vertex. This proves the lemma. ◀

▶ **Lemma 10.** *For each $i \in \{1, 2\}$, given $B_i \subseteq F$ and $R \subseteq V \setminus F$, it takes takes $O(|R| \log n \cdot \mathcal{T}_{\mathsf{rank}})$ time to compute $\widetilde{d}(v, E(B_i))$ for all $v \in R$.*

Proof. For $i = 1$ and $e = (b, v) \in E(B_i)$, we have $w_E(e) = w_1^\epsilon(b) - w_1^\epsilon(v)$. Therefore, $d(v, E(B_1))$ can be computed by finding the $b \in B$ with the smallest $d(b) + w_1^\epsilon(b)$ such that $(S_1 \setminus \{b\}) \cup \{v\} \in \mathcal{I}_1$ via Lemma 2. Similarly, for $i = 2$, we have $w_E(e) = w_2^\epsilon(v) - w_2^\epsilon(b)$, and thus $d(v, E(B_2))$ can be computed by finding the $b \in B$ with the smallest $d(b) - w_2^\epsilon(b)$. The lemma simply follows by calling Lemma 2 once for each $v \in R$. ◀

3.3 Bounding the Numbers

Finally, to conclude the analysis of our algorithm, we argue that the numbers such as $w_1^\epsilon(x)$ and $w_2^\epsilon(x)$ are bounded by $\widetilde{O}(\mathrm{poly}(nW))$ so that the number of bits needed to store them and the time for a single arithmetic operation only grow by a constant factor. In the weight adjustment stage, each number is adjusted at most $O(r)$ times and each adjustment changes the number by at most $O(W)$ since ϵ is at most W. Therefore, the accumulative change

to a number via weight adjustments is at most $O(\text{poly}(nW))$. For growth incurred by augmentations, we first assume that all vertices are reachable from s in $G_{w^\epsilon,S}$. Consider a single run of Lemma 9 and fix an $x \in V$. Let $P_x = \{s, v_1, \ldots, v_k\}$ with $v_k = x$ be the shortest sx-path in $G_{w^\epsilon,S}$. Suppose that $(v_1, v_2), (v_{k-1}, v_k) \in E_1$, then by definition, we have

$$\begin{aligned} d(x) &= w_1^\epsilon(v_1) - w_1^\epsilon(v_2) + w_2^\epsilon(v_3) - w_2^\epsilon(v_2) + \cdots + w_1^\epsilon(v_{k-1}) - w_1^\epsilon(v_k) \\ &\leq w_1^\epsilon(v_1) - w(v_2) + (w(v_3) + \epsilon) + \cdots + (w(v_{k-1}) + \epsilon) - w_1^\epsilon(v_k) \\ &\leq w_1^\epsilon(v_1) - w_1^\epsilon(v_k) + \left(\sum_{i=2}^{k-1} (-1)^{i+1} w(v_i)\right) + nW. \end{aligned}$$

Since $\widehat{w}_1^\epsilon(x) = w_1^\epsilon(x) + d(x)$ and $\widehat{w}_2^\epsilon(x) = w_2^\epsilon(x) - d(x)$ as defined in Lemma 8, we have

$$|\widehat{w^\epsilon}_1(x)| \leq |w_1^\epsilon(v_1)| + 2nW \quad \text{and} \quad |\widehat{w}_2^\epsilon(x)| \leq |w_1^\epsilon(v_1)| + 2nW. \tag{1}$$

Similarly, if $(v_{k-1}, v_k) \in E_2$, then

$$\begin{aligned} d(x) &= w_1^\epsilon(v_1) - w_1^\epsilon(v_2) + w_2^\epsilon(v_3) - w_2^\epsilon(v_2) + \cdots + w_2^\epsilon(v_k) - w_2^\epsilon(v_{k-1}) \\ &\leq w_1^\epsilon(v_1) - w(v_2) + (w(v_3) + \epsilon) + \cdots + (w(v_{k-2}) + \epsilon) - w(v_{k-1}) + w_2^\epsilon(v_k) \\ &\leq w_1^\epsilon(v_1) + w_2^\epsilon(v_k) + \left(\sum_{i=2}^{k-1} (-1)^{i+1} w(v_i)\right) + nW, \end{aligned}$$

implying (1) as well. The case when $(v_1, v_2) \in E_2$ holds similarly, except now we have

$$|\widehat{w}_1^\epsilon(x)| \leq |w_2^\epsilon(v_1)| + 2nW \quad \text{and} \quad |\widehat{w}_2^\epsilon(x)| \leq |w_2^\epsilon(v_1)| + 2nW. \tag{2}$$

Since the number of augmentations is $\widetilde{O}(r^{1/4})$, we indeed have that $|w_1^\epsilon(x)| = |w_2^\epsilon(x)| = O(\text{poly}(nW)) = \Theta((nW)^k)$ for some constant k. For the case where some vertex x is not reachable from s, we can simply set $d(x)$ to some $c(nW)^{k+1}$ for a large enough constant c and the desired bound still holds.

4 Concluding Remarks

We present a simple subquadratic algorithm for weighted matroid intersection under the rank oracle model, providing a partial yet affirmative answer to one of the open problems raised by Blikstad et al. [3]. Whether the same is achievable under the independence oracle model remains open. It seems that our techniques for computing shortest-path trees do not solely result in a subquadratic augmenting-path algorithm under the independence oracle. Removing the dependence on $\log W$ and making the algorithm run in strongly-polynomial time is also of interest. Finally, as noted in [3], there were very few non-trivial lower bound results for matroid intersection. It would be helpful to see if there is any super-linear lower bound on the number of queries for these problems or even for computing shortest-path trees in the exchange graphs under either oracle model.

References

1 Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971. doi:10.2307/1995784.

2 Joakim Blikstad. Breaking o(nr) for matroid intersection. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, pages 31:1–31:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.31.

3 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 421–432. ACM, 2021. doi:10.1145/3406325.3451092.

4 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1146–1168. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00072.

5 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic submodular function minimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1220–1231. ACM, 2017. doi:10.1145/3055399.3055419.

6 Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 445–457. SIAM, 2016. doi:10.1137/1.9781611974331.ch33.

7 William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986. doi:10.1137/0215066.

8 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

9 Jack Edmonds. Matroids and the greedy algorithm. *Math. Program.*, 1(1):127–136, 1971. doi:10.1007/BF01584082.

10 Jack Edmonds. Matroid intersection. *Annals of Discrete Mathematics*, 4:39–49, 1979. doi:10.1016/S0167-5060(08)70817-3.

11 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization – Eureka, You Shrink!, Papers Dedicated to Jack Edmonds, 5th International Workshop*, volume 2570 of *Lecture Notes in Computer Science*, pages 11–26. Springer, 2001. doi:10.1007/3-540-36478-1_2.

12 Jack Edmonds. Matroid partition. In *50 Years of Integer Programming 1958-2008 – From the Early Years to the State-of-the-Art*, pages 199–217. Springer, 2010. doi:10.1007/978-3-540-68279-0_7.

13 András Frank. A weighted matroid intersection algorithm. *J. Algorithms*, 2(4):328–336, 1981. doi:10.1016/0196-6774(81)90032-8.

14 Satoru Fujishige and Xiaodong Zhang. An efficient cost scaling algorithm for the independent assignment problem. *Journal of the Operations Research Society of Japan*, 38(1):124–136, 1995. doi:10.15807/jorsj.38.124.

15 Harold N. Gabow and Ying Xu. Efficient theoretic and practical algorithms for linear matroid intersection problems. *J. Comput. Syst. Sci.*, 53(1):129–147, 1996. doi:10.1006/jcss.1996.0054.

16 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 430–444. SIAM, 2016. doi:10.1137/1.9781611974331.ch32.

17 Eugene L. Lawler. Matroid intersection algorithms. *Math. Program.*, 9(1):31–56, 1975. doi:10.1007/BF01681329.

18 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 1049–1065. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.68.

19 Huy L. Nguyen. A note on cunningham's algorithm for matroid intersection. *CoRR*, abs/1904.04129, 2019. arXiv:1904.04129.

20 James B. Orlin and Ravindra K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Math. Program.*, 54:41–56, 1992. doi:10.1007/BF01586040.

21 Christopher Price. Combinatorial algorithms for submodular function minimization and related problems. Master's thesis, University of Waterloo, 2015. URL: `http://hdl.handle.net/10012/9356`.

22 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

23 Maiko Shigeno and Satoru Iwata. A dual approximation approach to weighted matroid intersection. *Oper. Res. Lett.*, 18(3):153–156, 1995. `doi:10.1016/0167-6377(95)00047-X`.

A Omitted Proofs

A.1 Proofs of Lemmas in Section 3.1

For self-containedness, we include the proof that the two bases obtained in the weight adjustment phase have a large intersection by Shigeno and Iwata [23] here.

▶ **Lemma 11** ([23]). *Let S_1 and S_2 be obtained from the procedure described in Lemma 6. Then, $|S_1 \cap S_2| \geq \left(1 - \frac{O(1)}{k}\right) r$.*

Proof. Let $p(S)$ denote $\sum_{x \in S} p(x)$. Observe that an element is never moved to $S_2 \setminus S_1$ during weight adjustments, and therefore we have $p(S_2 \setminus S_1) = 0$ and $p(S_1 \setminus S_2) = p(S_1) - p(S_2)$. Recall that S' is a common basis such that $(w^{2\epsilon}, S')$ is a 2ϵ-solution. Since $p(x)$ equals the number of adjustments of $w_2^\epsilon(x)$ and each such adjustment is preceded by an adjustment of $w_1^\epsilon(x)$, we have

$$p(x) \cdot \epsilon = w_2^\epsilon(x) - (w(x) - w_1^{2\epsilon}(x)) \leq w_1^{2\epsilon}(x) - w_1^\epsilon(x)$$

for each $x \in V$. Thus,

$$\begin{aligned} p(S_1 \setminus S_2) \cdot \epsilon &= (p(S_1) - p(S_2)) \cdot \epsilon \\ &\leq \left(w_1^{2\epsilon}(S_1) - w_1^\epsilon(S_1)\right) - \left(w_2^\epsilon(S_2) - w(S_2) + w_1^{2\epsilon}(S_2)\right) \\ &\overset{\text{(a)}}{\leq} w_1^{2\epsilon}(S_1) - w_1^\epsilon(S') - w_2^\epsilon(S') + w(S_2) - w_1^{2\epsilon}(S_2) \\ &\overset{\text{(b)}}{\leq} w_1^{2\epsilon}(S_1) - w(S') + w(S_2) - w_1^{2\epsilon}(S_2), \end{aligned}$$

where (a) is because S_i is w_i^ϵ-maximum for each $i \in \{1, 2\}$ and (b) is because $w(S') \leq w^\epsilon(S')$ as w^ϵ is an ϵ-splitting. Since $(w^{2\epsilon}, S')$ is a 2ϵ-solution, $w_2^{2\epsilon}(S) - 2\epsilon r \leq w(S) - w_1^{2\epsilon}(S) \leq w_2^{2\epsilon}(S)$ holds for each basis S. This combined with the fact that S' is $w_i^{2\epsilon}$-maximum for each $i \in \{1, 2\}$ implies

$$p(S_1 \setminus S_2) \cdot \epsilon \leq 2\epsilon r - w_2^{2\epsilon}(S') + w_2^{2\epsilon}(S_2) \leq 2\epsilon r \implies p(S_1 \setminus S_2) \leq 2r.$$

When the algorithm terminates, we have $p(x) = k$ for all $x \in S_1 \setminus S_2$, implying

$$p(S_1 \setminus S_2) = |S_1 \setminus S_2| \cdot k \leq 2r \implies |S_1 \setminus S_2| \leq \frac{2r}{k}.$$

As a result,

$$|S_1 \cap S_2| = r - |S_1 \setminus S_2| \geq \left(1 - \frac{O(1)}{k}\right) r. \qquad \blacktriangleleft$$

A.2 Proofs of Lemmas in Section 3.2

In this section, we prove the properties of the exchange graphs. The proofs for the more generalized auxiliary graph given by Fujishige and Zhang can be found in [14].

To prove Lemma 7, it would be more convenient to refer to the following definition of a directed bipartite graph based on exchange relationships, which is heavily used in unweighted matroid intersection algorithms. For $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, let $\widetilde{G}_S = (V \cup \{s,t\}, \widetilde{E})$ with $s,t \notin V$ denote the directed graph with $\widetilde{E} = \widetilde{E}_1 \cup \widetilde{E}_2 \cup \widetilde{E}_s \cup \widetilde{E}_t$, where

$$\widetilde{E}_1 = \{(x,y) \mid x \in S, y \notin S, \text{ and } (S \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1\},$$
$$\widetilde{E}_2 = \{(y,x) \mid x \in S, y \notin S, \text{ and } (S \setminus \{x\}) \cup \{y\} \in \mathcal{I}_2\},$$
$$\widetilde{E}_s = \{(s,x) \mid S \cup \{x\} \in \mathcal{I}_1\}, \text{ and}$$
$$\widetilde{E}_t = \{(x,t) \mid S \cup \{x\} \in \mathcal{I}_2\}.$$

▶ **Lemma 12** ([17]). *$\widetilde{G}_S$ for $|S| < r$ admits an st-path.*

We will use the existence of an st-path in $\widetilde{G}_{\widetilde{S}}$ to prove that such a path exists in $G_{w^\epsilon,S}$, for $\widetilde{S} = S_1 \cap S_2$. The following claims certify that $\widetilde{G}_{\widetilde{S}}$ and $G_{w^\epsilon,S}$ are almost the same.

▷ Claim 13. Let $\mathcal{M} = (V,\mathcal{I})$ be a matroid, $S \subseteq S' \in \mathcal{I}$, $x \in S$, and $y \notin S'$ such that $(S \setminus \{x\}) \cup \{y\} \in \mathcal{I}$ but $S \cup \{y\} \notin \mathcal{I}$, then $(S' \setminus \{x\}) \cup \{y\} \in \mathcal{I}$.

Proof. Let C be the unique circuit in $S \cup \{y\}$. Since $C \subseteq S' \cup \{y\}$ and $S' \cup \{y\}$ has only one circuit, C is the unique circuit in $S' \cup \{y\}$ as well. Moreover, $(S \setminus \{x\}) \cup \{y\} \in \mathcal{I}$ if and only if $x \in C$ and therefore $(S' \setminus \{x\}) \cup \{y\} \in \mathcal{I}$. ◁

▷ Claim 14. Let $\mathcal{M} = (V,\mathcal{I})$ be a matroid, $S \subseteq S' \in \mathcal{I}$ where S' is a basis of $\mathcal{M}$, and $x \notin S'$ such that $S \cup \{x\} \in \mathcal{I}$. Then, there exists a $y \in S' \setminus S$ such that $(S' \setminus \{y\}) \cup \{x\} \in \mathcal{I}$.

Proof. Let S'' be an arbitrary basis of $\mathcal{M}$ that contains $S \cup \{x\}$. Since $x \in S'' \setminus S'$, by the strong exchange property (see, e.g., [22, Theorem 39.6]) of bases, there exists a $y \in S' \setminus S'' \subseteq S' \setminus S$ such that $(S' \setminus \{y\}) \cup \{x\} \in \mathcal{I}$, completing the proof. ◁

We are now ready to prove Lemma 7.

Proof of Lemma 7. Let $\widetilde{P} = \{s, v_1, \ldots, v_k, t\}$ be the shortest st-path in $\widetilde{G}_{\widetilde{S}}$ for $\widetilde{S} = S_1 \cap S_2$. The existence of such a path is guaranteed by Lemma 12. We have $\widetilde{S} \cup \{v_i\} \notin \mathcal{I}_1$ and $\widetilde{S} \cup \{v_i\} \notin \mathcal{I}_2$ for each $1 < i < k$ since $\widetilde{P}$ is the shortest path. For an odd $1 \leq i < k$, we have $v_i \notin S$ and $v_{i+1} \in S$. If $v_i \notin S_2 \setminus S_1$, then by Claim 13, we have $(v_i, v_{i+1}) \in E(G_{w^\epsilon,S})$. Similarly, for an even $1 \leq i < k$, if $v_{i+1} \notin S_1 \setminus S_2$, then we have $(v_i, v_{i+1}) \in E(G_{w^\epsilon,S})$. Suppose that $v_1 \notin S_1 \setminus S_2$, then by Claim 14, we can find a $v_0 \in S_1 \setminus S_2$ such that $(S_1 \setminus \{v_0\}) \cup \{v_1\} \in \mathcal{I}_1$. Similarly, if $v_k \notin S_2 \setminus S_1$, then we can find a $v_{k+1} \in S_2 \setminus S_1$ such that $(S_2 \setminus \{v_{k+1}\}) \cup \{v_k\} \in \mathcal{I}_2$. Therefore, without loss of generality, we may assume that there exists the last vertex $v_i \in S_1 \setminus S_2$ and the first vertex $v_j \in S_2 \setminus S_1$ after v_i. Now, for each $i < k < j$, we have $v_k \notin (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$. Therefore, $(v_k, v_{k+1}) \in E(G_{w^\epsilon,S})$ holds for each $i \leq k < j$, and we obtain an st-path in $G_{w^\epsilon,S}$ as $P = \{s, v_i, \ldots, v_j, t\}$. This concludes the proof. ◀

Finally, to prove Lemma 8, we need the following results.

▶ **Lemma 15** ([21, Proposition 2.4.1]). *Given a matroid $\mathcal{M} = (V,\mathcal{I})$ and an $S \in \mathcal{I}$. Suppose that $(a_1, \ldots, a_p) \subseteq V \setminus S$ and $(b_1, \ldots, b_p) \subseteq S$ are two sequences satisfying the following conditions:*

1. $(S \setminus \{b_i\}) \cup \{a_i\} \in \mathcal{I}$ *for each* $1 \leq i \leq p$ *and*
2. $(S \setminus \{b_j\}) \cup \{a_i\} \notin \mathcal{I}$ *for each* $1 \leq j < i \leq p$.
Then, $(S \setminus \{b_1, \ldots, b_p\}) \cup \{a_1, \ldots, a_p\} \in \mathcal{I}$ *holds.*

▶ **Lemma 16** ([21, Lemma 2.4.2]). *Let* $\mathcal{M}$, S, $(a_1, \ldots, a_p)$, *and* $(b_1, \ldots, b_p)$ *be the same as in Lemma 15. Let* $S' = (S \setminus \{b_1, \ldots, b_p\}) \cup \{a_1, \ldots, a_p\}$. *For* $x \in S'$ *and* $y \in V \setminus S'$, *if* $(S' \setminus \{x\}) \cup \{y\} \in \mathcal{I}$ *but either* $y \in S$ *or* $(S \setminus \{x\}) \cup \{y\} \notin \mathcal{I}$, *then there exists* $1 \leq \ell \leq k \leq p$ *such that* $(S \setminus \{x\}) \cup \{a_k\} \in \mathcal{I}$ *and either* $b_\ell = y$ *or* $(S \setminus \{b_\ell\}) \cup \{y\} \in \mathcal{I}$.

In essence, Lemma 15 captures the validity of an augmentation while Lemma 16 models the condition in which new exchange relationships emerge in the augmented independent set. The following three claims imply Lemma 8. Recall that $P = \{s, v_1, \ldots, v_k, t\}$ is the shortest st-path with the least number of edges and $d(x)$ is the sx-distance in $G_{w^\epsilon, S}$.

▷ Claim 17. $|\widehat{S}_1 \cap \widehat{S}_2| > |S_1 \cap S_2|$ holds.

Proof. If $k = 2$, then either $v_1 \in \widehat{S}_1 \cap \widehat{S}_2$ or $v_k \in \widehat{S}_1 \cap \widehat{S}_2$ must hold, depending on whether $(v_1, v_2) \in E_1$ or $(v_1, v_2) \in E_2$, and the claim trivially holds in this case. Thus, in the following, we assume that $k > 2$. Since P is the shortest, we may assume that $v_i \in (S_1 \cap S_2) \cup (\overline{S_1} \cap \overline{S_2})$ holds for each $1 < i < k$. Also, for $1 < i < k-1$, if $v_i \in S_1 \cap S_2$, then v_{i+1} must be in $\overline{S_1} \cap \overline{S_2}$ due to the way $G_{w^\epsilon, S}$ is constructed. Similarly, if $v_i \in \overline{S_1} \cap \overline{S_2}$, then we must have $v_{i+1} \in S_1 \cap S_2$. Let $P_{\mathrm{mid}} = \{v_2, \ldots, v_{k-1}\}$, $I = S_1 \cap S_2$, and $O = \overline{S_1} \cap \overline{S_2}$. Clearly, we have

$$|\widehat{S}_1 \cap \widehat{S}_2| - |S_1 \cap S_2| = |P_{\mathrm{mid}} \cap O| - |P_{\mathrm{mid}} \cap I| + [\![v_1 \in \widehat{S}_1 \cap \widehat{S}_2]\!] + [\![v_k \in \widehat{S}_1 \cap \widehat{S}_2]\!]. \tag{3}$$

We prove the claim by considering the following four possible cases.

k is even and $v_2 \in I$. We have $(v_1, v_2) \in E_2$, $(v_{k-1}, v_k) \in E_2$, and $|P_{\mathrm{mid}} \cap I| = |P_{\mathrm{mid}} \cap O|$. Also, $v_1 \in \mathrm{tail}(P \cap E_2)$ and therefore $v_1 \in \widehat{S}_1 \cap \widehat{S}_2$.

k is even and $v_2 \in O$. We have $(v_1, v_2) \in E_1$, $(v_{k-1}, v_k) \in E_1$, and $|P_{\mathrm{mid}} \cap I| = |P_{\mathrm{mid}} \cap O|$. Also, $v_k \in \mathrm{head}(P \cap E_1)$ and therefore $v_k \in \widehat{S}_1 \cap \widehat{S}_2$.

k is odd and $v_2 \in I$. We have $(v_1, v_2) \in E_2$, $(v_{k-1}, v_k) \in E_1$, and $|P_{\mathrm{mid}} \cap I| = |P_{\mathrm{mid}} \cap O| + 1$. Also, $v_1 \in \mathrm{tail}(P \cap E_2)$, $v_k \in \mathrm{head}(P \cap E_1)$ and therefore $v_1, v_k \in \widehat{S}_1 \cap \widehat{S}_2$.

k is odd and $v_2 \in O$. We have $(v_1, v_2) \in E_1$, $(v_{k-1}, v_k) \in E_2$, and $|P_{\mathrm{mid}} \cap I| = |P_{\mathrm{mid}} \cap O| - 1$.

In all cases, we have $|\widehat{S}_1 \cap \widehat{S}_2| > |S_1 \cap S_2|$ via Equation (3), concluding the proof. ◁

We prove the following claims for $i = 1$. The proofs for $i = 2$ follow analogously.

▷ Claim 18. $\widehat{S}_i \in \mathcal{I}_i$ holds for each $i \in \{1, 2\}$.

Proof. Let $P_1 = P \cap E_1 = \{(b_1, a_1), (b_2, a_2), \ldots (b_p, a_p)\}$, where $(S_1 \setminus \{b_i\}) \cup \{a_i\} \in \mathcal{I}_1$ holds for each $1 \leq i \leq p$. Since P is the shortest path,

$$d(b_i) + w_1^\epsilon(b_i) - w_1^\epsilon(a_i) = d(a_i) \implies d(b_i) + w_1^\epsilon(b_i) = d(a_i) + w_1^\epsilon(a_i)$$

holds for each i. Reorder P_1 so that $d(b_1) + w_1^\epsilon(b_1) \leq d(b_2) + w_1^\epsilon(b_2) \leq \cdots \leq d(b_p) + w_1^\epsilon(b_p)$. Moreover, if $d(b_i) + w_1^\epsilon(b_i) = d(b_j) + w_1^\epsilon(b_j)$ for some i, j, then (b_i, a_i) precedes (b_j, a_j) in P_1 if and only if (b_i, a_i) precedes (b_j, a_j) in P. It follows that for each $1 \leq j < i \leq p$, it holds that $(S_1 \setminus \{b_j\}) \cup \{a_i\} \notin \mathcal{I}_1$ since otherwise we would have

$$d(b_j) + w_1^\epsilon(b_j) - w_1^\epsilon(a_i) \geq d(a_i) \implies d(b_j) + w_1^\epsilon(b_j) \geq d(a_i) + w_1^\epsilon(a_i). \tag{4}$$

Because $j < i$, (4) must take equality, but this would contradict with the fact the P has the least number of edges since the edge (b_j, a_i) "jumps" over vertices $a_j, b_{j+1}, \ldots, b_i$ in P and has the same weight as the subpath $b_j, a_j, \ldots, b_i, a_i$. As such, by Lemma 15, the claim is proved. ◁

▷ Claim 19. $\widehat{S}_i$ is $\widehat{w}_i^\epsilon$-maximum for each $i \in \{1, 2\}$.

Proof. Let $P_1 = P \cap E_1 = \{(b_1, a_1), \ldots, (b_p, a_p)\}$ be ordered the same way as in the proof of Claim 18. It suffices to show that $\widehat{w}_1^\epsilon(x) \geq \widehat{w}_1^\epsilon(y)$ holds for each $x \in \widehat{S}_1$ and $y \notin \widehat{S}_1$ with $(\widehat{S}_1 \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1$. Consider the following two cases.

1. $(S_1 \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1$: Since $(x, y) \in E_1$, it follows that

$$d(x) + w_1^\epsilon(x) - w_1^\epsilon(y) \geq d(y) \implies \widehat{w}_1^\epsilon(x) = d(x) + w_1^\epsilon(x) \geq d(y) + w_1^\epsilon(y) = \widehat{w}_1^\epsilon(y).$$

2. $(S_1 \setminus \{x\}) \cup \{y\} \notin \mathcal{I}_1$: By Lemma 16, there exists $1 \leq \ell \leq k \leq p$ such that (1) $(S_1 \setminus \{x\}) \cup \{a_k\} \in \mathcal{I}_1$ and either (2.1) $b_\ell = y$ or (2.2) $(S_1 \setminus \{b_\ell\}) \cup \{y\} \in \mathcal{I}_1$. (1) implies that $\widehat{w}_1^\epsilon(x) \geq \widehat{w}_1^\epsilon(a_k)$. If (2.1) holds, then $\widehat{w}_1^\epsilon(x) \geq \widehat{w}_1^\epsilon(a_k) \geq \widehat{w}_1^\epsilon(b_\ell) = \widehat{w}_1^\epsilon(y)$. If (2.2) holds, then $\widehat{w}_1^\epsilon(x) \geq \widehat{w}_1^\epsilon(a_k) \geq \widehat{w}_1^\epsilon(b_\ell) \geq \widehat{w}_1^\epsilon(y)$.

The claim is proved. ◁

Subsequences with Gap Constraints: Complexity Bounds for Matching and Analysis Problems

Joel D. Day
Loughborough University, UK

Maria Kosche
Computer Science Department, Universität Göttingen, Germany

Florin Manea
Computer Science Department and CIDAS, Universität Göttingen, Germany

Markus L. Schmid
Humboldt-Universität zu Berlin, Germany

Abstract

We consider subsequences with gap constraints, i. e., length-k subsequences p that can be embedded into a string w such that the induced gaps (i. e., the factors of w between the positions to which p is mapped to) satisfy given gap constraints $gc = (C_1, C_2, \ldots, C_{k-1})$; we call p a gc-subsequence of w. In the case where the gap constraints gc are defined by lower and upper length bounds $C_i = (L_i^-, L_i^+) \in \mathbb{N}^2$ and/or regular languages $C_i \in \mathrm{REG}$, we prove tight (conditional on the orthogonal vectors (OV) hypothesis) complexity bounds for checking whether a given p is a gc-subsequence of a string w. We also consider the whole set of all gc-subsequences of a string, and investigate the complexity of the universality, equivalence and containment problems for these sets of gc-subsequences.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases String algorithms, subsequences with gap constraints, pattern matching, fine-grained complexity, conditional lower bounds, parameterised complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.64

Related Version *Full Version*: https://arxiv.org/abs/2206.13896

Funding The work of the three authors with German affiliation was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG). Maria Kosche's work was supported by the project with number 389613931. Florin Manea's work was supported by the project with number 466789228. Markus L. Schmid's work was supported by the project with number 416776735.

1 Introduction

For a string $v = v_1 v_2 \ldots v_n$, where each v_i is a single symbol from some alphabet Σ, any string $u = v_{i_1} v_{i_2} \ldots v_{i_k}$ with $k \leq n$ and $1 \leq i_1 \leq i_2 \leq \ldots \leq i_k \leq n$ is called a *subsequence* (or *scattered factor* or *subword*) of v (denoted by $u \preceq v$). This is formalised by the *embedding* from the positions of u to the positions of v, i. e., the increasing mapping $e : \{1, 2, \ldots, k\} \to \{1, 2, \ldots, n\}$ with $j \mapsto i_j$ (we use the notation $u \preceq_e v$ to denote that u is a subsequence of v via embedding e). For example, the string `abacbba` has among its subsequences `aaa`, `abca`, `cba`, and `ababba`. With respect to `aaa`, there exists just one embedding, namely $1 \mapsto 1$, $2 \mapsto 3$, and $3 \mapsto 7$, but there are two embeddings for `cba`.

In this paper, we are interested in *subsequences with gap constraints* that can be embedded in such a way that the *gaps* of the embedding, i. e., the factors $v_{e(i)+1} v_{e(i)+2} \cdots v_{e(i+1)-1}$ between the images of the mapping, satisfy certain properties. We begin by discussing why the concept of classical subsequences (i. e., without gap constraints) is a central one in computer science, and then we will motivate and describe in detail our approach.

The concept of subsequences is employed in many different areas of computer science: in formal languages and logics (e. g., piecewise testable languages [59, 60, 41, 42, 43], or subword order and downward closures [35, 46, 45, 66]), in combinatorics on words [55, 25, 48, 47, 57, 53, 56], for modelling concurrency [54, 58, 16], in database theory (especially *event stream processing* [5, 34, 67]). Moreover, many classical algorithmic problems are based on subsequences, e. g., longest common subsequence [8] or shortest common supersequence [52]. Note that the longest common subsequence problem, in particular, has recently regained substantial interest in the context of fine-grained complexity (see [14, 15, 1, 2]).

There are two main types of algorithmic problems for subsequences investigated in the literature. Firstly, *matching*: the problem to decide whether a string u is a subsequence of a string v, i. e., whether $u \preceq v$ (the term matching is motivated by the point of view that u is a pattern that is to be matched with the string v). Secondly, the *analysis problems* are concerned with the sets $SubSeq(k, v)$ of all length-k subsequences of a given string v. More precisely, for given string $v \in \Sigma^+$ and integer $k \in \mathbb{N}$, we want to decide whether $SubSeq(k, v) = \Sigma^k$ (*universality*), or, for an additional string v', whether $SubSeq(k, v) \subseteq SubSeq(k, v')$ (*containment*) or $SubSeq(k, v) = SubSeq(k, v')$ (*equivalence*). For classical subsequences (as defined above), the matching problem is trivial, while the analysis problems are well-investigated and relatively well-understood. For instance, the equivalence problem was introduced by Imre Simon in his PhD thesis [59], and was intensely studied in the combinatorial pattern matching community (see [36, 32, 61, 63, 19, 24] and the references therein), before being optimally solved in 2021 [33]. In this work, we consider these problems with respect to an extended setting of subsequences, which we shall explain and motivate next.

Motivation for Our Setting. In the theoretical literature, problems on subsequences are usually considered in the setting where the embeddings (as witnesses for subsequences) can be arbitrary. This means that any subsequence u of string v is witnessed by a *canonical* embedding e that greedily maps each position i of u to the leftmost occurrence of symbol u_i in the suffix $v_{e(i-1)}v_{e(i-1)+1} \dots v_n$. For example, $u = \mathtt{aba}$ can be embedded into $v = \mathtt{abacbba}$ in six different ways, but the canonical embedding maps u to the prefix $v[1..3]$. This makes it often rather simple to deal with subsequences algorithmically: matching can be decided greedily in linear time; the set of all subsequences of a string v can be represented by a *deterministic* automaton of size $O(|v||\Sigma|)$ (which means that the analysis problems can be solved in polynomial time, although much more efficient methods exist in certain cases [33]).

For practical scenarios, on the other hand, it seems reasonable to also postulate some properties with respect to the *gaps* that are induced by the embedding. For example, if we model the scheduling of several threads on a single processor by shuffling several sequences into one string, then a-priori knowledge about the scheduling strategy may tell us that the subsequences describing the single threads will not have huge gaps (any kind of fairness property of the scheduling strategy implies this). Another example is finding alignments of bio-sequences by computing longest common subsequences. While any common subsequence of two strings can be interpreted as an alignment, it is questionable if this interpretation is still useful if roughly half of the positions of the common subsequence are mapped to the beginning of the strings, while the other half is mapped to the end of the strings, with a huge gap (say thousands of symbols) in between. This situation should rather be seen as two individual alignments. In fact, in this scenario the optimisation goal of finding a *longest* common subsequence, without further constraints, even seems counterproductive, since it may favour alignments that are to a large extent disconnected and are therefore less likely

to describe relevant properties. In the context of complex event processing, it might be desirable to describe the situation that between the events of a job A only events associated to a job B appear (e. g., due to unknown side-effects this leads to a failure of job A). In this case, we are interested in embedding a string as a subsequence such that the gaps only contain symbols from a certain subset of the alphabet (i. e., the events associated to job B). So, in practice, it makes sense to reason both about the length and the actual content of gaps induced by embeddings.

The large algorithmic tool box for problems based on subsequences is not always capable of handling the practically relevant scenarios, where we are interested in subsequences that can be embedded not just in *any* way, but in *some specific way* that is reasonable for the application scenario. We therefore investigate basic problems on subsequences in the setting where the gaps of the subsequences (or rather of the embeddings) have certain constraints.

Related Work. Subsequences with various types of gap constraints are considered in different contexts. Not unexpectedly, one of the main areas in which such subsequences were investigated is combinatorial pattern matching with biological motivations, see [10] and the references therein. In [49, 50], mining such subsequences is presented as a typical data-mining problem with applications in classification and clustering algorithms. In [44], a query class for event streams is introduced, which is based on subsequences with upper and lower length bounds as gap constraints. The longest common subsequence problem has also been extended to the case where the gaps have length constraints (see, e. g., [38] and the references therein).

Coming back to [10], a rather well-researched problem that is related to our setting is that of matching *variable length gap patterns*. In this setting, a pattern, defined as the string $u_1(p_1, q_1)u_2(p_2, q_2)\dots u_{m-1}(p_{m-1}, q_{m-1})u_m$ with $u_i \in \Sigma^+$, $(p_i, q_i) \in \mathbb{N}^2$ and $0 \leq p_i \leq q_i$, matches a string w if $w = w_0u_1w_1\dots u_{m-1}w_{m-1}u_mw_m$ with $p_i \leq |w_i| \leq q_i$. For this special pattern matching problem many algorithmic results exist (see [10] and the references therein); moreover, it has also been investigated in more practical papers that provide experimental evaluations of algorithms solving it, see, e. g., [7, 17]. The above can be seen as special variants of the matching problem for subsequences with gap-length constraints.

While the works above address mostly patterns with length constraints, the area of string constraint solving (with applications in formal verification, and a strong algorithm engineering component, see [3]) addresses the problem of aligning two strings containing constants (or contiguous sequences of one or more letters) and variables (or gaps). In general (see the aforementioned survey [3] and the references therein), the variables/gaps are subject to conjunctions of pairwise string-equality, length, or regular constraints (see also the discussion in this paper's full version [20]). Moreover, the problem of checking whether factors of words are part of a given regular language were addressed in the context of sliding window algorithms [27, 28, 29, 30, 31] or in the streaming model [9, 22].

On the other hand, we are not aware of any works that are concerned with (non-trivial) gap-constrained variants of the analysis problems (i. e., universality, containment, and equivalence). Let us now formally define the setting considered in this paper.

Subsequences With Gap Constraints. Since the gaps induced by an embedding are essentially strings (or words), it seems natural to formalise gap constraints for length-k subsequences by $(k-1)$-tuples of sets of strings (i. e., languages) $gc = (C_1, \dots, C_{k-1})$, where $C_i \subseteq \Sigma^*$ for every $i \in \{1, \dots, k-1\}$; we denote $|gc| = k-1$. A length-k subsequence $u = u_1u_2\dots u_k$ of $v = v_1v_2\dots v_n$ satisfies gc (i. e., it is a gc-subsequence) if $u \preceq_e v$ for an embedding e that satisfies gc in the sense that, for every $i \in \{1, \dots, k-1\}$, $v_{e(i)+1}\dots v_{e(i+1)-1} \in C_i$. By $SubSeq(gc, v)$ we denote the set of all gc-subsequences of v. In this setting, we consider:

- the *matching problem* MATCH: decide, for given strings w, w', and gap constraints gc with $|gc| = |w| - 1$, whether w is a gc-subsequence of w' (i. e., whether $w \in SubSeq(gc, w')$);
- the *universality problem* UNI: decide, for given string w and gap constraints gc with $|gc| = k - 1$, whether $SubSeq(gc, w) = \Sigma^k$;
- the *equivalence problem* EQU (respectively, the *containment problem* CON): decide, for given strings w, w', and gap constraints gc, whether $SubSeq(gc, w) = SubSeq(gc, w')$ (respectively, $SubSeq(gc, w) \subseteq SubSeq(gc, w')$).

Our formalisation of gap constraints is as general as possible. In order to obtain meaningful results we focus on *regular constraints*, where each C_i is a regular language, represented by a finite automaton accepting it, and on *length constraints*, where each C_i has the form $\{v \in \Sigma^* \mid L^-(i) \leq |v| \leq L^+(i)\}$ with $L^-(i), L^+(i) \in \mathbb{N} \cup \{0, +\infty\}$ and is represented as the pair $(L^-(i), L^+(i))$. We also consider *conjunctions* $((L^-(i), L^+(i)), R_i)$ *of length and regular constraints*, i. e., the gap must be of length between $L^-(i)$ and $L^+(i)$ *and* from the regular language R_i (note that simply "pushing" the length constraint into the regular language R_i would increase the size of R_i's representation as automaton by a factor $L^+(i)$, which is exponential in $L^+(i)$'s binary representation). These constraints cover the existing cases in the literature.

Our Contribution. We provide a comprehensive picture of the computational complexity of both the matching and the analysis problems, proving tight upper and lower bounds for them, with a focus on the latter.

With respect to matching, we show that we can check whether u is a gc-subsequence of v in *rectangular* time $O(|v||gc|)$, where, if each C_i is the conjunction of a regular constraint and a length constraint, $|gc|$ is the number of states of the DFAs that represent the regular constraints. In the absence of regular constraints (so, for length constraints only), such rectangular upper bounds are already reported in the literature (see [38]). Moreover, the case when length constraints are absent (so, we have regular constraints only) is rather straightforward. Our algorithm dealing with the case of conjunctions of regular and length constraints requires, however, a non-trivial extension of the existing approaches. Nevertheless, our main contribution in this area is that we can also prove a conditional lower bound that essentially states that these running times of those algorithms cannot be improved unless the *orthogonal vectors hypothesis* fails. More precisely, adding length or regular constraints to subsequences changes the matching problem from a trivial problem to a problem with provably rectangular complexity. Additionally, this proves also a conditional lower bound for matching variable length gap patterns (mentioned above), for which many upper bounds, but no matching lower bound were known before. It is also worth noting that the lower bound holds for the case of a constant alphabet and constant length constraints.

With respect to the problems of universality, equivalence, and containment, we show strong intractability results for both the cases of length constraints and of regular constraints. More precisely, these problems are NP-complete even for a fixed binary alphabet and for small, constant length (or regular) constraints (note that the problems are trivial for a unary alphabet). Moreover, for any fixed constant alphabet, the problems can be solved by brute-force algorithms in exponential time $2^{O(k)}|gc|\ell$ (recall that k is the length of subsequences; ℓ is the maximum length of the input strings), and we can show that for alphabets of size at least 3, the exponent can neither be lowered to any $o(k)$ (unless the *exponential time hypothesis* fails), nor to $k(1 - \epsilon)$ for any $\epsilon \geq 1$ (unless the *strong exponential time hypothesis* fails), and these lower bounds even hold for small constant length constraints. If we parameterise by both $|\Sigma|$ and k, then the brute-force algorithm is a trivial fpt-algorithm

(i.e., an algorithm whose running time is $f(k, |\Sigma|)n^{O(1)}$, for some computable function f, so an algorithm which is *fixed parameter tractable*, or fpt-algorithm for short). However, we can exclude fpt-running times for the cases where we parameterise by only $|\Sigma|$, or by only k (based on the assumptions P $\neq$ NP and FPT $\neq$ W[1], respectively). Note that for classical subsequences all these problems can be easily solved in polynomial time, so our results emphasise the fundamentally different nature of constrained subsequences.

As a last remark, for space reasons, missing proofs and additional comments are only presented in the full version of this paper [20].

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ and $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{N}$. By $\mathcal{P}(S)$, we denote the power set of a set S.

For a finite alphabet Σ, Σ^+ denotes the set of non-empty words over Σ and $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ (where ε is the empty word). For a word $w \in \Sigma^*$, $|w|$ denotes its length (in particular, $|\,\varepsilon\,| = 0$); for every $b \in \Sigma$, $|w|_b$ denotes the number of occurrences of b in w; we set $w^1 = w$ and $w^k = ww^{k-1}$ for every $k \geq 2$. For a string $w = w_1 w_2 \ldots w_n$ with $w_i \in \Sigma$ for every $i \in [n]$, and for every $i, j \in [|w|]$ with $i \leq j$, we define $w[i..j] = w_i w_{i+1} \ldots w_j$; moreover, we use $w[i]$ as shorthand for $w[i..i]$. For any string $w \in \Sigma^*$, we define $\mathsf{alph}(w) = \{b \in \Sigma \mid |w|_b \geq 1\}$. A *factor* of a string $w \in \Sigma^*$ is a string $v \in \Sigma^*$ such that $w = uvu'$ for $u, u' \in \Sigma^*$; if $u = \varepsilon$, then v is called a *prefix* of w, and if $u' = \varepsilon$, then v is called a *suffix* of w.

By REG, we denote the class of regular languages (see [37] for more details). For the considered algorithmic problems we use as computational model the standard unit-cost RAM with logarithmic word size, with inputs over integer alphabets (see this paper's full version [20]).

Hypotheses. We now recall some basic computational problems and respective algorithmic hypotheses. We shall use these hypotheses to obtain our conditional lower bounds.

The problem CNF-Sat gets as input a Boolean formula F in conjunctive normal form as a set of clauses $F = \{c_1, c_2, \ldots, c_m\}$ over a set of variables $V = \{v_1, v_2, \ldots, v_n\}$, i. e., for every $i \in [m]$, we have $c_i \subseteq \{v_1, \neg v_1, \ldots, v_n, \neg v_n\}$. The question is whether F is satisfiable. By k-CNF-Sat, we denote the variant where $|c_i| \leq k$ for every $i \in [m]$.

The *Orthogonal Vectors problem* (OV for short) is defined as follows: Given sets A, B each containing n Boolean-vectors of dimension d, check whether there are vectors $\vec{a} \in A$ and $\vec{b} \in B$ that are orthogonal, i. e., $\vec{a}[i] \cdot \vec{b}[i] = 0$ for every $i \in [d]$.

We shall use the following algorithmic hypotheses based on CNF-Sat and OV that are common for obtaining conditional lower bounds in fine-grained complexity (see the literature mentioned below for further details). In the following, poly is any fixed polynomial function.

- *Exponential Time Hypothesis* (ETH) [40, 51]: 3-CNF-Sat cannot be solved in time $2^{o(n)}\,\mathrm{poly}(n+m)$.
- *Strong Exponential Time Hypothesis* (SETH) [39, 65]: For every $\epsilon > 0$ there exists a k such that k-CNF-Sat cannot be decided in $O(2^{n(1-\epsilon)}\,\mathrm{poly}(n))$.
- *Orthogonal Vectors Hypothesis* (OVH) [12, 13, 65]: For every $\epsilon > 0$ there is no algorithm solving OV in time $\mathrm{O}(n^{2-\epsilon}\,\mathrm{poly}(d))$.

Subsequences With Gap Constraints. We now define subsequences with gap constraints (see also the introduction). In the following, let Σ be a finite *alphabet*. Recall that for a string w, an embedding is a function $e : [k] \to [|w|]$ such that $i < j$ implies $e(i) < e(j)$ for all

$i, j \in [k]$, and it *induces the subsequence* $\mathsf{subseq}_e(w) = w[e(1)]w[e(2)]\dots w[e(k)]$ *of* w. For every $j \in [k-1]$, the j^{th} *gap of* w *induced by* e is the string $\mathsf{gap}_e(w, j) = w[e(j)+1..e(j+1)-1]$. We say that e is the embedding of $\mathsf{subseq}_e(w)$ in w.

An ℓ*-tuple of gap constraints* is a tuple $gc = (C_1, C_2, \dots, C_\ell)$ with $C_i \subseteq \Sigma^*$ for every $i \in [\ell]$. For convenience, we set $gc[i] = C_i$ for every $i \in [\ell]$. We say that an embedding e *satisfies a* $(k-1)$*-tuple of gap constraints* gc *with respect to a string* w if it has the form $e : [k] \to [|w|]$, and, for every $i \in [k-1]$, $\mathsf{gap}_e(w, i) \in C_i$. Moreover, for a $(k-1)$-tuple gc of gap constraints, the set $SubSeq(gc, w)$ contains all subsequences of w induced by embeddings that satisfy gc, i. e., $SubSeq(gc, w) = \{\mathsf{subseq}_e(w) \mid e \text{ is an embedding that satisfies } gc \text{ w. r. t. } w\}$. The elements of $SubSeq(gc, w)$ are also called the *gc-subsequences of* w. Note that tuples of gap constraints do not have constraints for the prefix $w[1..e(1)]$ or suffix $w[e(k)..|w|]$. However, our formalism can model this case too (for details, see this paper's full version [20]). For a $(|u|-1)$-tuple gc of gap constraints, we write $u \preceq_{gc} v$ to denote that $u \preceq_e v$ for some embedding $e : [|u|] \to [|v|]$ that satisfies gc with respect to v, i. e., $u \preceq_{gc} v$ means that u is a gc-subsequence of v. We note that for tuples of gap constraints $gc = (C_1, C_2, \dots, C_{k-1})$ with $C_i = \Sigma^*$ for every $i \in [k-1]$, the set $SubSeq(gc, w)$ is just the set of all length-k subsequences of w.

Special Types of Gap Constraints. We now define the types of gap constraints that are relevant for our work. We say that the gap constraints $gc = (C_1, \dots, C_{k-1})$ are

- *regular constraints* if $C_i \in \mathrm{REG}$ for every $i \in [k-1]$. For every $i \in [k-1]$, we represent the regular constraint C_i by a deterministic finite automaton (for short, DFA) A_i accepting it. See this paper's full version [20] for a discussion on the choice of DFAs to represent regular constraints.
- *length constraints* if, for every $i \in [k-1]$, there are $L^-(i), L^+(i) \in \mathbb{N} \cup \{0, +\infty\}$ with $L^-(i) \leq L^+(i)$, such that $C_i = \{v \in \Sigma^* \mid L^-(i) \leq |v| \leq L^+(i)\}$. We represent length constraints succinctly by pairs of numbers $(L^-(i), L^+(i))$, $i \in [k-1]$, in binary encoding.
- *reg-len constraints* if, for every $i \in [k-1]$, C_i is the conjunction of a regular constraint C'_i and a length constraint $(L^-(i), L^+(i))$, i. e., $C_i = C'_i \cap \{v \in \Sigma^* \mid L^-(i) \leq |v| \leq L^+(i)\}$. We represent such constraints by $((L^-(i), L^+(i)), A'_i)$, where A'_i is a DFA accepting C'_i.

A gap constraint C_i is a *zero-gap* if and only if $C_i = \{\varepsilon\}$. Let $\mathsf{nz}(gc)$ be the number of non-zero-gaps of gc (that is, the number of positions i such that $C_i \neq \{\varepsilon\}$). For a tuple of regular or reg-len gap constraints gc, let $\mathsf{size}(gc)$ be the size of the overall representation of the respective constraints (total size of the automata defining the constraints) and let $\mathsf{states}(gc)$ be the total number of states of the DFAs A_i, for $i \in [k-1]$, corresponding to the non-zero gaps of gc.

Clearly, length constraints are the simplest type of gap constraints considered above. In particular, length constraints, and therefore reg-len constraints, can also be seen as a particular case of regular constraints. However, transforming length or reg-len constraints into a single automaton may cause an exponential size increase.

Problems for Subsequences With Gap Constraints. In this paper, we investigate the matching problem Match and the analysis problems Uni, Con, and Equ (see definitions in the introduction). For simplicity, the pairs (p, gc), which play the role of the patterns in Match, will be called *gap-constrained sequences*, or simply *gapped sequences* for short. By Match_Σ, we denote the problem variant where all instances are over the fixed alphabet Σ; for some class $\mathcal{C}$ of gap constraints, we use "Match with $\mathcal{C}$-constraints" to refer to the variant where the constraints are from $\mathcal{C}$. We use analogous notations for the analysis problems.

If $gc = (\Sigma^*, \Sigma^*, \ldots, \Sigma^*)$, then Match boils down to the simple task of checking whether a given string is a subsequence of another string. The equivalence problem for such trivial gap constraints, on the other hand, boils down to the well-known problem of deciding the Simon congruence for two strings (see the discussion in the introduction). Our setting naturally models many other classical problems; some are discussed in Section 5. Finally, even though our framework allows arbitrary gap constraints, we will stick to the specific natural and relevant types of constraints defined above (i. e., length, regular, reg-len constraints).

3 Matching Gapped Subsequences

This section contains two main results. Firstly, we show that Match with reg-len constraints can be solved in $O(|w|\,\mathsf{states}(gc)+\mathsf{size}(gc))$ time, which implies also rectangular upper bounds for Match with either length or regular constraints. Secondly, we show that, assuming OVH holds, there are no algorithms solving any of these problems polynomially faster.

Note that, when dealing with length constraints, a constraint $(L^-(i), L^+(i))$ is equivalent to the regular language $C_i = \{x \in \Sigma^* \mid L^-(i) \leq |x| \leq L^+(i)\}$, which is accepted by a DFA with $\Theta(L^+(i))$ states. So, we could also interpret a tuple gc of reg-len constraints as a tuple of regular constraints only, by considering in each component of gc the intersection of the regular constraint with the regular language defined by the length constraints. However, this would lead to a growth in the number of states needed to model gc, and, as we will see in the following, to a less efficient algorithm for Match. In this setting, we state our first main result. The full proof is given in the full version of this paper [20]. To emphasise the merits of our approach, we overview in the full version of this paper [20] also several simpler approaches and their complexity (and shortcomings).

▶ **Theorem 1.** *Match with reg-len constraints can be solved in* $O(|w|(\mathsf{states}(gc) + 1) + \mathsf{size}(gc))$ *time.*

Proof Sketch. Assume $|w| = n$, $|p| = m$, and $gc = ((L^-(1), L^+(1)), A_1), \ldots, (L^-(m-1), L^+(m-1)), A_{m-1}))$, where $A_i = (Q_i, q_{0,i}, F_i, \delta_i)$ are DFAs for the regular constraints and $(L^-(i), L^+(i))$ are the length constraints. Let $i_1, \ldots, i_{k-1} \in [m-1]$ be such that $C_i \neq \{\varepsilon\}$ (i. e., C_i is a non-zero constraint of gc), for all $i \in \{i_1, \ldots, i_{k-1}\}$, and $C_i = \{\varepsilon\}$, for all $i \notin \{i_1, \ldots, i_{k-1}\}$. Clearly, $\mathsf{states}(gc) = \sum_{j=1}^{k-1} |Q_{i_j}| \geq \mathsf{nz}(gc)$. With $i_0 = 0$ and $i_k = m$, we compute the words $p_j = p[(i_{j-1}+1)..i_j]$, for $j \in [k]$, and we construct in linear time longest common extension data structures (see [21]) for the word $x = wp$, allowing us to check in constant time whether $w[i+1..i+|p_j|] = p_j$, for $i \leq n$.

The main part of our algorithm consists in a dynamic programming approach. We compute a two-dimensional $n \times k$ array $D[\cdot][\cdot]$ (initialised with 0), where $D[i][\ell] = 1$ iff $p[1..|p_1 \cdots p_\ell|]$ can be embedded in $w[1..i]$ such that the first $\ell - 1$ non-zero constraints are satisfied, and p_ℓ is mapped to $w[1..i]$'s length-$|p_\ell|$ suffix. Otherwise, $D[i][\ell] = 0$. We first set $D[i][1] = 1$ iff $w[i - |p_1| + 1..i] = p_1$. Further, assume that, for some $t \in [k-1]$, we have computed $D[\cdot][\ell]$, for all $\ell \leq t$, and we want to compute $D[\cdot][t+1]$. The main component of this most involved part is computing an array $f_{t+1}[\cdot]$, with n elements, such that $f_{t+1}[i] = 1$ iff there exists a position j for which $D[j][t] = 1$, $w[j+1..i] \in L(A_t)$, and $L^-(t) \leq |w[j+1..i]| \leq L^+(t)$. We now sketch this step (a full description is given in the full version of this paper [20]).

We first collect in a list $L_{t+1} = j_1 < \ldots < j_r$ (increasingly sorted) all the positions j of w with $D[j][t] = 1$. We then compute a graph G_{t+1} with nodes (i, q), with $i \in [n]$ and $q \in Q_t$, that consists of the union, over $j \in L_{t+1}$, of the (not necessarily disjoint) paths

$[(j, q_{0,t}), (j+1, q_1^j), \ldots, (n, q_{n-j}^j)]$, where $\delta_t(q_{0,t}, w[j+1]) = q_1^j$ and $\delta_t(q_r^j, w[j+r+1]) = q_{r+1}^j$, for all $r \in [n-j-1]$. Intuitively, such a path records the trace of the computation of A_t on the input $w[j+1..n]$. These paths (and, therefore, the graph G_{t+1}) can be simultaneously constructed to avoid redundant computations. A_t is deterministic, thus, if two such paths intersect, then they are identical after their first common node. Consequently, G_{t+1} is a collection of disjoint trees $T_1, T_2, \ldots, T_z$. As there are no edges between any pair of nodes (n, q) and (n, q'), with $q, q' \in Q_t$, each such tree T_i can be seen as a rooted tree, whose root is its single node of the form (n, q) and whose leaves are some of the nodes $(j, q_{0,t})$, with $j \in L_{t+1}$.

For each tree T_i and for each leaf $(j, q_{0,t})$ of T_i, we mark all the ancestors (d, q) of $(j, q_{0,t})$ such that $L^-(t) \leq |w[j+1..d]| = d - j \leq L^+(t)$. After this, a node (j, q) of T_i is marked iff there exists a length-ℓ path $\mathcal{P}$, with $L^-(t) \leq \ell \leq L^+(t)$, which connects a leaf $(j', q_{0,t})$ of T_i to (j, q), i. e., $\delta_t(q_{0,t}, w[j'+1..j]) = q$. By using efficient data structures, computing and marking the trees takes time $O(n|Q_t|)$ and $O(\sum_{i=1}^{p} |T_i|)$, resp.

Finally, for $i = 1, \ldots, n$, we set $f_{t+1}[i] = 1$ iff there is a state $q \in F_t$ such that node (i, q) is marked. This means that $f_{t+1}[i] = 1$ iff there is a word $w[j+1..i]$ of length ℓ, with $L^-(t) \leq \ell \leq L^+(t)$, such that $j \in L_{t+1}$ and $\delta_t(q_{0,t}, w[j+1..i])$ is a final state (i. e., $w[j+1..i] \in L(A_t)$).

For computing the elements of D, we set $D[i][t+1] = 1$ iff $w[i - |p_{t+1}| + 1..i] = p_{t+1}$ and $f_{t+1}[i - |p_{t+1}|] = 1$. Then, we decide that $p \preceq_{gc} w$ iff there exists j with $D[j][k] = 1$.

The whole process can be implemented in $O(|w| \, \mathsf{states}(gc) + \mathsf{size}(gc))$ time. ◀

The next results are now immediate. Note that for these particular cases (but, to the best of our knowledge, not for their conjunction, covered in Theorem 1) simpler algorithms exist.

▶ **Corollary 2.**

(1) *Match with length constraints can be solved in* $O(|w|(\mathsf{nz}(gc) + 1))$ *time.*

(2) *Match with regular constraints can be solved in* $O(|w|(\mathsf{states}(gc) + 1) + \mathsf{size}(gc))$ *time.*

It is worth noting that the matching problem can be solved in $O(|w|)$ time when gc only defines constraints that are $\{\varepsilon\}$ or Σ^*, which covers, e. g., the cases of subsequence matching or string matching. In particular, the greedy strategy used for matching regular patterns with variables (see, e. g., [23]) can be easily adapted to solve Match with length constraints in linear time, when the upper bounds on each gap are trivial (i. e., they are all greater or equal to the length of the input word). So, as far as length constraints are concerned, it seems that non-trivial upper bounds lead to an increase in the difficulty of the Match problem; a particularly efficient approach for subsequences with general length constraints is given in [10], but, in the worst case, it still has rectangular complexity. However, even when non-trivial length upper bounds are used, there are still some simpler particular cases. For instance, when working with *strings with don't cares* (or partial words), where each gap has a fixed length (i. e., the lower and upper bounds are the same), Match can be solved in time $O(|w| \log |p|)$ [18].

A gapped sequence (p, gc) with reg-len constraints can be represented as a classical regular expression $r_{(p,gc)}$, so Match can be solved by a textbook algorithm in $O(|w||r_{(p,gc)}|)$ [62], which is optimal w. r. t. polynomial speed-ups, conditional on OVH [6]. However, including the string p and the length constraints in the regular expression might, once more, lead to a slower algorithm compared to our direct approach, as $|r_{p,gc}|$ may be much larger than $\mathsf{states}(gc)$.

To summarise, at an intuitive level, we could say that as long as we have non-trivial length or regular constraints, MATCH seems to become more difficult than its counterpart for classical subsequences. This intuitive remark is confirmed by our second main result.

▶ **Theorem 3.** *MATCH with length constraints cannot be solved in $\mathcal{O}(|w|^h \,\mathsf{nz}(gc)^g)$ time with $h + g = 2 - \epsilon$ for some $\epsilon > 0$, unless OVH fails. This holds even if $|\Sigma| = 4$ and all length constraints are $(0, \ell)$ with $\ell \leq 6$.*

Proof Sketch. Let $A = \{\vec{a}_1, \ldots, \vec{a}_n\}$ and $B = \{\vec{b}_1, \ldots, \vec{b}_n\}$, with $A, B \subset \{0,1\}^d$ be an OV-instance. We transform A into a string $w \in \Sigma^* = \{0, 1, \#, @\}^*$ and B into a string $p \in \Sigma^*$ and a $(|p|-1)$-tuple gc of length constraints. For convenience, we represent the gapped sequence (p, gc) with $p = p[1] \cdots p[m]$ by writing the length constraints in between the symbols, i. e., $p[1] \overset{gc[1]}{\leftrightarrow} p[2] \overset{gc[2]}{\leftrightarrow} \cdots \overset{gc[m-1]}{\leftrightarrow} p[m]$, and we omit $\overset{gc[i]}{\leftrightarrow}$ if $gc[i] = (0,0)$. For example, if $p = abab$ and $gc[1] = (0,0)$, $gc[2] = (1,5)$, and $gc[3] = (0,6)$, we use the notation $ab \overset{(1,5)}{\leftrightarrow} a \overset{\leq 6}{\leftrightarrow} b$.

Let $\vec{a}_i = (a_i^1, \ldots, a_i^d)$ and $\vec{b}_i = (b_i^1, \ldots, b_i^d)$, for all $i \in [n]$. We shall represent the vectors from A and B by different encodings $\mathsf{C}_a(\cdot)$ and $\mathsf{C}_b(\cdot)$, respectively. The 0 and 1 entries in the A-vectors are encoded by $\mathsf{C}_a(0) = 010$ and $\mathsf{C}_a(1) = 100$, and the 0 and 1 entries in the B-vectors are encoded by $\mathsf{C}_b(0) = 10$ and $\mathsf{C}_b(1) = 01$. We note that for every $x, y \in \{0,1\}$, $\mathsf{C}_b(x)$ is a factor of $\mathsf{C}_a(y)$ if and only if $x \cdot y = 0$. This means that the orthogonality of $\vec{a}_i$ and $\vec{b}_{i'}$ is characterised by the situation that, for every $j \in [d]$, $\mathsf{C}_b(b_{i'}^j)$ is a factor of $\mathsf{C}_a(a_i^j)$.

We represent each bit a_i^j of $\vec{a}_i \in A$ as the string $\#\,\mathsf{C}_a(0)\#\#\,\mathsf{C}_a(a_i^j)\#\#\,\mathsf{C}_a(0)\#$, and the vector $\vec{a}_i$ as the concatenation $\mathsf{C}_a(\vec{a}_i) = \prod_{j=1}^{d}([_1\#\,\mathsf{C}_a(0)\#]_1[_2\#\,\mathsf{C}_a(a_i^j)\#]_2[_3\#\,\mathsf{C}_a(0)\#]_3)$, where the brackets $[_1\ldots]_1$, $[_2\ldots]_2$, $[_3\ldots]_3$ are not actual symbols of the gadget, but serve the only purpose to illustrate that $\mathsf{C}_a(\vec{a}_i)$ has three individual *tracks*, where track 1 and 3 correspond to d occurrences of $\#\,\mathsf{C}_a(0)\#$ (representing the all-0 vector), while track 2 represents the actual vector $\vec{a}_i$. These three tracks play a central role in the correctness of the reduction.

For $i \in [n]$, every vector $\vec{b}_i \in B$ is also represented by listing all bit encodings $\mathsf{C}_b(b_i^j)$, but in a slightly different way and, most importantly, as a gapped sequence (in the notation defined above): $(\mathsf{C}_b(\vec{b}_i), gc_i) = \left(\prod_{j=1}^{d-1}(\# \overset{\leq 1}{\leftrightarrow} \mathsf{C}_b(b_i^j) \overset{\leq 1}{\leftrightarrow} \#\# \overset{\leq 3}{\leftrightarrow} \#\# \overset{\leq 3}{\leftrightarrow} \#)\right) \# \overset{\leq 1}{\leftrightarrow} \mathsf{C}_b(b_i^d) \overset{\leq 1}{\leftrightarrow} \#$.

It can be shown (see the full version of this paper [20]) that if $\mathsf{C}_b(\vec{b}_i) \preceq_e \mathsf{C}_a(\vec{a}_\ell)$ and e satisfies gc_i, then the embedding e maps each $\mathsf{C}_b(b_i^j)$ to the $\mathsf{C}_a(0)$ of $\mathsf{C}_a(\vec{a}_\ell)$'s first track, or each $\mathsf{C}_b(b_i^j)$ to the $\mathsf{C}_a(a_\ell^j)$ of $\mathsf{C}_a(\vec{a}_\ell)$'s second track, or each $\mathsf{C}_b(b_i^j)$ to the $\mathsf{C}_a(0)$ of $\mathsf{C}_a(\vec{a}_\ell)$'s third track. More precisely, due to how we use the symbols $\#$, the factor $\mathsf{C}_b(b_i^1)$ must be mapped to $[_1\#\,\mathsf{C}_a(0)\#]_1$ or to $[_2\#\,\mathsf{C}_a(a_i^1)\#]_2$ or to $[_3\#\,\mathsf{C}_a(0)\#]_3$. Since we have 4 occurrences of $\#$ between each $\mathsf{C}_b(b_i^j)$ and $\mathsf{C}_b(b_i^{j+1})$, and between two consecutive parts of the same track in $\mathsf{C}_a(\vec{a}_i)$, all the following factors $\mathsf{C}_b(b_i^2), \mathsf{C}_b(b_i^3), \ldots$ must be mapped to the same track $\mathsf{C}_b(b_i^1)$ is mapped to. This is illustrated in Figure 1. Based on these considerations, it is clear that $\mathsf{C}_b(\vec{b}_i) \preceq_e \mathsf{C}_a(\vec{a}_\ell)$ with e mapping $\mathsf{C}_b(\vec{b}_i)$ to $\mathsf{C}_a(\vec{a}_\ell)$'s second track is possible if and only if $\vec{a}_\ell$ and $\vec{b}_i$ are orthogonal.

$\mathsf{C}_b(\vec{b}_i) =$	#	$\overset{\leq 1}{\leftrightarrow}\,\mathsf{C}_b(b_i^1)\,\overset{\leq 1}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	#	$\overset{\leq 1}{\leftrightarrow}\,\mathsf{C}_b(b_i^2)\,\overset{\leq 1}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	…
$\mathsf{C}_b(\vec{b}_i) =$				#	$\overset{\leq 1}{\leftrightarrow}\,\mathsf{C}_b(b_i^1)\,\overset{\leq 1}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	#	$\overset{\leq 1}{\leftrightarrow}\,\mathsf{C}_b(b_i^2)\,\overset{\leq 1}{\leftrightarrow}$	#	…
$\mathsf{C}_b(\vec{b}_i) =$							#	$\overset{\leq 1}{\leftrightarrow}\,\mathsf{C}_b(b_i^1)\,\overset{\leq 1}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	#	$\overset{\leq 3}{\leftrightarrow}$	#	…
$\mathsf{C}_a(\vec{a}_\ell) =$	#	$\mathsf{C}_a(0)$	#	#	$\mathsf{C}_a(a_i^j)$	#	#	$\mathsf{C}_a(0)$	#	#	$\mathsf{C}_a(0)$	#	#	$\mathsf{C}_a(a_i^j)$	#	…

Figure 1 Possible embeddings of $\mathsf{C}_b(\vec{b}_i)$ in $\mathsf{C}_a(\vec{a}_\ell)$, selecting its first, second, or third track.

The remaining challenge is to combine the gadgets $\mathsf{C}_a(\vec{a}_i)$ into a string w, and the gadgets $(\mathsf{C}_b(\vec{b}_i), gc_i)$ into a gapped sequence (p, gc), such that $p \preceq_e w$ for an embedding e satisfying gc if and only if e is such that every $(\mathsf{C}_b(\vec{b}_i), gc_i)$ is mapped to some $\mathsf{C}_a(\vec{a}_\ell)$, and there is

necessarily at least one pair $i, \ell \in [n]$ such that $(\mathsf{C}_b(\vec{b}_i), gc_i)$ is embedded into $\mathsf{C}_a(\vec{a}_\ell)$'s second track. We next define w and (p, gc), and then discuss why they satisfy the property from above:

$$w = \left(\prod_{i=1}^{n-1} @\, \mathsf{C}_a(\vec{a}_i)\right) @\, \mathsf{C}_a(\vec{a}_n) \left(\prod_{i=1}^{n-1} @\, \mathsf{C}_a(\vec{a}_i)\right) @\,,$$
$$(p, gc) = @ \overset{\leq 5}{\leftrightarrow} \left(\prod_{j=1}^{n-1} \mathsf{C}_b(\vec{b}_j) \overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 6}{\leftrightarrow}\right) \mathsf{C}_b(\vec{b}_n) \overset{\leq 5}{\leftrightarrow} @.$$

If $p \preceq_e w$ for an embedding e satisfying gc, then the first @-symbol of p is mapped to the @-symbol of w occurring before an occurrence of $\mathsf{C}_a(\vec{a}_{\ell_1})$ for some ℓ_1, and this occurrence is in the prefix $\left(\prod_{i=1}^{n-1} @\, \mathsf{C}_a(\vec{a}_i)\right)$ of w. By reasoning about the occurrences of symbols # and the length constraints (see the full version of this paper [20]), we can show that $\mathsf{C}_b(\vec{b}_1)$ must be embedded in $\mathsf{C}_a(\vec{a}_{\ell_1})$ in the way discussed above (i. e., gc_1 is satisfied and $\mathsf{C}_b(\vec{b}_1)$ is entirely mapped to some track $q \in \{1, 2, 3\}$ of $\mathsf{C}_a(\vec{a}_{\ell_1})$). For simplicity, assume that $\ell_1 \leq n-1$. The factor $\overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 1}{\leftrightarrow} \# \overset{\leq 3}{\leftrightarrow} \# \overset{\leq 6}{\leftrightarrow}$ between $(\mathsf{C}_b(\vec{b}_1), gc_1)$ and the next part $(\mathsf{C}_b(\vec{b}_2), gc_2)$ will enforce that $\mathsf{C}_b(\vec{b}_2)$ is embedded in $\mathsf{C}_a(\vec{a}_{\ell_1+1})$, and, moreover, it will be mapped to $\mathsf{C}_a(\vec{a}_{\ell_1+1})$'s track q or $q+1$ (as there can be at most 18 symbols between $\mathsf{C}_b(\vec{b}_1)$ and $\mathsf{C}_b(\vec{b}_2)$, track 3 cannot be reached in the case $q = 1$).

By repeating this argument, we can show that if $(\mathsf{C}_b(\vec{b}_j), gc_j)$ is embedded in track s of $\mathsf{C}_a(\vec{a}_{\ell_j})$ (with $\ell_j \leq n-1$), then $(\mathsf{C}_b(\vec{b}_{j+1}), gc_{j+1})$ is embedded in track s or $s+1$ of $\mathsf{C}_a(\vec{a}_{\ell_j+1})$ in case that $s \in \{1, 2\}$, and it is necessarily embedded in track s of $\mathsf{C}_a(\vec{a}_{\ell_j+1})$ in case that $s = 3$. If $\ell_j = n$, then analogously $(\mathsf{C}_b(\vec{b}_{j+1}), gc_{j+1})$ is mapped to $\mathsf{C}_a(\vec{a}_1)$ of w's suffix $(\prod_{i=1}^{n-1} @\, \mathsf{C}_a(\vec{a}_i))@$. Consequently, each $\mathsf{C}_b(\vec{b}_j)$ is mapped to a track of $\mathsf{C}_a(\vec{a}_{\ell_j})$, and the tracks to which these $\mathsf{C}_b(\vec{b}_j)$ are mapped may start with track 1 or 2, and then can only increase until we possibly map some $\mathsf{C}_b(\vec{b}_j)$ to track 3. However, after having mapped the last occurrence of # in $\mathsf{C}_b(\vec{b}_n)$ to an occurrence of # in $\mathsf{C}_a(\vec{a}_{\ell_n})$, we can afford a gap of length at most 5 before mapping the last symbol @ of (p, gc) to an occurence of @ in w. By the structure of (p, gc) and w, this is only possible if $\mathsf{C}_b(\vec{b}_n)$ is mapped to track 2 or 3 of $\mathsf{C}_a(\vec{a}_{\ell_n})$.

We conclude that if $p \preceq_{gc} w$, then, for some $j, \ell_j \in [n]$, $(\mathsf{C}_b(\vec{b}_j), gc_j)$ is mapped to track 2 of $\mathsf{C}_a(\vec{a}_{\ell_j})$; thus, $\vec{a}_{\ell_j}$ and $\vec{b}_j$ are orthogonal. On the other hand, the explanations from above show that if $\vec{a}_{\ell_j}$ and $\vec{b}_j$ are orthogonal vectors, then p can be embedded into w by an embedding that satisfies gc, i. e., an embedding that maps $(\mathsf{C}_b(\vec{b}_j), gc_j)$ to track 2 of $\mathsf{C}_a(\vec{a}_{\ell_j})$, all $(\mathsf{C}_b(\vec{b}_{j'}), gc_{j'})$ with $1 \leq j' < j$ to the first tracks of some $\mathsf{C}_a(\vec{a}_{\ell_{j'}})$, and all $(\mathsf{C}_b(\vec{b}_{j'}), gc_{j'})$ with $j < j' \leq n$ to the third tracks of some $\mathsf{C}_a(\vec{a}_{\ell_{j'}})$.

In this reduction, we have $|\Sigma| = 4$, all constraints are $(0, \ell)$ with $\ell \leq 6$, and $|w|, |p| \in \Theta(nd)$. If Match can be solved in $O(|w|^g |p|^h)$ with $g + h = 2 - \epsilon$ for some $\epsilon > 0$, then OV can be solved in $O(nd + (nd)^{2-\epsilon})$. Since $\mathsf{nz}(gc) \in \Theta(nd)$, solving Match in $O(|w|^g\, \mathsf{nz}(gc)^h)$ with $g + h = 2 - \epsilon$ for some $\epsilon < 0$ also contradicts OVH. ◀

We emphasise that, according to our proof, these lower bounds hold for Match with length constraints even if we only have constant upper bounds on the length of the gaps.

▶ **Corollary 4.** *Match with regular constraints cannot be solved in $\mathcal{O}(|w|^h\, \mathsf{states}(gc_p)^g)$ time with $h + g = 2 - \epsilon$ for some $\epsilon > 0$, unless OVH fails. This holds even if $|\Sigma| = 4$ and all regular constraints are expressed by constant size DFAs.*

From Theorem 3 and Corollary 4, we also get that Match with length, regular, or reg-len constraints cannot be solved in $\mathcal{O}(|w|^h |p|^g)$ time, with $h + g = 2 - \epsilon$, nor in $\mathcal{O}(|w|^{2-\epsilon})$ time. Moreover (see the full version of this paper [20]) we can show similar lower bounds for $|\Sigma| = 2$ as well.

Compared to the OVH-bound for regular expression matching of [6], we provide a lower bound for a much more restricted problem (i. e., matching gapped sequences with length constraints, a subclass of regular expressions that still seems to have a significant practical relevance); thus, a stronger lower bound (this is also why our OV-reduction has a significantly different structure and is technically more involved than that of [6]). In particular, our lower bound applies (unlike those from [6]) to the case of matching variable length gap patterns, and settles the complexity of that problem. We wrap up this section by noting that Theorem 3 and Corollary 4 show that (if OVH holds) the algorithm of Theorem 1, also when used for regular constraints or length constraints only, is optimal in the sense that there are no algorithms which can solve Match in the respective settings polynomially faster.

4 Analysis Problems for Gapped Subsequences

Let us recall that the *universality*, *containment* and *equivalence problem* (denoted by Uni, Con and Equ for short) consist in deciding $SubSeq(gc, w) = \Sigma^k$, $SubSeq(gc, w) \subseteq SubSeq(gc, w')$, and $SubSeq(gc, w) = SubSeq(gc, w')$, respectively, for a given $(k-1)$-tuple gc of gap constraints and strings $w, w' \in \Sigma^*$. As mentioned before, these problems can be solved in polynomial time for classical subsequences (see the full version of this paper [20] for further details). We show next that these problems become much harder for non-trivial length or regular constraints.

From Corollary 2 and Theorem 1, we can directly conclude the following brute-force upper bounds.

▶ **Theorem 5.**

(1) *The problems Uni, Con and Equ with length (or reg-len) constraints can be solved in time* $O(|\Sigma|^k \, \mathsf{nz}(gc)\ell)$ *(respectively,* $O(|\Sigma|^k \, \mathsf{states}(gc)\ell)$*), where* $\ell = \max\{|w|, |w'|\}$.

(2) *The problems* Uni_Σ, Con_Σ *and* Equ_Σ *with length (or reg-len) constraints can be solved in time* $2^{O(k)} \, \mathsf{nz}(gc)\ell$ *(respectively,* $2^{O(k)} \, \mathsf{states}(gc)\ell$*), where* $\ell = \max\{|w|, |w'|\}$.

We shall next complement these brute-force upper bounds by suitable lower bounds, which demonstrate that significantly faster algorithms are unlikely to exist. For convenience, we state our complexity results for the complement problems, i. e., *non-universality problem* (NUni), *non-containment problem* (NCon), and *non-equivalence problem* (NEqu). Moreover, we state the lower bounds for the case of length constraints only. By simply interpreting the length constraints as regular constraints, all the lower bounds also apply to the case of regular constraints (this does not cause an exponential size increase of the instances, see the full version of this paper [20].

Our first result establishes the general NP-completeness (even for small constant alphabets and length constraints), and that the exponent $O(k)$ of Theorem 5(2) cannot be significantly improved, unless ETH or SETH fail. We will discuss some proof ideas later on.

▶ **Theorem 6.** *For every fixed alphabet* Σ *with* $|\Sigma| \geq 3$, NUni_Σ, NCon_Σ *and* NEqu_Σ *with length constraints are* NP*-complete, even if all length constraints are* $(1, 5)$. *Moreover,*

- *they cannot be solved in subexponential time* $2^{o(k)} \operatorname{poly}(|w|, k))$ *(unless ETH fails),*
- *they cannot be solved in time* $O(2^{k(1-\epsilon)} \operatorname{poly}(|w|, k))$ *(unless SETH fails).*

This directly leads to the question whether these problems are tractable if $|\Sigma| \leq 2$. This is obviously true for unary alphabet $\Sigma = \{\mathsf{a}\}$ (note that in this case, $SubSeq(gc, w) = \Sigma^k = \{\mathsf{a}^k\}$ if $(\sum_{i \in [k]} L^-(i)) + k \leq |w|$, and $SubSeq(gc, w) = \emptyset$ otherwise), but NP-complete for $|\Sigma| = 2$:

$$\begin{pmatrix} \{\mathtt{a}\} & \{\mathtt{b}\} & \{\mathtt{a},\mathtt{c},\mathtt{d}\} \\ \{\mathtt{c},\mathtt{d}\} & \{\mathtt{b}\} & \{\mathtt{a}\} \\ \{\mathtt{a}\} & \{\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e}\} & \{\mathtt{d}\} \\ \{\mathtt{e}\} & \{\mathtt{b}\} & \{\mathtt{c},\mathtt{e}\} \end{pmatrix} \qquad \begin{pmatrix} \{0\} & \{1\} & \{0\} & \{0,1\} & \{0,1\} & \{0,1\} \\ \{1\} & \{1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0\} \\ \{0,1\} & \{0,1\} & \{0\} & \{0\} & \{0\} & \{0,1\} \end{pmatrix}$$

Figure 2 Left side: example instance of MetaNUni for $\Gamma = \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e}\}$, $q = 4$, and $k = 3$. Note that, e. g., $W_{3,2} = \{\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e}\}$ and $W_{4,1} = \{\mathtt{e}\}$; moreover, $\mathcal{L}(W_1) = \{\mathtt{a}\} \cdot \{\mathtt{b}\} \cdot \{\mathtt{a},\mathtt{c},\mathtt{d}\} = \{\mathtt{a}\,\mathtt{b}\,\mathtt{a}, \mathtt{a}\,\mathtt{b}\,\mathtt{c}, \mathtt{a}\,\mathtt{b}\,\mathtt{d}\}$. Since $\cup_{i\in[4]}\mathcal{L}(W_i) \neq \Gamma^3$, this is a negative instance. Right side: the CNF-Sat-instance $c_1 = \{v_1, \neg v_2, v_3\}$, $c_2 = \{\neg v_1, \neg v_2, v_5\}$, $c_3 = \{v_3, v_4, v_5\}$ over the variables $\{v_1, v_2, \ldots, v_6\}$ as an instance of MetaNUni . Note that $100010 \notin \cup_{i\in[3]}\mathcal{L}(W_i)$; thus, 100010 is a satisfying assignment.

▶ **Theorem 7.** *For every fixed alphabet Σ with $|\Sigma| = 2$, NUni$_\Sigma$, NCon$_\Sigma$ and NEqu$_\Sigma$ with length constraints are* NP*-complete even if each length constraint is* $(0, 0)$ *or* $(3, 9)$.

Let us now consider the case where Σ is not treated as a constant. Theorem 5 means that NUni, NCon and NEqu with length constraints are trivially fixed parameter tractable if parameterised by both $|\Sigma|$ and k. Moreover, since $\ell = \max\{|w|, |w'|\}$ bounds both $|\Sigma|$ and k, we also have fixed parameter tractability with respect to ℓ for trivial reasons. Are the problems fixed-parameter tractable with respect to the single parameter $|\Sigma|$ or the single parameter k? With respect to $|\Sigma|$, this is answered in the negative by Theorem 7 (unless P = NP). With respect to parameter k, the following result gives a negative answer as well.

▶ **Theorem 8.** *Problems NUni, NCon and NEqu with length constraints cannot be solved in running time* $\mathrm{O}(f(k)\,\mathrm{poly}(|w|, k))$ *for any computable function f (unless* FPT = W[1]*).*

This result only holds for unbounded alphabets and length constraints. Indeed, for constant Σ the brute-force algorithm is an fpt-algorithm with respect to k. Moreover, if the upper length constraints are bounded by some constant ℓ, then we only have to enumerate at most ℓ^{k-1} candidate tuples of gap sizes and check whether one of them induces an embedding satisfying gc with respect to w, which again would yield an fpt-algorithm with respect to k.

Proof Ideas for the Lower Bounds. We only consider the non-universality problem here. Full proof details can be found in the full version of this paper [20]. Theorem 6 can be proven by a reduction from CNF-Sat. In order to get the ETH and SETH lower bounds, this reduction must yield instances with a $(k-1)$-tuple of gap constraints, where k is the number of Boolean variables. Theorem 8 can be shown by a similar reduction that starts from the standard parameterisation of the independent set problem. Both reductions can be defined by using a *meta non-universality problem* (MetaNUni for short) as an intermediate step, which we define next.

Let $\Gamma = \{b_1, b_2, \ldots, b_m\}$ be some alphabet, and let $q, k \in \mathbb{N}$. An instance of the problem is a $(q \times k)$-matrix with the entries $W_{i,j}$, which are subsets of Γ. For every $i \in [q]$, we associate with row i of the matrix the language $\mathcal{L}(W_i) = W_{i,1} \cdot W_{i,2} \cdots W_{i,k}$, i. e., we simply represent the elements of $W_{i,1} \times W_{i,2} \times \ldots \times W_{i,k}$ as length-k strings over Γ in the natural way. The question is then to decide whether $\cup_{i\in[q]}\mathcal{L}(W_i) \neq \Gamma^k$ (see Figure 2 for an example).

We next discuss, how we can reduce CNF-Sat to MetaNUni. Let $F = \{c_1, c_2, \ldots, c_q\}$ be a Boolean formula in CNF on variables $\{v_1, \ldots, v_k\}$ (i. e., $c_i \subseteq \{v_1, \neg v_1, \ldots, v_k, \neg v_k\}$). We define alphabet $\Gamma = \{0, 1\}$ and the $(q \times k)$-matrix with the entries $W_{i,j}$ as follows. For every $i \in [q]$ and $j \in [k]$, we define $W_{i,j} = \{0\}$, if $v_j \in c_i$, $W_{i,j} = \{1\}$, if $\neg v_j \in c_i$, and $W_{i,j} = \{0, 1\}$, if $\{v_j, \neg v_j\} \cap c_i = \emptyset$. It can be verified with moderate effort, that for every $i \in [q]$, $\mathcal{L}(W_i)$ contains exactly the Boolean assignments that do not satisfy clause c_i. Hence, $\cup_{i\in[q]}\mathcal{L}(W_i) \neq \{0, 1\}^k$ if and only if F is satisfiable (see Figure 2 for an example).

In a rather similar way, we can also phrase the independent set problem in terms of MetaNUni. For the independent set problem, we get an undirected graph $G = (V, E)$ with $|V| = n$ and $E = \{e_1, e_2, \ldots, e_m\}$, and a $k \in [|V|]$, and the question is whether G has a k-independent set, i. e., a set $A \subseteq V$ with $|A| = k$ and $\{u, u'\} \notin E$ for every $u, u' \in A$ with $u \neq u'$. This can be expressed in terms of MetaNUni as follows. We interpret the set V of vertices as the alphabet Γ. We fix some bijection $\nu : \{(i, r, s) \in [m] \times [k] \times [k] \mid r \neq s\} \to [mk(k-1)]$. For every $i \in [m]$ with $e_i = (u, v)$, and every $r, s, j \in [k]$ with $r \neq s$, we define $W_{\nu(i,r,s),j} = \{u\}$, if $j = r$, $W_{\nu(i,r,s),j} = \{v\}$, if $j = s$, and $W_{\nu(i,r,s),j} = V$, else. For example, if $e_9 = (v_3, v_7)$ and $k = 4$, then row $\nu(9, 2, 4)$ of the matrix would be $V\ \{v_3\}\ V\ \{v_7\}$.

It is a bit more difficult to see why this reduction works. The idea is that we represent sets of vertices of cardinality *at most* k by length-k strings over V (note that sets of cardinality strictly less than k can be represented by strings with repeated symbols). For every edge (u, v) and for all pairs of positions $r, s \in [k]$, the language $\mathcal{L}(W_{\nu(i,r,s)}) = W_{\nu(i,r,s),1} W_{\nu(i,r,s),2} \ldots W_{\nu(i,r,s),k}$ represented by row $\nu(i, r, s)$ of the matrix contains exactly the strings $w \in \Gamma^k$ with $(w[r], w[s]) = (u, v)$, i. e., strings that represent non-independent sets with edge (u, v). For the example $e_9 = (v_3, v_7)$ and $k = 4$, we have $\mathcal{L}(W_{\nu(9,2,4)}) = \{v_1 v_3 v_1 v_7,\ v_2 v_3 v_1 v_7,\ \ldots,\ v_n v_3 v_1 v_7,\ \ldots,\ v_1 v_3 v_2 v_7,\ v_2 v_3 v_2 v_7,\ \ldots\}$.

This whole idea works only because, in our setting, we assume that every vertex has a loop since then strings w of V^k contain an edge $(w[r], w[s]) \in E$ for some $r, s \in [k]$ if and only if the corresponding set of vertices is not independent or of cardinality strictly less than k (the latter is represented by a loop, i. e., $w[r] = w[s]$). In summary, G has a k-independent set if and only if not all length-k strings are in $\bigcup_{i \in [m], r, s \in [k], r \neq s} \mathcal{L}(W_{\nu(i,r,s)})$.

The main technical challenge is to show a reduction from MetaNUni to NUni with length constraints. We next give a sketch of this reduction. Let $\Gamma = \{b_1, b_2, \ldots, b_m\}$, $q, k \in \mathbb{N}$, and, for every $i \in [q], j \in [k]$, let $W_{i,j} \subseteq \Gamma$. We transform this MetaNUni instance into an instance of NUni with length constraints as follows. We first define the alphabet $\Sigma = \Gamma \cup \{\#\}$ (with $\# \notin \Gamma$). Then we define a $(k-1)$-tuple $gc = (C_1, C_2, \ldots, C_{k-1})$ of gap constraints with $C_i = (L^-(i), L^+(i)) = (m - 1, 3m - 1)$ for every $i \in [k-1]$ (recall that m is Γ's cardinality). To conclude the reduction, we have to construct a string $K(W_1, \ldots, W_q)$ over Σ, such that $SubSeq(gc, K(W_1, \ldots, W_q)) = \Sigma^k$ if and only if $\cup_{i \in [q]} \mathcal{L}(W_i) = \Gamma^k$. We do this in several steps.

For every $i \in [q]$ and $j \in [k]$, let $w_{i,j} \in \Gamma^*$ be some string representation of $W_{i,j}$, i. e., $\mathsf{alph}(w_{i,j}) = W_{i,j}$ and $|w_{i,j}| = |W_{i,j}| \leq m$. For every $i \in [q]$, we define the string

$$S(W_i) = w_{i,1}(\#)^{m-1} w_{i,2}(\#)^{m-1} \ldots (\#)^{m-1} w_{i,k}\,.$$

We can show that those gc-subsequences of $S(W_i)$ that do not contain occurrences of symbol $\#$ must be mapped to $S(W_i)$ in such a way that each $j \in [k]$ is mapped to $w_{i,j}$. More precisely, for every $i \in [q]$, we have that $(SubSeq(gc, S(W_i)) \cap \Gamma^*) = \mathcal{L}(W_i)$. (†)

Next, we define a string T whose purpose it is to contain *all* gc-subsequences that contain at least one occurrence of $\#$. For every $i \in [k]$, let $T_i = T_{i,1} T_{i,2} \ldots T_{i,k}$, where, for every $j \in [k] \setminus \{i\}$, $T_{i,j} = b_1 b_2 \ldots b_m \#^m$, and $T_{i,i} = \#^m$. We define T by $T = T_1(\#^{3m}) T_2(\#^{3m}) \ldots (\#^{3m}) T_k$. The idea here is that any gc-subsequence of T must be mapped entirely into some T_i, which, due to the length constraints, forces position i to be mapped to $T_{i,i} = \#^m$, i. e., to an occurrence of $\#$. More precisely, we have $SubSeq(gc, T) = \{w \in \Sigma^k \mid |w|_\# \geq 1\}$. (⋄)

Finally, we set $K(W_1, \ldots, W_q) = T(\#^{3m}) S(W_1)(\#^{3m}) S(W_2)(\#^{3m}) \ldots (\#^{3m}) S(W_q)$. By using (†) and (⋄) from above, we can now prove $SubSeq(gc, K(W_1, \ldots, W_q)) = \Sigma^k$ if and only if $\cup_{i \in [q]} \mathcal{L}(W_i) = \Gamma^k$, which concludes the proof of correctness.

For proving Theorem 7, using MetaNUni as an intermediate step seems not possible, since it introduces another symbol $\#$ to the alphabet. However, we can devise a similar reduction. The main difference is that we represent each Boolean variable by two consecutive symbols of the subsequence, i. e., we need a $(2k-1)$ tuple of length constraints (therefore,

the reduction does not yield a SETH bound as mentioned in Theorem 6). Since we cannot conveniently use a separator # that is not used for expressing Boolean assignments, the constructed string is more complicated in this reduction (see the full version of this paper [20] for full details).

5 Special Variants

In the last section of this paper, we consider two natural variants of our setting, and show how the particularities of these variants have a substantial impact on the complexity of the problems investigated above.

Gap Length Equalities. We investigate whether the polynomiality of the matching problem (see Section 3) is preserved under adding *gap length equalities* to the gap constraints, i. e., constraints of the form $|\mathsf{gap}_i| = |\mathsf{gap}_j|$ which are satisfied by an embedding e with respect to w if $|\mathsf{gap}_e(w,i)| = |\mathsf{gap}_e(w,j)|$. Our main motivation is that such length equality constraints (and more complex ones, e. g., described by linear inequalities like $2|\mathsf{gap}_7| + |\mathsf{gap}_3| \leq |\mathsf{gap}_2|$) are of interest in the theory of string solving [3]. Unfortunately, the matching problem becomes immediately NP-hard (the following result can be shown by adapting the NP-completeness proof for matching patterns with variables from [4]; see the the full version of this paper [20] for details).

▶ **Theorem 9.** *Match with length constraints and gap length equalities is* NP*-complete, even for binary alphabets and length constraints* $(0, +\infty)$.

Gap Constrained Subsequences With Multiplicities. With respect to the equivalence problem, we can show a positive result for the following modified setting. Let us consider the gc-subsequences of the sets $SubSeq(gc, w)$ with multiplicities. For example, for $w_1 = \mathtt{abba}$ and $w_2 = \mathtt{abab}$, we have $SubSeq(gc_2, w_1) = SubSeq(gc_2, w_2) = \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}\}$ with $gc_2 = (\Sigma^*)$. There is exactly one way of embedding $\mathtt{aa}$ and $\mathtt{bb}$ into both w_1 and w_2. On the other hand, $\mathtt{ab}$ can be embedded into w_1 in two different ways and into w_2 in three different ways. More precisely, the sets of gc_2-subsequences of w_1 and w_2 with multiplicities are $\{(\mathtt{aa}, 1), (\mathtt{ab}, 2), (\mathtt{ba}, 2), (\mathtt{bb}), 1\}$ and $\{(\mathtt{aa}, 1), (\mathtt{ab}, 3), (\mathtt{ba}, 1), (\mathtt{bb}), 1\}$, respectively.

Let us now formalise this setting. For strings u and v, and a $(|u|-1)$-tuple gc of gap constraints, we denote by $|u|_{v,gc}$ the number of distinct embeddings $e : |u| \to |v|$ that satisfy gc and $v \preceq_e u$. For example, $|\mathtt{bbaa}|_{\mathtt{ba},gc_2} = 4$, as $u[1]u[3] = u[2]u[3] = u[1]u[4] = u[2]u[4] = \mathtt{ba}$. For any $(k-1)$-tuple gc of gap constraints, we define the function $\Psi_{gc}(\cdot)\colon \Sigma^* \to \mathbb{N}^{(\Sigma^k)}$ by $\Psi_{gc}(w)[p] = |w|_{p,gc}$ for every $p \in \Sigma^k$. The *equivalence problem with multiplicities* is to decide, for a given $(k-1)$-tuple gc of gap constraints, and strings $w, w' \in \Sigma^*$, whether $\Psi_{gc}(w) = \Psi_{gc}(w')$. Note that for the case $gc = (\Sigma^*, \ldots, \Sigma^*)$ this is called the k-binomial equivalence, and was studied in the area of combinatorics on words (see, e. g., [55, 47, 48, 26]).

We show that equivalence with multiplicities can be decided in polynomial time (in contrast to the NP-completeness of the case without multiplicities).

▶ **Theorem 10.** *If $gc = (C_1, \ldots, C_{k-1})$ and C_i can be decided in polynomial time then the equivalence problem with multiplicities can be solved in polynomial time.*

Proof Sketch. We adapt the main idea from [26]. For a given string w and tuple gc of gap constraints, we can construct in polynomial time an NFA $A_{w,gc}$ with $L(A_{w,gc}) = SubSeq(gc, w)$. Moreover, for every $p \in SubSeq(gc, w)$, there is a one-to-one correspondence

between distinct embeddings e with $p \preceq_e w$ satisfying gc and accepting paths of $A_{w,gc}$ on input p. Consequently, we can decide $\Psi_{gc}(w) = \Psi_{gc}(w')$ by deciding the *path equivalence* of $A_{w,gc}$ and $A_{w',gc}$, which can be done in polynomial time by the algorithm from [64]. ◀

In the case of length, regular or reg-len constraints, the equivalence problem with multiplicities can be solved in $O(\max\{|w|, |w'|\}^4 k^4 + \mathsf{size}(gc))$ time. The *containment problem with multiplicities* (i. e., deciding $\Psi_{gc}(w)[p] \leq \Psi_{gc}(w')[p]$ for all $p \in \Sigma^k$) seems to be more difficult. To our knowledge, whether the case of classical subsequences (i. e., length constraints $(0, \infty)$) can be solved in polynomial time is open. On the other hand, for the case of length constraints $(0, 0)$ only (i. e., consecutive factors), or of length constraints (ℓ, ℓ) only (i. e., the case of the so called *partial words* [11, 18], where each gap has fixed length ℓ, and can be seen as a factor of length ℓ of wild cards, or don't care positions), showing polynomial time solvability is relatively simple.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014. doi:10.1007/978-3-662-43948-7_4.

3 Roberto Amadini. A survey on string constraint solving. *ACM Computing Surveys (CSUR)*, 55(1):1–38, 2021.

4 Dana Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21(1):46–62, 1980.

5 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex event recognition languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 7–10, 2017. doi:10.1145/3093742.3095106.

6 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.

7 Johannes Bader, Simon Gog, and Matthias Petri. Practical variable length gap pattern matching. In *Experimental Algorithms – 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, pages 1–16, 2016. doi:10.1007/978-3-319-38851-9_1.

8 Ricardo A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991.

9 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *ICALP*, volume 198 of *LIPIcs*, pages 119:1–119:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

10 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci.*, 443:25–34, 2012. doi:10.1016/j.tcs.2012.03.029.

11 Francine Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words*. Discrete mathematics and its applications. CRC Press, 2008.

12 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

13 Karl Bringmann. Fine-grained complexity theory (tutorial). In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 4:1–4:7, 2019. doi:10.4230/LIPIcs.STACS.2019.4.

14 Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, streaming, and fine-grained complexity of (weighted) LCS. In *Proc. FSTTCS 2018*, volume 122 of *LIPIcs*, pages 40:1–40:16, 2018.

15 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235, 2018.

16 Sam Buss and Michael Soltys. Unshuffling a square is NP-hard. *J. Comput. Syst. Sci.*, 80(4):766–776, 2014. doi:10.1016/j.jcss.2013.11.002.

17 Manuel Cáceres, Simon J. Puglisi, and Bella Zhukova. Fast indexes for gapped pattern matching. In *SOFSEM 2020: Theory and Practice of Computer Science – 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, pages 493–504, 2020. doi:10.1007/978-3-030-38919-2_40.

18 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007.

19 Maxime Crochemore, Borivoj Melichar, and Zdenek Tronícek. Directed acyclic subsequence graph – Overview. *J. Discrete Algorithms*, 1(3-4):255–280, 2003.

20 Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid. Subsequences with gap constraints: Complexity bounds for matching and analysis problems. *CoRR*, 2022. doi:10.48550/ARXIV.2206.13896.

21 Patrick Dinklage, Johannes Fischer, Alexander Herlez, Tomasz Kociumaka, and Florian Kurpicz. Practical Performance of Space Efficient Data Structures for Longest Common Extensions. In *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:20, 2020.

22 Bartlomiej Dudek, Pawel Gawrychowski, Garance Gourdel, and Tatiana Starikovskaya. Streaming regular expression membership and pattern matching. In *SODA*, pages 670–694. SIAM, 2022.

23 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Pattern matching with variables: Efficient algorithms and complexity results. *ACM Trans. Comput. Theory*, 12(1):6:1–6:37, 2020.

24 Lukas Fleischer and Manfred Kufleitner. Testing Simon's congruence. In *Proc. MFCS 2018*, volume 117 of *LIPIcs*, pages 62:1–62:13, 2018.

25 Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing k-binomial equivalence. In Multidisciplinary Creativity*, a collection of papers dedicated to G. Păun 65th birthday*, pages 239–248, 2015. available in CoRR abs/1509.00622.

26 Dominik D Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing k-binomial equivalence. *arXiv preprint arXiv:1509.00622*, 2015.

27 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *STACS*, volume 96 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

28 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *FSTTCS*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

29 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *ICALP*, volume 107 of *LIPIcs*, pages 127:1–127:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

30 Moses Ganardi, Danny Hucke, and Markus Lohrey. Sliding window algorithms for regular languages. In *LATA*, volume 10792 of *Lecture Notes in Computer Science*, pages 26–35. Springer, 2018.

31 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *ISAAC*, volume 149 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

32 Emmanuelle Garel. Minimal separators of two words. In *Proc. CPM 1993*, volume 684 of *Lecture Notes in Computer Science*, pages 35–53, 1993.

33 Pawel Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 34:1–34:18, 2021. doi:10.4230/LIPIcs.STACS.2021.34.

34 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020. doi:10.1007/s00778-019-00557-w.

35 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. LICS 2017*, pages 1–12, 2017.

36 Jean-Jacques Hebrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theor. Comput. Sci.*, 82(1):35–49, 22 May 1991.

37 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

38 Costas S. Iliopoulos, Marcin Kubica, M. Sohel Rahman, and Tomasz Walen. Algorithms for computing the longest parameterized common subsequence. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 265–273, 2007. doi:10.1007/978-3-540-73437-6_27.

39 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

40 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

41 Prateek Karandikar, Manfred Kufleitner, and Philippe Schnoebelen. On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.*, 115(4):515–519, 2015.

42 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In *Proc. CSL 2016*, volume 62 of *LIPIcs*, pages 37:1–37:22, 2016.

43 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Log. Methods Comput. Sci.*, 15(2), 2019.

44 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints. In *25th International Conference on Database Theory, ICDT 2022*, volume 220 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICDT.2022.18.

45 Dietrich Kuske. The subtrace order and counting first-order logic. In *Proc. CSR 2020*, volume 12159 of *Lecture Notes in Computer Science*, pages 289–302, 2020.

46 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In *Proc. FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 348–364, 2019.

47 Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the k-binomial complexity of the Thue-Morse word. In *Proc. DLT 2019*, volume 11647 of *Lecture Notes in Computer Science*, pages 278–291, 2019.

48 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *Electron. J. Combin.*, 24(1.44):36 pp., 2017.

49 Chun Li and Jianyong Wang. Efficiently mining closed subsequences with gap constraints. In *SDM*, pages 313–322. SIAM, 2008.

50 Chun Li, Qingyan Yang, Jianyong Wang, and Ming Li. Efficient mining of gap-constrained subsequences and its various applications. *ACM Trans. Knowl. Discov. Data*, 6(1):2:1–2:39, 2012.

51 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011. URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/92`.

52 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978.

53 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *J. Comput. Syst. Sci.*, 68(1):1–21, 2004.

54 William E. Riddle. An approach to software system modelling and analysis. *Comput. Lang.*, 4(1):49–66, 1979. `doi:10.1016/0096-0551(79)90009-2`.

55 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theor. Comput. Sci.*, 601:47–57, 2015.

56 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theoret. Comput. Sci.*, 340(2):188–203, 2005.

57 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theor. Comput. Sci.*, 418:116–120, 2012. `doi:10.1016/j.tcs.2011.10.017`.

58 Alan C. Shaw. Software descriptions with flow expressions. *IEEE Trans. Software Eng.*, 4(3):242–254, 1978. `doi:10.1109/TSE.1978.231501`.

59 Imre Simon. *Hierarchies of events with dot-depth one — Ph.D. thesis*. University of Waterloo, 1972.

60 Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222, 1975.

61 Imre Simon. Words distinguished by their subwords (extended abstract). In *Proc. WORDS 2003*, volume 27 of *TUCS General Publication*, pages 6–13, 2003.

62 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. `doi:10.1145/363347.363387`.

63 Zdenek Tronícek. Common subsequence automaton. In *Proc. CIAA 2002 (Revised Papers)*, volume 2608 of *Lecture Notes in Computer Science*, pages 270–275, 2002.

64 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, 1992. `doi:10.1137/0221017`.

65 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015. `doi:10.4230/LIPIcs.IPEC.2015.17`.

66 Georg Zetzsche. The complexity of downward closure comparisons. In *Proc. ICALP 2016*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016.

67 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 217–228, 2014. `doi:10.1145/2588555.2593671`.

Succinct List Indexing in Optimal Time

William L. Holland ✉
School of Computing and Information Systems, The University of Melbourne, Parkville, Australia

Abstract

An *indexed* list supports (efficient) access to both the offsets and the items of an arbitrarily ordered set under the effect of *insertions* and *deletions*. Existing solutions are engaged in a space-time trade-off. On the one hand, time efficient solutions are composed as a package of data structures: a linked-list, a hash table and a tree-type structure to support indexing. This arrangement observes a memory commitment that is outside the information theoretic lower bound (for ordered sets) by a factor of 12. On the other hand, the memory lower bound can be satisfied, up to an additive lower order term, trivially with an array. However, operations incur time costs proportional to the length of the array.

We revisit the list indexing problem by attempting to balance the competing demands of space and time efficiency. We prepare the first *succinct* indexed list that supports efficient query and update operations. To implement an ordered set of size n, drawn from the universe $\{1, \ldots, m\}$, the solution occupies $n(\log m + o(\log n))$ bits (with high probability) and admits all operations *optimally* in $\mathcal{O}(\log n / \log \log n)$ time.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Succinct Data Structures, Lists, Dynamic Data Structures

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.65

Funding This work was supported by an Australian Government Research Training Program RTP Scholarship.

Acknowledgements We acknowledge the Wurundjeri People of the Kulin Nations as traditional owners of the land on which we live and work.

1 Introduction

List indexing is a popular problem in the data structures literature from the previous century. The problem demands a representation of a list $\mathcal{L}$ that supports the following operations:

- `insert`$(\mathcal{L}, x, y)$: insert element y at the index succeeding element x.
- `delete`$(\mathcal{L}, x)$: delete element x from the list.
- `index`$(\mathcal{L}, i)$: return the element at index i in the list.
- `position`$(\mathcal{L}, x)$: return the index of element x in the list.

A list that supports these operations[1] is named an *indexed list*. Notably, the `position` query is a key inbuilt function in programming languages such as Python and Java. For a list encoding an arbitrarily *ordered* set $\mathcal{L} \subseteq [m]$ of size n, a lower bound of (amortized) $\Omega(\log n / \log \log n)$ time per operation is due to Fredman and Saks [4]. The bound is assembled in the cell probe model. Matching upper bounds, in the word RAM model, are provided by Dietz [3]. As is noted by Andersson [1], the solution requires item pointers be provided as arguments to the operations. Thus, the data structure of Dietz appears in three components; a hash-table to retrieve item pointers; the underlying linked-list; and the structure to support (fast) indexing. This configuration requires $\mathcal{O}(n \log m)$ bits of space with a constant factor around 12.

[1] We assume that the update sequence does not contain any repetitions. We elaborate on the implication of including repetitions in Appendix A.

A data structure, supporting a particular query (or set of queries), is *succinct* if it accommodates a memory commitment "close to" the information-theoretic lower bound *and* admits the prescribed query operation(s) "efficiently". Formally, if a minimum of $\mathcal{B}$ bits are required, in the information-theoretic sense, to store the data, a succinct data structure occupies $\mathcal{B} + o(\mathcal{B})$ bits of memory. Despite the affluent and diverse state of the field of succinct data structures [5, 9, 11, 14, 18], a succinct representation of an indexed list does not exist and we pursue the problem of finding such a representation.

The information-theoretic lower bound for encoding an ordered set is

$$B(m, n) = \log\left(\frac{m!}{(m-n)!}\right) = n(\log m - \mathcal{O}(1)) \tag{1}$$

bits. A low-memory solution can be constructed by positioning the items consecutively in an array that has $\log m$ bit cells. The per-item cost of the encoding is $\mathcal{O}(1)$ bits above the information-theoretic lower bound. However, updates to the list and `position` queries are slow, requiring $\mathcal{O}(n)$ time. This observation leads to an interesting question: can a list be encoded at close to $\log m$ bits per item and support dynamic indexing operations efficiently?

In response to this question, we first present a simple data structure, named the Princess List (PL), that significantly improves the update and query efficiency of the canonical linked-list at a small space overhead. We then detail an optimized implementation of the Princess List (named PL+) that occupies $n(\log m + o(\log n))$ bits of space, with high probability, and admits *optimal* query and update times, matching the prior state-of-the-art. The structure pays, per-item, a sublogarithmic number of bits above the information-theoretic lower bound for ordered sets. In the appendix we demonstrate an alternate solution that achieves the memory commitment with probability 1, but with update times close to optimal with high probability.

The Princess List

At a high level, the PL is a divide-and-conquer approach, where the list items are partitioned, via a hash function, into a collection of disjoint sublists. If the hash function that performs the partition maps on to a small range $[\sigma] = \{0, 1, \ldots, \sigma - 1\}$, the list can be mapped to a random *string* on the alphabet $[\sigma]$. Each character $c \in [\sigma]$ represents an order-preserved sublist $\mathcal{L}_c$ on the set $S_c = \{x \in \mathcal{L} \mid h(x) = c\}$ and each occurrence of a character represents a unique item. The PL is comprised of the random string $h(\mathcal{L})$ and the sublists $\mathcal{L}_c$.

An operation entails identifying, through the random string, the correct (and unique) sublist to update or query, followed by an execution of the operation on the small sublist. The random string maintains the inter-order between the sublists and, consequently, allows us to relate each subproblem to the superproblem. Thus, the dynamic string negotiates the divide-and-conquer strategy and, simultaneously, maintains the order of the full list. In other words, the string articulates both how we branch into subproblems and how we merge the sublists back together. With this set-up, queries are fast, comprising a constant number of queries on the string and a linear scan on a subproblem.

The high level structure, composed of the random string and the sublists, introduces three key challenges. First, as each item is represented by both a string character and an item identifier in the sublist, the representation of an item in each component must be concise. Item identifiers are expensive, requiring $\log m$ bits, and we exploit a random permutation of the universe to partition each identifier into both its hash character and a small unique identifier in the assigned sublist. This is a technique known as the quotient filter [17]. Consequently, the total information needed for both the reduced small-alphabet

string and the sublists is concise. Second, we need to maintain all the component data structures in a compact form. To store the small alphabet string we utilize an existing solution for succinct dynamic strings [15]. To reduce the size of the bit allocation for the sublists, we use a linked-list representation with $\omega(1)$ items packed into each node. This allows the per-item cost of the pointers to be sublogarithmic. Third, as we are required to perform linear scans on the sublists, we don't want the cardinality of any sublist to grow too large. To mitigate the impact of a large sublist, we introduce a threshold and store any sublist that exceeds the threshold under the non-succinct but time-optimal solution of Dietz [3].

1.1 Contribution and outline

We present the first succinct indexed list with update time and query time performance identical to the state-of-the-art. We first introduce a simple and novel solution to list indexing (the PL) that lowers the traversal cost on a linked-list (§3). We then curate an optimized instance of our simple solution (the PL+) that achieves the attributes available in the following theorem (§4).

▶ **Theorem 1.** *For any constant $\gamma > 0$, an ordered set of n items, drawn from the universe $[m]$, where $m = \text{poly}(n)$, can be stored in $n(\log m + o(\log n))$ bits with probability $1 - \mathcal{O}(1/n^{\gamma})$ and support* `index` *and* `position` *in $\mathcal{O}(\log n / \log\log n)$ time and* `insert` *and* `delete` *in $\mathcal{O}(\log n / \log\log n)$ amortized time.*

This constitutes our main result and contribution. The significance here is that our data structure performs all operations optimal in run time – equal to the prior state-of-the-art – while spending close to $\log m$ bits per item. As a comparison, we achieve better asymptotic performance than balanced binary search trees for updates and access queries with additional support for indexing on ordered sets *and* a succinct representation (over ordered sets) with high probability. In addition, to obtain a succinct representation with probability one, we demonstrate a modified construction (detailed in the appendix) that secures the following properties.

▶ **Theorem 2.** *For any constant $\gamma > 0$, an ordered set of n items, drawn from the universe $[m]$, where $m = \text{poly}(n)$, can be stored in $n(\log m + o(\log n))$ bits and support* `index` *in $\mathcal{O}(\log n / \log\log n)$ time,* `position` *in $\mathcal{O}(\log n)$ time with probability $1 - \mathcal{O}(1/n^{c})$ and any sequence of $\mathcal{O}(n)$ updates (*`insert` *or* `delete`*) takes $\mathcal{O}(n \log n)$ time with probability $1 - \mathcal{O}(1/n^{\gamma})$.*

One constraint for the data structure is that the universe size is a polynomial in the problem size. Therefore, as currently stated, our result will not hold on small problem instances. However, as small problem sizes are less interesting, this is only a minor concern . In these instances we can afford to store a non-optimal representation and, indeed, this is likely preferable. The constraint is a condition of our hash family [20]. Other succinct data structures, such as Backyard Cuckoo Hashing [2], which utilize the latter result, also inherit this condition.

1.2 The rank-select problem

List indexing has proximity to the well known *rank-select problem* [6, 7]. For the latter, a solution constitutes a succinct representation of a sequence C (where repetitions are allowed), drawn from a universe (or alphabet) Σ, that supports the following operations.

- $C[i]$: return the character at index i.
- $\texttt{rank}_b(C, i)$: given $i \in \{0, \ldots, n-1\}$ and $b \in \Sigma$ return $|\{j \in \{0, \ldots, i\} \mid C[j] = b\}|$, *i.e.*, the number of occurrences of character b in the subsequence $C[0 \ldots i]$.
- $\texttt{select}_b(C, j)$: given $j \in \{0, \ldots, n-1\}$ and $b \in \Sigma$ return $\min\{x \mid \texttt{rank}_b(C, x) = j\}$, *i.e.*, the index of the j^{th} occurrence of b in C.

The `select` operation is equivalent to the `position` query and an access $C[i]$ is exactly the `index` query. Thus, a compressed rank-select data structure for dynamic sequences [10, 13, 15, 16] would act as a solution to the list indexing problem. The two problems are very close and, indeed, the rank-select problem inherits the time lower bounds of the list indexing problem. The main difference between the two problems is the assumption about the universe size. For the rank-select problem it is reasonable to focus on texts drawn from a small universe of characters. However, the solutions become intractable for problems on larger universes and, in particular, where repetitions do not occur. For example, with $m = |\Sigma|$, a state-of-the-art solution by Navarro and Nekrich has a memory allocation with a redundancy term of $\mathcal{O}(m \log n)$ [15]. In the list indexing problem, where $m \geq n$, this leads to a non-succinct representation. Further, a solution by Munro and Nekrich [13] that supports "arbitrarily large alphabets" meets a similar fate. While the memory allocation is quoted as $H_k(C) + o(n \log m)$ bits[2], it appears to require (implicitly) that $m \leq n$. The issue is that an auxiliary data structure is stored for each character in Σ. Resolving this issue is not as simple as assigning 0 bits for non-occurring characters as this introduces the need for a search structure. Even a state-of-the-art dynamic succinct dictionary [2] would push the memory allocation over a succinct allowance. A succinct indexed list remedies this restriction on the universe size and is an open problem that we address here.

To help distinguish our problem from this influential strain of prior work, it is best to think of an indexed list as a dictionary. In this context, it would make little sense to use a compressed sequence as a solution to the problem. Before we progress with the exposition, we introduce some background around strings, the hashing scheme we engage and the existing solutions to list indexing

2 Background

We proceed in the (unit cost) word RAM model of computation, with word size $w = \Theta(\log m)$. Consequently, items from the universe can be stored in $\mathcal{O}(1)$ machine words and bitwise operations on words can be performed in constant time. We use the superscript notation $\texttt{index}^M$, $\texttt{position}^M$, $\texttt{insert}^M$, $\texttt{delete}^M$ to refer (unambiguously) to query and update algorithms on a list under the representation of the data structure M. We drop the superscript M and the argument $\mathcal{L}$ when both are obvious from the context.

2.1 Strings

In addition to the `rank` and `select` queries, a dynamic string supports the following update operations:

- $\texttt{insertSt}(C, b, i)$: insert character $b \in \Sigma$ at index i in string C.
- $\texttt{deleteSt}(C, i)$: delete character at index i in string C.

[2] $H_k(S)$ is the k^{th} order empirical entropy of the string S. This term is somewhat redundant in our setting; as repetitions are not allowed, the empirical entropy is high.

The fundamental dynamic string implementation is the wavelet tree. It emerged in the context of text compression [8], as a tool for compressing suffix arrays, and has since seen application in a diverse range of problems. The power of the wavelet tree rests in its capacity to support both the compression of strings and fast query and update operations. The state-of-the-art wavelet tree is advanced by Navarro and Nekrich and acknowledges the following properties.

▶ **Lemma 3** ([15]). *For $\varphi \in (0,1)$, a dynamic string of length n on an alphabet of size σ can be stored in $n\log\sigma + \mathcal{O}(n\log\sigma/\log^{1-\varphi} n) + \mathcal{O}(\sigma\log n)$ bits and support the queries* `rank` *and* `select` *in $\mathcal{O}(\varphi^{-2}\log n/\log\log n)$ time and* `insertSt` *and* `deleteSt` *in $\mathcal{O}(\varphi^{-2}\log n/\log\log n)$ amortized time.*

2.2 k-wise independent hashing

The division of the list into disjoint sublists is coordinated by a random source. Our compact representation utilizes existing hash families with *limited* independence, small description and constant time evaluation.

A family of functions is k-wise independent if, for a function $f[m] \to [r]$ selected uniformly at random from the family, the image of any k-tuple, $(f(x_1), \ldots, f(x_k))$, is uniformly distributed in $[r]^k$. There exists no family of k-wise independent hash functions that meets our requirements – that is, small description and constant time evaluation. However, the construction of Siegel [20] is a good approximation and is sufficient for our purposes.

▶ **Theorem 4** ([20]). *Let $S \subseteq U = [m]$ be a set of $n = k^{\mathcal{O}(1)}$ elements. For any constants $\varepsilon, c > 0$ there is a RAM algorithm constructing a random family $\mathcal{H}$ of functions in $o(n)$ time (provided $m = \text{poly}(n)$) and $o(k)$ words of space, such that:*

- *with probability $1 - \mathcal{O}(1/n^c)$, $\mathcal{H}$ is k-wise independent;*
- *there is a RAM data structure of $\mathcal{O}(k^{1+\varepsilon})$ words representing its functions such that a function can be evaluated in constant time. The data structure can be initialized to a random function in $\mathcal{O}(k^{1+\varepsilon})$ time.*

Significantly, the tail probabilities of k-wise independent random variables can be bound with a Chernoff-like result. The result we employ comes from Schmidt *et. al* [19].

▶ **Theorem 5** ([19] Theorem 5.III.b). *If X is the sum of k-wise independent random variables, each confined to the interval $[0,1]$, with $\mu = \mathrm{E}[X]$ and $k = \lfloor \delta\mu e^{-1/3} \rfloor$, then $\Pr[|X - \mu| \geq \delta\mu] \leq e^{-\delta\mu/3}$.*

Negatively related random variables

Furthermore, in our analysis, we encounter instances of sums of indicator random variables that are not k-wise independent, but do hold the property of being *negatively related*. Happily, Jansen established that Chernoff bounds apply to sums of negatively related random variables [12] and we utilize this result in our analysis.

2.3 Prior work

The literature on list indexing assumes that item pointers are readily available (doubtless through a hash table) as arguments to the operations. A non-optimal or naive solution entails a straightforward application of a balanced (or height-bounded) binary tree. List items are stored, in order from left to right, in the leaves of the tree and internal nodes store a count of the number of leaves (or, the size of the sublist) in the subtree rooted at the internal node. An `index`(i) query begins at the root and, following the logic of the counts stored at internal

nodes, can branch into the sublist containing the correct index. Conversely, `position`(x) begins at a leaf and accumulates a global index on a leaf-to-root path; the count of any left sibling at an internal node gets aggregated to the current index. Updates require recourse to a balancing criteria and may entail some restructuring. All operations take $\mathcal{O}(\log n)$ time (worst-case or amortized depending on the choice of tree) and the tree requires $\mathcal{O}(n \log m)$ bits to store.

This idea was extended by Dietz with what are now fairly standard techniques [3]. The binary tree is replaced with a weight balanced B-tree with branching factor $\Theta(\log^{\varphi} m)$, for $\varphi \in (0, 1)$. As before, internal nodes count the number of leaves in the subtree they root. On a leaf-to-root path, a `position` query accumulates the counts, at each internal node, of all *left* siblings. To remove dependence on the branching factor, the sum of the counts of left siblings is evaluated on a *partial sums* data structure. The latter stores an array of integers $A[1 \dots b]$ and admits the following procedures.

- `add`(i, δ): perform $A[i] \leftarrow A[i] + \delta$, where $\delta = \log^{\mathcal{O}(1)} m$.
- `sum`(j): return $\sum_{i \leq j} A[i]$.

On problem size $b = \mathcal{O}(\log^{\varphi} m)$, both operations can be performed in $\mathcal{O}(1)$ amortized time. Thus, the cost of a `position` query is bound by the height ($\mathcal{O}(\log_{\log^{\varphi} m} n)$) of the tree. Efficient navigation for `index`(i) queries is provided by the additional operation:

- `select_ps`(i): return the smallest j such that `sum`$(j) \geq i$.

Although the partial sums structure of Dietz does not explicitly solve the abstraction of `select` queries of prefix sums[3], a constant time solution, on small problem sizes, is provided by Raman *et al.* [18]. Thus, the cost of both query operations is bound by the height of the tree and is thereby optimal at $\mathcal{O}(\log n / \log \log n)$. Due to rebuilding requirements on both the partial sums structure and the weight balanced B-tree, update costs are amortized. The solution requires $\mathcal{O}(n \log m)$ bits.

▶ **Lemma 6** ([3]). *The list indexing problem can be solved by a data structure, occupying $\mathcal{O}(n \log m)$ bits, that supports* `index` *and* `position` *in $\mathcal{O}(\log n / \log \log n)$ time and* `insert` *and* `delete` *in $\mathcal{O}(\log n / \log \log n)$ amortized time.*

Andersson and Petersson define an approximate version of the problem [1]. The `position` query is permitted to err with a user defined relative error and the `index` query can retrieve *any* item from a neighborhood, of size proportional to the relative error, surrounding the correct index. The added flexibility allows the authors to remove the dependence on the problem size from the update and query costs. For relative error parameter $\varepsilon \in (0, 1)$, queries can be evaluated in constant worst-case time and updates in amortized $\mathcal{O}(\varepsilon^{-2})$.

3 The Princess List: a simple solution to list indexing

The linked-list, obliged to traverse all preceding nodes in the ordered set, performs the operation `index`(i) in $\mathcal{O}(i)$ time. To reduce this expense, the PL employs a divide-and-conquer strategy that partitions the list into a collection of disjoint sublists. A search proceeds on a single sublist and thereby reduces the expense of the linear scan. To allocate items to sublists, we map the list, via a hash function h, onto a small alphabet $\Sigma = [\sigma]$ and represent the mapped list

$$h(\mathcal{L}) \cong \langle h(\mathcal{L}[0]), \dots, h(\mathcal{L}[|\mathcal{L}| - 1]) \rangle$$

[3] They augment the weight balanced B-tree with additional pointers that implicitly implement the functionality.

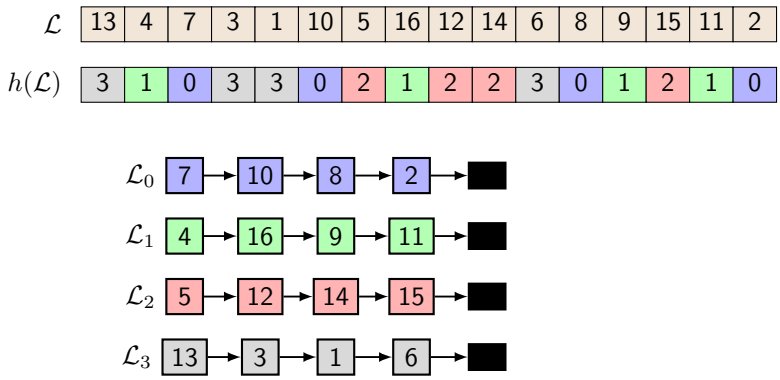


Figure 1 The Princess List. The list $\mathcal{L}$ of items from $U \cong \{1, \ldots, 16\}$ is hashed, via the function $h : U \to \{0, \ldots, 3\}$, into the string $h(\mathcal{L})$. Items mapping to character c are stored in the (linear) sublist $\mathcal{L}_c$. To evaluate $\texttt{index}(9)$: identify $h(\mathcal{L})[9] = 2$; calculate $\texttt{rank}_2(h(\mathcal{L}), 9) = 2$; and perform $\texttt{index}(\mathcal{L}_2, 2) = 14$ with a linear scan.

with a dynamic string. Under this regime, multiple items may map to the same character and we store such a set of items under a (order-preserved) list representation. The random string $h(\mathcal{L})$ organises the branching into subproblems and, simultaneously, preserves the order of the full list. The range of the hash function permits control over the size of the sublists. An image of the PL arrangement, which outlines, at a high level, the division of the problem into subproblems and the support of the string $h(\mathcal{L})$ towards managing the relation between subproblems, is available in Figure 1.

Fixing notation, let M denote the list data structure used to implement the sublists $\mathcal{L}_\Sigma$ and $\mathsf{PL}^M(\mathcal{L}, \sigma)$ name the subsequent Princess List representation. To evaluate $\texttt{index}(i)$, the program begins by identifying the relevant sublist through the operation $c = h(\mathcal{L})[i]$. Therefore, the item at location i in $\mathcal{L}$ belongs to sublist $\mathcal{L}_c$. Subsequently, the global location i needs to be translated to a local location on the sublist. This is achieved by a `rank` query; $i' = \texttt{rank}_c(h(\mathcal{L}), i) - 1$ denotes the position of $\mathcal{L}[i]$ in the sublist $\mathcal{L}_c$. The query is completed by the operation $\texttt{index}^M(\mathcal{L}_c, i')$, which, under a linked-list representation, stipulates a linear scan and requires $\mathcal{O}(i')$ time. Conversely, to evaluate $\texttt{position}(x)$, the program starts at the sublist $\mathcal{L}_{h(x)}$. The value $j = \texttt{position}^M(\mathcal{L}_{h(x)}, x) + 1$ provides the rank of the character occurrence $h(x)$ that corresponds to the item x. The query is completed by determining the global index of the j^{th} occurrence of $h(x)$ in $h(\mathcal{L})$ through the query $\texttt{position}(x) = \texttt{select}_{h(x)}(h(\mathcal{L}), j)$. Pseudo-code for the query operations is available in Figure 2.

The update $\texttt{insert}(x, y)$ begins by identifying the index of insertion in both the string $i = \texttt{position}^{\mathsf{PL}}(\mathcal{L}, x)$ and the relevant subproblem $i' = \texttt{rank}_{h(y)}(h(\mathcal{L}), i)$. Character $h(y)$ is then inserted at index $i + 1$ in the dynamic string and item y is placed after item $\texttt{position}^M(\mathcal{L}_{h(y)}, i')$ in sublist $\mathcal{L}_{h(y)}$. The $\texttt{delete}(x)$ operation proceeds in a similar fashion; it locates the index of the placeholder of x in $h(\mathcal{L})$ and identifies the relevant subproblem $c = h(x)$. The character at this index is then removed from $h(\mathcal{L})$ and x is removed from $\mathcal{L}_c$. Pseudo-code for the update operations is available in Figure 3.

The efficiencies of the operations depend on the efficiency of the dynamic string $h(\mathcal{L})$ and the size of the sublists. By Lemma 3, all string queries can be supported in $\mathcal{O}(\log n / \log\log n)$ time and string updates in $\mathcal{O}(\log n / \log\log n)$ amortized time. Each sublist has expected size n/σ. The amount by which sublist sizes deviate from their expectation depends on both σ and the randomness of the underlying hashing scheme.

Procedure $\texttt{index}^{\mathsf{PL}}(i)$
- $c \leftarrow h(\mathcal{L})[i]$
- $i' \leftarrow \texttt{rank}_c(h(\mathcal{L}), i) - 1$
- **return** $\texttt{index}^{M}(\mathcal{L}_c, i')$

Procedure $\texttt{position}^{\mathsf{PL}}(x)$
- $c \leftarrow h(x)$
- $j \leftarrow \texttt{position}^{M}(\mathcal{L}_c, x) + 1$
- **return** $\texttt{select}_c(h(\mathcal{L}), j)$

Figure 2 Query operations for the $\mathsf{PL}^{M}(\mathcal{L}, \sigma)$ representation of the list $\mathcal{L}$. The data structure M is used to implement the sublists.

Procedure $\texttt{delete}^{\mathsf{PL}}(x)$
- $i \leftarrow \texttt{position}^{\mathsf{PL}}(x)$
- $c \leftarrow h(x)$
- $\texttt{delete}^{M}(\mathcal{L}_c, x)$
- $\texttt{deleteSt}(h(\mathcal{L}), i)$
- **return**

Procedure $\texttt{insert}^{\mathsf{PL}}(x, y)$
- $i \leftarrow \texttt{position}^{\mathsf{PL}}(x)$
- $c \leftarrow h(y)$
- $i' \leftarrow \texttt{rank}_c(h(\mathcal{L}), i)$
- $z \leftarrow \texttt{position}^{M}(\mathcal{L}_c, i')$
- $\texttt{insert}^{M}(\mathcal{L}_c, z, y)$
- $\texttt{insertSt}(h(\mathcal{L}), c, i + 1)$
- **return**

Figure 3 Update operations for the $\mathsf{PL}^{M}(\mathcal{L}, \sigma)$ representation of the list $\mathcal{L}$.

3.1 Towards a compact representation

In itself, the PL does not articulate a succinct form. Thus, before moving to the main result, we need to construct the bridge between the general form and an optimal encoding. This is achieved in three parts. First, we need to paramteterize the PL by choosing an alphabet size. The verdict on the parameter choice is provided by the $\Omega(\log n / \log\log n)$ time lower bound for the operations of an indexed list [4]. We require *linear* scans when operating on sublists. Thus, it is desirable that subproblems have size $\mathcal{O}(\log n / \log\log n)$. As sublists have expected cardinality n/σ, the latter implies an alphabet size of

$$\sigma = \Omega\left(n \cdot \frac{\log\log n}{\log n}\right). \tag{2}$$

Second, when following a standard linked-list implementation, the memory allocation of the subproblems trespasses over $n(\log m + o(\log n))$ bits. Therefore, an alternate list representation is required at the sublists. Multiple directions are available and we opt for packing a super-constant number of items in the nodes of a linked-list to reduce the per-item cost of the pointers. Third, in light of the space allocated to the dynamic string – which is $\omega(1)$ bits per character by equation (2) – we cannot afford to store the full key of each item. The key length can be reduced with a technique, standard in succinct dictionaries [17], named the quotient filter. A random permutation $\pi : [m] \to [m]$ is employed. For item x, the leftmost $\log\sigma$ bits of $\pi(x)$ specify the subproblem to which x is a member and the rightmost $(\log m - \log\sigma)$ bits of $\pi(x)$ are stored in the list. In this manner, the representation of the item is split between a character in the dynamic string *and* an identifier in the allocated sublist. Permutations are necessary to avoid collisions in the sublists. The remainder of the paper illustrates the details of the bridge outlined above. We refer to this construction, an optimal instance of the PL, as PL+.

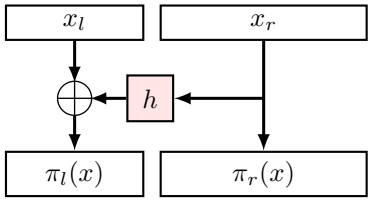


Figure 4 One-round Feistal permutation [2].

4 PL+: state-of-the-art wavelet trees and packed linked-lists

Items are allocated to sublists with Siegel's hash family [20]. Therefore, by Theorem 4, with probability $1 - \mathcal{O}(1/n^{\gamma})$, item allocations are k-wise independent, for $k^{\mathcal{O}(1)} = n$ and any constant $\gamma > 0$. Further, the hash function is evaluated in constant time and is stored in $o(n)$ bits. For our construction, we require that $\log n = \Theta(\log m) = \Theta(w)$. In other words, it is assumed that the universe size is a polynomial in the problem size.

Our structure, PL+, is an instantiation of the $\mathsf{PL}^{\mathsf{PackUnpack}}(\mathcal{L}, \Theta(n \log\log n / \log n))$ form, where PackUnpack refers to a type of linked-list that we detail below. To implement the dynamic string, PL+ appoints the wavelet tree of Navarro and Nekrich [15]. As the ideal alphabet size changes with n, the structure is periodically rebuilt to conform with the configuration of asymptotic constaints of the alphabet. We set σ to a power-of-two (for efficiency) such that

$$\sigma \in \left[\frac{1}{2}n \cdot \frac{\log\log n}{\log n}, 2n \cdot \frac{\log\log n}{\log n}\right] \tag{3}$$

Therefore, a rebuild happens, in the worst-case, every $\Omega(n)$ updates. A simple rebuild can be performed by constructing a new string under the updated alphabet. This takes $\mathcal{O}(n \log n / \log\log n)$ time by Lemma 3. In addition, selecting a new hash function from the constructed hash family takes $o(n)$ time. Periodically, but not with every rebuild, we will also have to reconstruct the hash family. The latter occurrence takes $o(n)$ time. Thus, rebuilding contributes $\mathcal{O}(\log n / \log\log n)$ amortized time to each update.

4.1 A random permutation

Without loss of generality, and for ease of demonstration, we assume that m is a power-of-two. The permutation is generated from a collection of one-round Feistal permutations. We use them in a manner similar to Arbitman *et al.* [2], where existing hash functions, and the properties they inhabit, are recruited to generate the randomness. Let $\mathcal{H}_\sigma$ be a class of $h : [m/\sigma] \to [\sigma]$ functions. Let x_l denote the leftmost $\log\sigma$ bits of a key x and x_r denote the rightmost $\log(m/\sigma)$ bits. A permutation $\pi_h : [m] \to [m]$ is defined for each instance $h \in \mathcal{H}_\sigma$:

$$\pi_h(x) = (\pi_l(x), \pi_r(x)) = (x_l \oplus h(x_r), x_r).$$

This is a permutation as every pair of keys x and y, where $x_r = y_r$, receive the same hash value, but map to a different bucket under the $\oplus$ operation. An image of the permutation is provided in Figure 4. With the permutation π, an item x is allocated to a subproblem with the random character $c = x_l \oplus h(x_r)$ and is stored under the representation x_r in $\mathcal{L}_c$. In this manner, items are allocated to subproblems according to the randomness prescribed by our hashing scheme.

4.2 The sublists

From Lemma 3 (with the parameter φ set to some constant), the dynamic string $h(\mathcal{L})$ occupies

$$n \log \sigma + o(n \log n) + \mathcal{O}(\sigma \log n) = n(\log \sigma + o(\log n)) \tag{4}$$

bits. Thus, for a list implementation that is asymptotically close to $n \log m$ bits, the remaining budget available to the sublists is $n(\log m - \log \sigma + o(\log n))$. As stated above, an item is stored in sublist $\mathcal{L}_{\pi_l(x)}$ with the $(\log m - \log \sigma)$ (permuted) bits of $\pi_r(x)$. This representation is in line with the budget stated above. It remains to demonstrate how the collection of sublists can be stored efficiently, with low redundancy.

A sublist is stored in a *packed* linked-list. A node in the linked-list contains at most $\log \log n$ items stored in a packed array. If the keys are represented in K bits, the contents of each node occupy $K \cdot \log \log n$ bits. Only the last node in the list is permitted to have less than $\log \log n$ items. Consequently, the nodes are packed as tight as possible. Updates and queries can be performed on linear traversals. Note that the update operations require the shifting of items prior to insertion and after deletion.

As we perform linear scans (in the tradition of the linked-list), the runtime of operations depends on the cardinality of the sublists. Thus, for optimality, sublist cardinalities cannot exceed $\mathcal{O}(\log n / \log \log n)$. Unfortunately, in all likelihood, some sublists will contain $\omega(\log n / \log \log n)$ items. To mitigate this outcome, we track the cardinalities of each sublist and, for a large enough constant $C > 0$, if a sublist cardinality exceeds $C \log n / \log \log n$, we implement the sublist with an *uncompressed* solution. The latter could be the time optimal solution put forth by Dietz [3] (see Lemma 6). With this set-up all operations require $\mathcal{O}(\log n / \log \log n)$ time to complete. We refer to this list data structure, which resorts to an uncompressed solution when its cardinality exceeds a specified threshold $C \cdot \log n / \log \log n$, as a pack-unpack list with threshold parameter C ($\mathsf{PackUnpack}(C)$).

4.3 Memory allocation for sublists

The proofs for the remainder of this section are located in Appendix B.2. The key with PL+ is to ensure that the number of items that belong to uncompressed sublists does not grow too large. To provide an upper bound on the latter, we compute both the maximum cardinality of the sublists and bound the number of sublists with cardinalities that exceed our threshold. We begin with the former. The upcoming pair of lemmas rely on the properties of k-wise independence. Recall that the construction of the hash family fails with probability $\mathcal{O}(1/n^\gamma)$, for any constant $\gamma > 0$, and the analysis of the probabilities must account for this occurrence. We use the notation $\xi_\gamma = \mathcal{O}(1/n^\gamma)$ to refer to this error term incurred by our choice of hash family.

▶ **Lemma 7.** *For $\gamma > 0$,*

$$\max_{c \in [\sigma]} |\mathcal{L}_c| \leq C_1 \log n \tag{5}$$

with probability at least $1 - e^{-\mathcal{O}(C_1) \log n} - \xi_\gamma$.

The proof is a standard rehearsal of bounding the maximum load in a balls-in-bins problem with a Chernoff bound. We now bound the number of sublists that exceed the threshold.

▶ **Lemma 8.** *For $C_1 = \mathcal{O}(\log n)$, C set to a sufficiently large constant and any $\gamma > 0$,*

$$|\{c \mid |\mathcal{L}_c| \geq C \cdot \log n / \log\log n\}| \leq \frac{n}{C_1 \log n \cdot \log\log n} \tag{6}$$

with probability at least $1 - e^{-n/(\mathcal{O}(C_1)\log n \log\log n)} - \xi_\gamma$.

Let Q_C be the set of items that belong to unpacked lists when using threshold parameter C. The cardinality $|Q_C|$ is less than the product of the max load and the number of sublist cardinalities that exceed the threshold. Therefore, with the combination of Lemmas 7 and 8, and an appropriate choice of C_1, we arrive at the following result.

▶ **Lemma 9.** *For a sufficiently large constant C and $\gamma > 0$, $|Q_C| \leq n/\log\log n$ with probability at least $1 - \xi_\gamma$.*

Each item in an unpacked list requires $\mathcal{O}(\log n)$ bits to store. Therefore, this result implies that, with high probability, items belonging to unpacked lists occupy a total of $\mathcal{O}(n \log n/\log\log n)$ bits. This allocation is dominated by the space occupied by the packed lists and leads to the following result.

▶ **Lemma 10.** *For any $\gamma > 0$, if each item occupies K bits, the memory allocation of the sublists, with sufficiently large threshold parameter C, aggregates to $n(K + o(\log n)) + o(n)K$ bits with probability at least $1 - \mathcal{O}(1/n^\gamma)$.*

5 Space and time efficiency of PL+

The memory allocation for PL+ is determined by the implementations of the string $h(\mathcal{L})$ and the sublists $\{\mathcal{L}_c\}$. Therefore, with the combination of Lemma 3 and Lemma 10, we arrive at a bound on the memory allocation of PL+.

▶ **Lemma 11.** *For any $\gamma > 0$, for $m = \text{poly}(n)$, to store a list of n items, PL+ occupies $n(\log m + o(\log n))$ bits of space with probability at least $1 - \mathcal{O}(1/n^\gamma)$.*

As the alphabet size is a function of the number of items, PL+ needs to be rebuilt periodically. By relation (3), this occurs every $\Omega(n)$ updates. The change in alphabet also affects the permutation and we periodically require a *different* initialization of the hash family to support the scheme. As discussed above, the string can be rebuilt in $\mathcal{O}(n \log n/\log\log n)$ time. For a crude bound – which assumes that all items are uncompressed – the sublists can be rebuilt in $\mathcal{O}(n \log n/\log\log n)$ time. Therefore, the overall cost of the rebuild is $\mathcal{O}(\log n/\log\log n)$ amortized time per update.

▶ **Lemma 12.** *On a list of n items, PL+ performs* `index` *and* `position` *in $\mathcal{O}(\log n/\log\log n)$ time and updates* (`insert` *or* `delete`) *complete in $\mathcal{O}(\log n/\log\log n)$ amortized time.*

Combined, Lemmas 11 and 12 curate our main result, which we restate below.

▶ **Theorem 1.** *For any constant $\gamma > 0$, an ordered set of n items, drawn from the universe $[m]$, where $m = \text{poly}(n)$, can be stored in $n(\log m + o(\log n))$ bits with probability $1 - \mathcal{O}(1/n^\gamma)$ and support* `index` *and* `position` *in $\mathcal{O}(\log n/\log\log n)$ time and* `insert` *and* `delete` *in $\mathcal{O}(\log n/\log\log n)$ amortized time.*

The error term in Theorem 1 is dominated by the likelihood that the hash family fails. On the condition that the construction of the random hash family is successful, the desired bit allocation holds with probability $1 - (1/n)^{\omega(1)}$. The fail rate applies to each update. Therefore our result holds, with high probability, on any sequence of operations that is polynomial in length. Notably, the structure incurs, per-item, a sublogarithmic number of bits above the information theoretic bound and, taken with the efficient query and update operations, constitutes a *succinct* representation: by equation (1),

$$\begin{aligned} n(\log m + o(\log n)) &= B(m,n) + \mathcal{O}(n) + n \cdot o(\log n) \\ &= (1 + o(1))B(m,n). \end{aligned}$$

The significance of the result is that the runtime of all operations is optimal, matching the previous best, and we reduce the memory commitment to a succinct representation with high probability.

To remove the probabilistic expression surrounding the memory allocation, we can just commit to using packed linked-lists for *all* sublists irrespective of their cardinalities. The outcome of this set-up is described in Theorem 2 and we provide the necessary detail and analysis in Appendix C.

6 Conclusion

The PL is an uncomplicated approach to list indexing. The algorithms are simple and offer immediate improvements over solutions that employ linear traversals. The data structure admits an instance that is close to optimal in space with respect to ordered sets. This is the first succinct indexed list with optimal time query and update operations. There are two directions for further work. In all prior work, update operations are amortized. Thus, even in the non-succinct case, designing worst-case update operations is an open problem. Further, achieving a non-probabilistic memory allocation with optimal time operations would conclude this line of work.

References

1 Arne Andersson and Ola Petersson. Approximate indexed lists. *Journal of Algorithms*, 29(2):256–276, 1998.

2 Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *FOCS*, pages 787–796. IEEE, 2010.

3 Paul F Dietz. Optimal algorithms for list indexing and subset rank. In *Workshop on Algorithms and Data Structures*, pages 39–46. Springer, 1989.

4 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *STOC*, pages 345–354, 1989.

5 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *SEA*, pages 326–337. Springer, 2014.

6 Alexander Golynski, J Ian Munro, and S Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *SODA*, volume 6, pages 368–373, 2006.

7 Rodrigo González and Gonzalo Navarro. Rank/select on dynamic compressed sequences and applications. *Theoretical Computer Science*, 410(43):4414–4422, 2009.

8 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850. Society for Industrial and Applied Mathematics, 2003.

9 Torben Hagerup. Highly succinct dynamic data structures. In *International Symposium on Fundamentals of Computation Theory*, pages 29–45. Springer, 2019.

10 Meng He and J Ian Munro. Succinct representations of dynamic strings. In *International Symposium on String Processing and Information Retrieval*, pages 334–346. Springer, 2010.

11 William L Holland, Anthony Wirth, and Justin Zobel. Recency queries with succinct representation. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, 2020.

12 Svante Janson. Large deviation inequalities for sums of indicator variables. *arXiv preprint*, 2016. `arXiv:1609.00533`.

13 J Ian Munro and Yakov Nekrich. Compressed data structures for dynamic sequences. In *Algorithms-ESA 2015*, pages 891–902. Springer, 2015.

14 J Ian Munro and Kaiyu Wu. Succinct data structures for chordal graphs. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, 2018.

15 Gonzalo Navarro and Yakov Nekrich. Optimal dynamic sequence representations. In *SODA*, pages 865–876. SIAM, 2013.

16 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms (TALG)*, 10(3):1–39, 2014.

17 Rasmus Pagh. Low redundancy in static dictionaries with o (1) worst case lookup time. In *ICALP*, pages 595–604. Springer, 1999.

18 Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct dynamic data structures. In *WADS*, pages 426–437. Springer, 2001.

19 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

20 Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004.

A Allowing for Repetitions

If we allow for repetitions, there are some precedents (from Python or Java) on how to modify the behavior of the operations. For example, the `position` function could return the lowest index of an item occurrence. The question is; how does this affect the theoretical performance?

Repetitions reduce the entropy of the list and the size of the lower bound. Certainly there are mechanisms that can leverage the reduction in entropy in the sublists (i.e. we can apply compression techniques). However, there is a point (in terms of the number of repetitions) at which this becomes a dynamic sequences problem and we would shift towards using the dynamic string of Munro and Nekrich [13].

The key difference between our version of the problem and that of the original propblem proposed by Dietz [3], is that the updates and the `position` query take pointers (or "records") as inputs. It would make less sense to have repeated records in the list if we access them via pointers. Thus it is sensible to think of these records – each associated with a key in the list – as distinct from each other. Given that the index query returns a key but not a record, we think, even under the Dietz version of the problem, it is natural to think of the keys as distinct.

Further, we think our formalization of indexing an *ordered set* is what makes this an interesting problem.

B Relegated Proofs

B.1 Memory Allocation for Sublists

▶ **Lemma 7.** *For $\gamma > 0$,*

$$\max_{c\in[\sigma]} |\mathcal{L}_c| \leq C_1 \log n \tag{5}$$

with probability at least $1 - e^{-\mathcal{O}(C_1)\log n} - \xi_\gamma$.

Proof. Fix a sublist c. Let X_i be an indicator for the event that the i^{th} item added to $\mathcal{L}$ is placed in sublist c. It holds that $\mathrm{E}[X_i] = 1/\sigma$. The cardinality $X = \sum_{i=1}^{n} X_i$ of sublist c is the sum of k-wise independent random variables (for $k = n^\alpha, \alpha \in (0,1)$). Therefore, by Theorem 5, with $\mu = \mathrm{E}[X] \in [\log n/(2\log\log n), 2\log n/\log\log n]$ by Equation (3),

$$\begin{aligned}\mathsf{Pr}[X \geq C_1 \log n] &\leq \mathsf{Pr}[X \geq C_1 \log\log n/2 \cdot \mu] \\ &\leq \mathsf{Pr}[|X-\mu| \geq (C_1 \log\log n/2 - 1)\cdot\mu] \\ &\leq e^{-\mathcal{O}(C_1)\cdot\log n},\end{aligned}$$

by our choice of k and Theorem 4. The third line holds on the condition that $C_1 \geq 3\log\log n$. A union bound absorbs the probability that the construction of the hash family fails and we arrive at the stated result. ◀

▶ **Lemma 8.** *For $C_1 = \mathcal{O}(\log n)$, C set to a sufficiently large constant and any $\gamma > 0$,*

$$|\{c \mid |\mathcal{L}_c| \geq C \cdot \log n/\log\log n\}| \leq \frac{n}{C_1 \log n \cdot \log\log n} \tag{6}$$

with probability at least $1 - e^{-n/(\mathcal{O}(C_1)\log n\log\log n)} - \xi_\gamma$.

Before we proceed with the proof, we need an additional result. Let $X_{i,c}$ indicate that item i is allocated to sublist c and $X_c = \sum_{i=1}^{n} X_{i,c}$ denote the sublist cardinalities for $c \in [\sigma]$. Further, we define

$$Y_c = \begin{cases} 1 & \text{if } X_c > C \cdot \log n/\log\log n \\ 0 & \text{otherwise}\end{cases}$$

to indicate the event that sublist c exceeds the threshold. The task is to bound the sum $\sum_{c=1}^{\sigma} Y_c$. However, the random variables $\{Y_c\}$ are not independent. Fortunately, the $\{Y_c\}$ are *negatively related* and a Chernoff bound can be applied to acquire a concentration result on the sum of the indicators [12]. Negatively related random variables are defined as follows.

▶ **Definition 13** ([12]). *The indicator random variables $\{Y_i\}_{i\in[\sigma]}$ (defined on some probability space) are negatively related if for each $j \leq \sigma$ there exists further random variables $\{J_{i,j}\}_{i\in[\sigma]}$ defined on the same probability space such that:*

- *the definition of the random vector $\{J_{i,j}\}_{i\in[\sigma]}$ equals the conditional distribution of $\{Y_i\}_{i\in[\sigma]}$ given $Y_j = 1$;*
- *$J_{i,j} \leq Y_i, \forall i \neq j$.*

▶ **Lemma 14.** *The random variables $\{Y_c\}_{c\in[\sigma]}$ are negatively related.*

Proof. We can imagine this as an experiment where n balls are thrown into σ bins where the bin locations for the balls are n^α-wise independent for $\alpha \in (0,1)$. Let X_c denote the load of bin c and let $L = C \cdot \log n / \log\log n$. The indicator Y_c is 1 if $X_c > L$. To demonstrate that the indicators $\{Y_c\}$ are negatively related we construct the distributions $\{J_{c,j}\}_{c\in[\sigma]}$ for $j \in [\sigma]$ that satisfy the two properties of Definition 13. Fix a $j \in [\sigma]$. The random variables $\{J_{c,j}\}$ take the following form. First throw the n balls into σ bins as above. If $X_j > L$, then set $J_{c,j} = Y_c$ for all $c \in [\sigma]$. Otherwise, pick $\lfloor (L - X_j + 1) \rfloor$ balls uniformly at random from bins in $[\sigma] \setminus \{j\}$ and place them in bin j. Let X_c^* denote the modified bin loads. Now, $J_{c,j} = 1$ if $X_c^* > L$. As the bin loads X_c^* are conditioned on $X_j > L$, the indicators have the correct distribution. Further, $J_{c,j} \le Y_c$ as we only remove items from bins. Thus, the $\{Y_c\}$ are negatively related. ◀

Now that we can apply a Chernoff bound, we can complete the proof of Lemma 8.

Proof of Lemma 8. As X is the sum of k-wise independent random variables, by Theorem 5, our choice of k and the observation that $\mu = \mathrm{E}[X] \le 2\log n / \log\log n$, the following holds:

$$\begin{aligned}\mathsf{Pr}[Y_c = 1] &= \mathsf{Pr}[X_c > C \cdot \log n / \log\log n] \\ &\le \mathsf{Pr}[|X_c - \mu| > (C/2 - 1) \cdot \mu] \\ &\le e^{-\mathcal{O}(C)\mu}.\end{aligned}$$

Let $Y = \sum_{c=1}^{\sigma} Y_c$ denote the number of sublists that exceed the cardinality threshold.

$$\mathrm{E}[Y] \le e^{-\mathcal{O}(C)\mu} \cdot \sigma \le e^{-\mathcal{O}(C)\log n/\log\log n} \cdot \frac{n \log\log n}{\log n} \le \frac{n}{(\log n)^{\mathcal{O}(C)}}. \tag{7}$$

The indicators $\{Y_c\}$ are not independent. However, by Lemma 14, they are *negatively related* and we are permitted to apply a Chernoff bound on their sum. Therefore, for $\delta = n/(C_1 \cdot 2\mathrm{E}[Y] \log n \log\log n)$,

$$\mathsf{Pr}[Y > n/(C_1 \cdot \log n \log\log n)] \le \mathsf{Pr}\big[|Y - \mathrm{E}[Y]| > \delta \mathrm{E}[Y]\big] \le e^{-n/(\mathcal{O}(C_1)\log n \log\log n)}$$

on the condition that $\delta \ge 1$. The latter occurs if

$$1 \le \delta = \frac{n}{C_1 \cdot 2\mathrm{E}[Y] \log n \log\log n} \le \frac{(\log n)^{\mathcal{O}(C)}}{2C_1 \log\log n}$$

The second inequality comes from Equation (7) and holds for $C_1 = \mathcal{O}(\log n)$ and sufficiently large C. Adding the error rate of the hash family, via the union bound, completes the proof. ◀

▶ **Lemma 9.** *For a sufficiently large constant C and $\gamma > 0$, $|Q_C| \le n/\log\log n$ with probability at least $1 - \xi_\gamma$.*

Proof. By design, $|Q_C| \le n/\log\log n$ if inequalities (5) & (6) both hold. By Lemmas 7 and 8, with $C_1 = \Theta(\log n)$, in conjunction with a union bound, both the inequalities hold with probability $1 - (1/n)^{\omega(1)} - \xi_\gamma$. This completes the proof. ◀

▶ **Lemma 10.** *For any $\gamma > 0$, if each item occupies K bits, the memory allocation of the sublists, with sufficiently large threshold parameter C, aggregates to $n(K + o(\log n)) + o(n)K$ bits with probability at least $1 - \mathcal{O}(1/n^\gamma)$.*

Proof. The structure is comprised of at most σ packed linked-lists. These linked-lists contain at most $n/\log\log n + \sigma$ nodes. Each node contains $\log\log n$ items of K bits and two $\mathcal{O}(\log n)$ bit pointers. This accumulates to a bit commitment of size

$$\begin{aligned}
&\left(\frac{n}{\log\log n} + \sigma\right)(K \cdot \log\log n + \mathcal{O}(\log n)) \\
&= nK + n \cdot o(\log n) + \sigma K \cdot \log\log n + \sigma\mathcal{O}(\log n) \\
&= n(K + o(\log n)) + K\Theta\left(\frac{n\log^2\log n}{\log n}\right) + \Theta(n\log\log n) \\
&= n(K + o(\log n)) + o(n)K. \qquad (8)
\end{aligned}$$

By Lemma 9, the structure contains at most $n/\log\log n$ uncompressed items with probability $1 - \xi_\gamma = 1 - \mathcal{O}(1/n^\gamma)$. Each uncompressed item occupies $\mathcal{O}(\log n)$ bits. Thus, combined, uncompressed items occupy

$$n/\log\log n \cdot \mathcal{O}(\log n) = n \cdot o(\log n)$$

bits. This allocation is absorbed by the allocation for compressed items (Equation (8)). ◀

B.2 Performance for PL+

▶ **Lemma 11.** *For any $\gamma > 0$, for $m = \text{poly}(n)$, to store a list of n items, PL+ occupies $n(\log m + o(\log n))$ bits of space with probability at least $1 - \mathcal{O}(1/n^\gamma)$.*

Proof. For the permutation $\pi : [m] \to [m]$, the leftmost $\log\sigma$ bits specify the subproblem and the rightmost $\lceil\log m\rceil - \log\sigma$ bits are stored in the sublist. Thus, the space occupied by the sublists is determined by Lemma 10 with key size $K = \lceil\log m\rceil - \log\sigma$. By equation (4), the cost of the dynamic string is $n(\log\sigma + o(\log n))$ bits. Aggregated with the sublists, we get a memory commitment (measured in bits) of

$$\begin{aligned}
&n(\log\sigma + o(\log n)) + nK + n \cdot o(\log n) + o(n)K \\
&= n(\log\sigma + o(\log n)) + n\log(m/\sigma) + o(n)\log(m/\sigma) \\
&= n(\log m + o(\log n)) + o(n) \\
&= n(\log m + o(\log n)),
\end{aligned}$$

as required. The forth term in line 2 is $o(n)$ on the stated condition $m = \text{poly}(n)$. The failure probability is inherited from Lemma 10. ◀

▶ **Lemma 12.** *On a list of n items, PL+ performs* `index` *and* `position` *in $\mathcal{O}(\log n/\log\log n)$ time and updates* (`insert` *or* `delete`) *complete in $\mathcal{O}(\log n/\log\log n)$ amortized time.*

Proof. By Lemma 3, taking parameter φ as constant, all queries to the random string take $\mathcal{O}(\log n/\log\log n)$ time and update operations require $\mathcal{O}(\log n/\log\log n)$ amortized time. By Lemma 6, all operations on the uncompressed implementation for PackUnpack take $\mathcal{O}(\log n/\log\log n)$ time. The query $\texttt{index}^{\mathsf{PL+}}(i)$ demands two operations on the random string and an $\texttt{index}^{\mathsf{PackUnpack}}$ query on a sublist. The local $\texttt{index}^{\mathsf{PackUnpack}}(\mathcal{L}_c, i')$ requires either a linear scan of length $\mathcal{O}(\log n/\log\log n)$ or an index operation on a fast uncompressed solution. The $\texttt{position}^{\mathsf{PL+}}$ query requires one constant time hash evaluation, a `select` query and $\texttt{position}^{\mathsf{PackUnpack}}$ query on the subproblem. Similar to the case of the `index` query, all subroutines take $\mathcal{O}(\log n/\log\log n)$ time and the claim for `position` holds.

An update is obtained by a combination of an update to the dynamic string and an update to the subproblems. The former operation has an amortized cost of $\mathcal{O}(\log n/\log\log n)$ and the runtime of the latter, similar to the query operations, requires a constant number of (possibly amortized) $\mathcal{O}(\log n/\log\log n)$ time subroutines. Periodic rebuilding adds amortized $o(\log n/\log\log n)$ time to each update. This completes the proof. ◀

C Alternative implementation for PL+

In our main result we achieve a succinct representation with high probability. The structure can be modified such that a succinct representation occurs with probability one: we commit to using packed linked lists (pack) and tolerate sublists with cardinality $\omega(\log n/\log\log n)$. Further, the local $\texttt{index}^{\mathsf{Pack}}(\mathcal{L}_c, i)$ query can take advantage of knowing *which* node the offset i belongs to. For every packed linked-list, we store a dynamic array of pointers, which we name *skip* pointers, where the j^{th} pointer points to the j^{th} node in the chain. This permits constant time access to a specified node. Consequently, the $\texttt{index}^{\mathsf{Pack}}(\mathcal{L}_c, i)$ performs small $\mathcal{O}(\log\log n)$ traversals on a portion of the sublist $\mathcal{L}_c$. The skip pointers add a negligible $\sigma \cdot \mathcal{O}(\log n)$ bits to the memory allocation of the structure. The runtimes of the other operations depend on the cardinality of the engaged sublist. The latter is at most $\mathcal{O}(\log n)$ with probability $1 - \mathcal{O}(1/n^{\gamma})$. Note that Lemma 10 accounts for σ packed lists. Thus, we arrive at the following result.

▶ **Theorem 2.** *For any constant $\gamma > 0$, an ordered set of n items, drawn from the universe $[m]$, where $m = \text{poly}(n)$, can be stored in $n(\log m + o(\log n))$ bits and support* `index` *in $\mathcal{O}(\log n/\log\log n)$ time,* `position` *in $\mathcal{O}(\log n)$ time with probability $1 - \mathcal{O}(1/n^{c})$ and any sequence of $\mathcal{O}(n)$ updates* (`insert` *or* `delete`) *takes $\mathcal{O}(n \log n)$ time with probability $1 - \mathcal{O}(1/n^{\gamma})$.*

As the expected cardinality of a sublist is $n/\sigma = \mathcal{O}(\log n/\log\log n)$, the *expected* runtime of `position` and the expected amortized runtime of the updates is $\mathcal{O}(\log n/\log\log n)$.

Super-Cubic Lower Bound for Generalized Karchmer–Wigderson Games

Artur Ignatiev
St.Petersburg State University, Russia
HSE University, St.Petersburg, Russia

Ivan Mihajlin
St.Petersburg Department of Steklov Mathematical Institute of Russian Academy of Sciences, Russia

Alexander Smal
St.Petersburg Department of Steklov Mathematical Institute of Russian Academy of Sciences, Russia
Technion, Haifa, Israel

Abstract

In this paper, we prove a super-cubic lower bound on the size of a communication protocol for generalized Karchmer–Wigderson game for an explicit function $f : \{0,1\}^n \to \{0,1\}^{\log n}$. Lower bounds for original Karchmer–Wigderson games correspond to De Morgan formula lower bounds, thus the best known size lower bound is cubic. The generalized Karchmer–Wigderson games are similar to the original ones, so we hope that our approach can provide an insight for proving better lower bounds on the original Karchmer–Wigderson games, and hence for proving new lower bounds on De Morgan formula size.

To achieve super-cubic lower bound we adapt several techniques used in formula complexity to communication protocols, prove communication complexity lower bound for a composition of several functions with a multiplexer relation, and use a technique from [18] to extract the "hardest" function from it. As a result, in this setting we are able to show that there is a relatively small set of functions such that at least one of them does not have a small protocol. The resulting lower bound of $\tilde{\Omega}\left(n^{3.156}\right)$ is significantly better than the bound obtained from the counting argument.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases communication complexity, circuit complexity, Karchmer–Wigderson games

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.66

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2022/016` [10]

Funding *Artur Ignatiev*: The chapters 1-2 were supported by the Ministry of Science and Higher Education of the Russian Federation, agreement 075-15-2019-1620 date 08/11/2019 and 075-15-2022-289 date 06/04/2022. The chapters 3-4 were supported by the Basic Research Program and the "Priority 2030" program at HSE University.
Alexander Smal: Received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement №802020-ERC-HARMONIC.

1 Introduction

1.1 Background

The circuit complexity of Boolean functions is one of the classical areas of complexity theory. Initially, the study of this area was considered as an easier way to prove $\mathsf{P} \neq \mathsf{NP}$. In fact, proving bounds on circuit size seems to be much easier than proving bounds on the number of steps that some Turing machine does. The desire to prove lower bounds on circuit complexity has attracted many brilliant researchers. The seeming simplicity of this problem turned out to be deceiving. From the Shannon's counting argument we know that a random Boolean function on n inputs has circuit complexity at least $2^{n-o(n)}$ with probability almost 1. At the same time, we do not know any explicit function that does not have linear-sized circuits. Despite over 60 years of attempts, it is still not clear how to prove even more modest lower

bounds – we do not know explicit functions that does not have circuits of size less than $4n$. The best known lower bound for unrestricted Boolean circuits shows that there is a function that can not be computed by a circuit of size less than $(3+\epsilon)n$ [4, 17]. A slightly better lower bound of $5n - o(n)$ [11] can be obtained if we consider circuits without parity gates.

The desire to learn how to prove lower bounds on circuits motivates us to study more restricted models. One of the most important such models is De Morgan formulas. In contrast to circuit complexity, in formula complexity we know how to prove superlinear lower bounds. Moreover, we know that there is an explicit function that does not have formulas of size $\tilde{\Omega}(n^3)$ [7]. This lower bound is the result of more than 40 years of research starting with works of Subbotovskaya [21] and Khrapchenko [15]. Improving this lower bound is the central challenge in formula complexity.

Karchmer, Raz, and Wigderson [13] suggested an approach for proving super-polynomial formula size lower bound for a Boolean function from class P. The idea of this approach is to study the formula complexity of *the block-composition* of Boolean functions.

▶ **Definition 1.** *Let* $f : \{0,1\}^m \to \{0,1\}$ *and* $g : \{0,1\}^n \to \{0,1\}$ *be Boolean functions.* The block-composition $f \diamond g : \{0,1\}^{nm} \to \{0,1\}$ *is defined by*

$$(f \diamond g)(x_1, \ldots, x_m) = f(g(x_1), \ldots, g(x_m)),$$

where $x_1, \ldots, x_m \in \{0,1\}^n$.

Let $\mathrm{D}(f)$ denote the minimal depth of De Morgan formula for function f. It is easy to show that $\mathrm{D}(f \diamond g) \leq \mathrm{D}(f) + \mathrm{D}(g)$ by constructing a formula for $f \diamond g$ by substituting every variable in a formula for f with a copy of formula for g. Karchmer, Raz, and Wigderson [13] conjectured that this upper bound is roughly optimal.

▶ **Conjecture 2** (The KRW conjecture). *Let* $f : \{0,1\}^m \to \{0,1\}$ *and* $g : \{0,1\}^n \to \{0,1\}$ *be non-constant functions. Then* $\mathrm{D}(f \diamond g) \approx \mathrm{D}(f) + \mathrm{D}(g)$.

If the conjecture is true then there is a polynomially computable function that does not have De Morgan formula of polynomial size, and hence $\mathsf{P} \not\subseteq \mathsf{NC}^1$. Consider the function $h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, which interprets its first input as a truth table of a function $f : \{0,1\}^{\log n} \to \{0,1\}$ and computes the value of the block-composition of $\log n / \log\log n$ functions f on its second input (note that "$\diamond$" is associative):

$$h(f,x) = (\underbrace{f \diamond \cdots \diamond f}_{\log n / \log\log n})(x).$$

It is not hard to see that $h \in \mathsf{P}$. To show that $h \notin \mathsf{NC}^1$, let $\tilde{f}$ be a function with maximal depth complexity. By Shannon's counting argument $\tilde{f}$ has depth complexity roughly $\log n$. Assuming the KRW conjecture, $\tilde{f} \diamond \cdots \diamond \tilde{f}$ has depth complexity roughly $\log n \cdot (\log n / \log\log n) = \omega(\log n)$, and hence $\tilde{f} \diamond \cdots \diamond \tilde{f} \notin \mathsf{NC}^1$. Any formula for h must compute $\tilde{f} \diamond \cdots \diamond \tilde{f}$ if we hard-wire $f = \tilde{f}$ in it, so $h \notin \mathsf{NC}^1$. This argument is especially attractive since it does not seem to break any known meta mathematical barriers such as the concept of "natural proofs" by Razborov and Rudich [20] (the function h is very special, so the argument does not satisfy "largeness" property). It is worth noting that the proof would work even assuming some weaker version of the KRW conjecture, like $\mathrm{D}(f \diamond g) \geq \mathrm{D}(f) + \epsilon \cdot \mathrm{D}(g)$ or $\mathrm{D}(f \diamond g) \geq \epsilon \cdot \mathrm{D}(f) + \mathrm{D}(g)$ for some $\epsilon > 0$. Also, it is not necessary to prove the KRW conjecture for all pairs of functions – it would be enough to show that for every f there exists a hard function g such that $\mathrm{D}(f \diamond g) \approx \mathrm{D}(f) + \mathrm{D}(g)$.

The seminal work of Karchmer and Wigderson [14] established a correspondence between De Morgan formulas for non-constant Boolean function f and communication protocols for the Karchmer–Wigderson game for f.

▶ **Definition 3.** The Karchmer–Wigderson game (KW game) *for Boolean function* $f : \{0,1\}^n \to \{0,1\}$ *is the following communication problem: Alice gets an input* $x \in \{0,1\}^n$ *such that* $f(x) = 0$*, and Bob gets as input* $y \in \{0,1\}^n$ *such that* $f(y) = 1$*. Their goal is to find a coordinate* $i \in [n]$ *such that* $x_i \neq y_i$*. The KW game can be considered as a communication problem for* the Karchmer–Wigderson relation for f*:*

$$\mathrm{KW}_f = \{(x, y, i) \mid x, y \in \{0,1\}^n, i \in [n], f(x) = 0, f(y) = 1, x_i \neq y_i\}.$$

Karchmer and Wigderson showed that the communication complexity of KW_f is exactly equal to the formula depth complexity of f. This correspondence allows us to use communication complexity methods for proving formula depth lower bounds. In fact, Conjecture 2 can be reformulated in terms of communication complexity of the Karchmer–Wigderson game for the block-composition of two arbitrary Boolean functions. Let $\mathrm{CC}(R)$ denote deterministic communication complexity of relation R. This leads to the following reformulation of the KRW conjecture.

▶ **Conjecture 4** (The KRW conjecture (reformulation))**.** *Let* $f : \{0,1\}^m \to \{0,1\}$ *and* $g : \{0,1\}^n \to \{0,1\}$ *be non-constant functions. Then* $\mathrm{CC}(\mathrm{KW}_{f \diamond g}) \approx \mathrm{CC}(\mathrm{KW}_f) + \mathrm{CC}(\mathrm{KW}_g)$.

The study of Karchmer–Wigderson games had already been shown to be a potent tool in the monotone setting – the monotone KW games were used to separate then monotone counterpart of classes NC^1 and NC^2 [13]. Therefore, there is reason to believe that the communication complexity perspective might help to prove new lower bounds in the non-monotone setting.

In a series of works [3, 8, 6, 2] several steps were taken towards proving the KRW conjecture. In the last paper of this series [2] the authors presented an alternative proof for the block-composition of an arbitrary function with the parity function in the framework of the Karchmer–Wigderson games (this result was originally proved in [7] using an entirely different approach). Their result gives an alternative proof of the cubic formula size lower bound for Andreev's function [7].

In [18], the authors proposed a new conjecture, the XOR-KRW conjecture, which is a relaxation of the KRW conjecture. This relaxation is still strong enough to imply $\mathsf{P} \not\subseteq \mathsf{NC}^1$ if proven. They also presented a weaker version of this conjecture that might be used for breaking n^3 lower bound for De Morgan formulas. The conjecture employs an alternative composition operation.

▶ **Definition 5.** *For any* $n, m, k \in \mathbb{N}$ *with* $k \mid n$*, and functions* $f : \{0,1\}^n \to \{0,1\}$ *and* $g_1, \ldots, g_m : \{0,1\}^k \to \{0,1\}^k$ the XOR-composition $f \boxplus_m g : \{0,1\}^{nm} \to \{0,1\}$ *is defined by*

$$(f \boxplus_m (g_1, \ldots, g_m))(x_{1,1}, \ldots, x_{n/k,m}) = \\ f\left(g_1(x_{1,1}) \oplus \cdots \oplus g_m(x_{1,m}), \ldots, g_1(x_{n/k,1}) \oplus \cdots \oplus g_m(x_{n/k,m})\right),$$

where $x_{i,j} \in \{0,1\}^k$ *for all* $i \in [n/k]$ *and* $j \in [m]$*, and* $\oplus$ *denotes bit-wise XOR.*

The authors suggested the following general version of the XOR-KRW conjecture and showed that it implies separation of P and NC^1.

▶ **Conjecture 6** (The XOR-KRW conjecture). *There exist $m \in \mathbb{N}$ and $\epsilon > 0$, such that for all natural $n, k \in \mathbb{N}$ with $k \mid n$, and every non-constant $f : \{0,1\}^n \to \{0,1\}$, there exists $g : \{0,1\}^k \to \{0,1\}^k$ such that $\mathrm{D}(f \boxplus_m g) \geq \mathrm{D}(f) + \epsilon k - O(1)$.*

In [18], the authors suggested focusing on the specific case of $k = n$ and $m = 2$, which might be enough to prove a super-cubic formula size lower bound for a specific formula. In this setting, the authors considered a communication problem that correspond to a universal relation XOR-composed with the Karchmer–Wigderson relation for some function $g : \{0,1\}^n \to \{0,1\}^n$ and proved $1.5n - o(n)$ lower bound on its communication complexity. As an implication of this result, the authors showed the same lower bound on the communication complexity of a universal relation block-composed with the Karchmer–Wigderson relation for some function $g : \{0,1\}^n \to \{0,1\}$.

In this work, we extend the latter result in multiple directions. First of all, we refine the lower bound from [18] such that it works for arbitrary $m \geq 2$. Second, we use this lower bound to prove a super-cubic lower bound on the size of communication protocol for a generalized Karchmer–Wigderson game for some function $f : \{0,1\}^n \to \{0,1\}^{\log n}$.

▶ **Definition 7.** The generalized Karchmer–Wigderson game (generalized KW game) *for function $f : \{0,1\}^n \to \{0,1\}^r$ is the following communication problem: Alice gets an input $x \in \{0,1\}^n$, Bob gets $y \in \{0,1\}^n$, and they are promised that $f(x) \neq f(y)$. Their goal is to find a coordinate $i \in [n]$ such that $x_i \neq y_i$. This problem corresponds to a communication problem for* the generalized Karchmer–Wigderson relation for *f:*

$$\mathrm{KW}_f = \{(x, y, i) \mid x, y \in \{0,1\}^n, i \in [n], f(x) \neq f(y), x_i \neq y_i\}.$$

We show that the following theorem holds.

▶ **Theorem 8.** *There exists a function $f : \{0,1\}^n \to \{0,1\}^{\log n}$ such that any communication protocol for generalized Karchmer–Wigderson game for f has size at least $\tilde{\Omega}(n^{3.156})$.*

To achieve this we extend Håstad's technique [7] to work with communication protocols for generalized Karchmer–Wigderson games.

A universal relation of size n [3] is exactly the generalized Karchmer–Wigderson game for the identity function $\mathrm{Id}_n : \{0,1\}^n \to \{0,1\}^n$: $\mathrm{U}_n = \{(x, y, i) \mid x, y \in \{0,1\}^n, i \in [n], x_i \neq y_i\}$. It is known, that U_m requires m bit of communication. So, one can show that generalized Karchmer–Wigderson game for a function $\{0,1\}^n \to \{0,1\}^m$ that computes Id_m of the first m bits of its input also requires m bits of communication. For this reason it is crucial that we focus on the regime of $\{0,1\}^n \to \{0,1\}^{\log n}$, where such an informal argument gives a relatively low bound.

1.2 Organization of the paper

In Section 2, we prove a lower bound for the XOR-composition of Id_n with multiple functions. In Section 3, we use the results of Section 2 to prove the super-cubic lower bound on the protocol size. Section 4 contains a conclusion and a list of open problems. Some parts of the proofs are presented in appendix. In Appendix A, we present the proof of Technical Lemma used in Section 2. In Appendix B, we show that any formula balancing technique that preserves monotonicity can be also used for communication protocols. In Appendix C, we define restrictions for generalized Karchmer–Wigderson games and show that we can use The Main Shrinkage Theorem from [7] to bound the expected size of a protocol after it has been hit with a random restriction.

2 Lower bound for $\mathrm{CC}(\mathrm{KW}_{\mathrm{Id}_n \boxplus_m g})$

In this section, we abuse the notation in the following way: talking about communication complexity of a generalized Karchmer–Wigderson game for some function f we write $\mathrm{CC}(f)$ instead of $\mathrm{CC}(\mathrm{KW}_f)$, and use the same notation to denote a XOR-composition of functions and the corresponding generalized Karchmer–Wigderson game.

We are going to only focus on the special case of XOR-composition with $k = n$ and prove the following theorem. (Note that Id_n can be replaced with any permutation function.)

▶ **Theorem 9.** *For all $n, m \in \mathbb{N}$, there exists $g : \{0,1\}^n \to \{0,1\}^n$ such that*

$$\mathrm{CC}(\mathrm{Id}_n \boxplus_m g) \geq (2 - 2^{-m+1})n - O(\log n).$$

It is convenient for the proof to extend the definition of XOR-composition to allow m different functions instead of one function applied to m arguments.

▶ **Definition 10.** *For any $n, m \in \mathbb{N}$ and functions $f, g_1 \ldots, g_m : \{0,1\}^n \to \{0,1\}^n$ the* XOR-composition *$f \boxplus (g_1, \ldots, g_m) : \{0,1\}^{nm} \to \{0,1\}^n$ is defined by*

$$\big(f \boxplus (g_1, \ldots, g_m)\big)(x_1, \ldots, x_m) = f\big(g_1(x_1) \oplus \cdots \oplus g_m(x_m)\big),$$

where $x_i \in \{0,1\}^n$ for all $i \in [m]$ and $\oplus$ denotes bit-wise XOR.

We prove Theorem 9 by showing a lower bound for such an extended XOR-composition.

▶ **Theorem 11.** *For all $n, m \in \mathbb{N}$ there exist $g_1, \ldots, g_m : \{0,1\}^n \to \{0,1\}^n$ such that*

$$\mathrm{CC}(\mathrm{Id}_n \boxplus (g_1, \ldots, g_m)) \geq (2 - 2^{-m+1})n - O(\log n).$$

A specific case of this theorem for $m = 2$ was proved in [18, Theorem 21]. To show that Theorem 11 implies Theorem 9 we need the following lemma.

▶ **Lemma 12.** *For all $n, m \in \mathbb{N}$ and functions $g_1, \ldots, g_m \in \{0,1\}^n \to \{0,1\}^n$, there exist a function $g \in \{0,1\}^{n'} \to \{0,1\}^{n'}$ for $n' = n + \lceil \log m \rceil$ such that*

$$\mathrm{CC}(\mathrm{Id}_{n'} \boxplus_m g) \geq \mathrm{CC}(\mathrm{Id}_n \boxplus (g_1, \ldots, g_m)).$$

Proof. Consider function g' defined by the following equation: $g'(x, y) = g_{\overline{y}+1}(x) \circ 0^{\lceil \log n \rceil}$, where $x \in \{0,1\}^n$, $y \in \{0,1\}^{\lceil \log m \rceil}$, "$\circ$" denotes concatenation of bit strings, and $\overline{y}$ denotes a number with a binary expansion y. It is easy to see that for all $x_1, \ldots, x_m \in \{0,1\}^n$,

$$(\mathrm{Id}_{n'} \boxplus_m g')\big((x_1, 0_2), \ldots, (x_m, (m-1)_2)\big) = \big(\mathrm{Id}_n \boxplus (g_1, \ldots, g_m)\big)(x_1, \ldots, x_m) \circ 0^{\lceil \log n \rceil},$$

where k_2 defines binary expansion of k of length $\lceil \log m \rceil$. Since $\mathrm{Id}_n \boxplus (g_1, \ldots, g_m)$ is a subfunction of $\mathrm{Id}_{n'} \boxplus g'$, the lower bound applies. ◀

Proof of Theorem 9 assuming Theorem 11. Let $\tilde{n} = n - \lceil \log m \rceil$. By Theorem 11 there exist functions $g_1, \ldots, g_m : \{0,1\}^{\tilde{n}} \to \{0,1\}^{\tilde{n}}$ such that

$$\mathrm{CC}(\mathrm{Id}_{\tilde{n}} \boxplus (g_1, \ldots, g_m)) \geq (2 - 2^{-m+1})\tilde{n} - O(\log \tilde{n}) = (2 - 2^{-m+1})n - O(\log n).$$

Now we apply Lemma 12 to get the desired bound:

$$\mathrm{CC}(\mathrm{Id}_n \boxplus_m g') \geq \mathrm{CC}(\mathrm{Id}_{\tilde{n}} \boxplus (g_1, \ldots, g_m)) \geq (2 - 2^{-m+1})n - O(\log n).$$ ◀

2.1 Proof of Theorem 11

In the proof of this theorem, we need to consider communication complexity in *half-duplex communication model with zero*. The idea of half-duplex communication was introduced in [9] and later developed in [1]. In half-duplex communication model, every player can send messages in every round, but if both players send simultaneously, then their messages get lost. That allows them to "mix" classical communication protocols (see proof of Lemma 16). Formally speaking, every round each player chooses one of three actions: "send 0", "send 1", or "receive". If one of the players sends while the other receives then communication works like in the classical model. If both players send simultaneously then both messages get lost (and players do not know about it). If both players receive simultaneously they get zeroes.

In [18], the authors introduced *partially half-duplex communication*. In partially half-duplex communication problems the players receive inputs divided in two parts: Alice receives (f, x), Bob receives (g, y). They can use half-duplex communication but with a restriction: if $f = g$ then the communication must have only *classical* rounds (every round one of the players sends some bit and the other one receives).

We are going to prove a lower bound for $\mathrm{Id}_n \boxplus (g_1, \ldots, g_m)$ by induction on m. Assume that we have already proven a lower bound for some value of m. To prove a lower bound for $m + 1$ we take the following steps:

- prove a lower bound on partially half-duplex communication complexity for an intermediate communication problem where g_{m+1} is replaced with *a multiplexer relation* (see Definition 13 and Lemma 15),
- argue that if the intermediate communication problem is hard for partially half-duplex communication then there is a "hardest" function g_{m+1} such that if we hard-wire it into the multiplexer then the resulting communication problem has the same lower bound (see Lemma 16).

▶ Remark. Starting from here, we will always consider *non-promise* communication problems. For generalized Karchmer–Wigderson games this means that if the promise is broken, i.e., $f(x) = f(y)$, then the players are allowed to output a special symbol "$\perp$". It is not hard to see that communication complexity of non-promise Karchmer–Wigderson game differs by no more than two from communication complexity of the promise version: the players can use a protocol for promise version and then verify the answer using additional two bits of communication. Since the complexity differs only by an additive constant, this will not affect our lower bounds.

We start with the definition of an intermediate communication problem.

▶ **Definition 13.** *For any $n, m \in \mathbb{N}$ and functions $f, g_1, \ldots, g_m : \{0,1\}^n \to \{0,1\}^n$ the* XOR-composition with a multiplexer *$f \boxplus (g_1, \ldots, g_m, \mathrm{Mux})$ defines the following communication problem: Alice is given $x_1, \ldots, x_m, z_a \in \{0,1\}^n$ and some function $h_a : \{0,1\}^n \to \{0,1\}^n$, Bob is given $y_1, \ldots, y_m, z_b \in \{0,1\}^n$ and some function $h_b : \{0,1\}^n \to \{0,1\}^n$. Their goal is to find $i \in [(m+1)n]$ such that $(x_1 \circ \cdots \circ x_m \circ z_a)_i \neq (y_1 \circ \cdots \circ y_m \circ z_b)_i$. If $h_a \neq h_b$ or $g_1(x_1) \oplus \cdots \oplus g_m(x_m) \oplus h_a(z_a) = g_1(y_1) \oplus \cdots \oplus g_m(y_m) \oplus h_b(z_b)$ then the players are allowed to output $\perp$.*

For the remaining of the section we fix n. Let $\mathcal{P}_n$ be the set of all permutations of $\{0,1\}^n$, and $N = 2^n$, $\mathcal{X}_m = \mathcal{P}_n \times \{0,1\}^{nm} \times \{0,1\}^n$. To simplify the formulas we are going to use $\vec{x}$ and $\vec{y}$ to denote $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$, respectively. In the same manner we denote $g_1, \ldots, g_m$ with $\vec{g}$ and use $\vec{g} \otimes \vec{x}$ as a shortcut for $g_1(x_1) \oplus \cdots \oplus g_m(x_m)$. We use $\mathrm{CC}_{A \times B}$ and $\mathrm{CC}^{phd}_{A \times B}$ to denote the communication complexity of a communication problem restricted to a rectangle $A \times B$.

The proof is by induction on m. The following lemma plays the role of an induction hypothesis.

▶ **Lemma 14.** *For all $m, k \in \mathbb{N}$, $k \le n-3$, there exist functions $g_1, \dots, g_m \in \mathcal{P}_n$ such that for any set $S \subset \{0,1\}^{nm}$ of size $2^{-k}N^m$,*

$$\mathrm{CC}_{S\times S}(\mathrm{Id}_n \boxplus (g_1, \dots, g_m)) \ge (2 - 2^{-m+1})n - k - O(\log n).$$

The base case for $m = 1$ can be proved using to Shannon's counting argument but we do not prove it as our proof of the first induction step does not depend on it (see the proof of Lemma 15).

Assuming Lemma 14 for $m > 1$ or nothing for $m = 1$ we follow the ideas from [18] and prove a lower bound on partially half-duplex communication complexity of the XOR-composition with a multiplexer, the communication problem with one of the functions replaced by the multiplexer relation. A half-duplex protocol for $\mathrm{Id}_n \boxplus (g_1, \dots, g_m, \mathrm{Mux})$ is called *partially half-duplex* if it has the following property: whenever Alice and Bob are given the same function they are not allowed to perform non-classical communication. In other words, in a partially half-duplex protocol Alice and Bob never send or receive simultaneously if $h_a = h_b$. Let CC^{phd} denote partially half-duplex communication complexity.

▶ **Lemma 15.** *For all $m, k \in \mathbb{N}$, $k \le n-3$, there exist functions $g_1, \dots, g_m \in \mathcal{P}_n$ such that for any set $S \in \mathcal{X}_m$ of size $2^{-k}N^{m+1}N!$,*

$$\mathrm{CC}^{phd}_{S\times S}(\mathrm{Id}_n \boxplus (g_1, \dots, g_m, \mathrm{Mux})) \ge \big(2 - 2^{-m}\big)n - k - O(\log n).$$

This lemma generalizes a lemma proved in [18, Lemma 46]. After we prove Lemma 15 for some value of m we use the following "extraction" lemma to replace the multiplexer relation with a function. This lemma is the main reason why we need (partially) half-duplex communication.

▶ **Lemma 16.** *If $\mathrm{CC}^{phd}_{S\times S}(\mathrm{Id}_n \boxplus (g_1, \dots, g_m, \mathrm{Mux})) \ge \gamma$ on any rectangle $S \times S$ with $|S| \ge \delta|\mathcal{P}_n|$ for some $\gamma, \delta > 0$ then exists $g_{m+1} \in \mathcal{P}_n$ such that*

$$\mathrm{CC}_{S'\times S'}(\mathrm{Id}_n \boxplus (g_1, \dots, g_m, g_{m+1})) \ge \gamma - \lceil \log n \rceil - 2$$

for any set $S' \subset \{0,1\}^{nm}$ of size at least δ.

Proof. We prove by contradiction. Suppose that for every every function $h \in \mathcal{P}_n$ there is a rectangle $S_h \times S_h$ such that $|S_h| \ge \delta$ and $\mathrm{CC}_{S_h\times S_h}\big(\mathrm{Id}_n \boxplus (g_1, \dots, g_m, h)\big) \le d < \gamma - \lceil \log n \rceil - 2$ for some $d \in \mathbb{N}$. Let $S = \bigcup_{h\in\mathcal{P}_n} \{(h, \vec{x}, z) \mid (\vec{x}, z) \in S_h\}$. It is easy to see that $|S| \ge \delta|\mathcal{P}_n|$. We are going to show that in this case $\mathrm{CC}^{phd}_{S\times S}(\mathrm{Id}_n \boxplus (g_1, \dots, g_m, \mathrm{Mux})) < \gamma$.

Consider the following half-duplex protocol for $\mathrm{Id}_n \boxplus (g_1, \dots, g_m, \mathrm{Mux})$. Alice, who is given $\vec{x}$, z_a, and h_a, follows the protocol Π_{h_a} using $(\vec{x}, z_a)$ as her input. Meanwhile Bob, who is given $\vec{y}$, z_b, and h_b, follows the protocol Π_{h_b} using $(\vec{y}, z_b)$ as his input. If $h_a \ne h_b$ they might use different protocols, which is fine because we are in the half-duplex communication model. When Alice reaches some leaf of Π_{h_a} she starts receiving until the end of round d. Bob does the same thing. After d rounds of communication Alice has a candidate i for the answer of the game, which is a valid output if $h_a = h_b$. Bob has a candidate j, that is equal to i if $h_a = h_b$. Now Alice and Bob just need to check that indeed $(\vec{x}, z_a)_i \ne (\vec{y}, z_b)_j$ and $i = j$, which can be done in $\lceil \log n \rceil + 2$ rounds of communication. They output i if both conditions are true, and $\perp$ otherwise. The total number of rounds of this half-duplex protocol is $d + \lceil \log n \rceil + 2 < \gamma$. ◀

Together Lemma 15 and Lemma 16 immediately imply Lemma 14 for $m+1$.

Proof of Lemma 14. We prove the statement of the lemma for $m+1$ functions. Apply Lemma 16 for the result of Lemma 15 with $\delta = 2^{-k}N^{m+1}$ and $\gamma = \big(2-2^{-m}\big)n - O(\log n)$. ◀

That concludes the induction step. Theorem 11 follows from Lemma 14 by setting $k = 0$.

Proof of Theorem 11. By Lemma 14 for $k = 0$, we have

$$\mathrm{CC}(\mathrm{Id}_n \boxplus (g_1, \ldots, g_m)) \geq (2 - 2^{-m+1})n - O(\log n).$$ ◀

It remains for us to prove Lemma 15.

2.2 Proof of Lemma 15

The proof of Lemma 15 has many similarities with the proof of the main result of [18]. The main technical novelty of the following proof is that we have found a way to make the proof inductive. In [18], there is a dichotomy between two communication complexity reductions: a reduction from deterministic equality and a reduction from non-deterministic non-equality. In this proof, we replace the reduction from deterministic equality with an inductive step.

The proof is split in two parts. In the first part, given a protocol we will find a large enough collection of subrectangles in it. All the nodes corresponding to these subrectangles will have equal *partial transcripts*. In the classical communication model, *a partial transcript* of a node of the protocol is a bit string consisting of all the messages that are sent on the path from the root to this node. For a partially half-duplex protocol we can also define a partial transcript of a node in the same way if all the preceding communication of the node is classical. An important difference is that in the classical model a partial transcript uniquely defines a node. In the half-duplex model the same partial transcript of length d can correspond to at most 2^d nodes of the protocol, e.g. a partial transcript "00" can correspond to 4 different nodes: a node where both messages were sent by Alice, a node where both messages were send by Bob, and two nodes where both players sent messages in different order.

▶ **Lemma 17.** *For any partially half-duplex protocol* Π *for* $\mathrm{Id}_n \boxplus (g_1, \ldots, g_m, \mathrm{Mux})$ *on a rectangle* $S \times S$, $|S| \geq 2^{-k}N^{m+1}N!$, *there exists a rectangle of inputs* $R \times R$, $R \subset S$, $|R| \geq 8N^m N!$, *and a string* $T \in \{0,1\}^{n-k-3}$, *such that if Alice and Bob are given the same input from* R *then the transcript of the first* $n-k-3$ *rounds is equal to* T.

Proof. Let $D = \{((h, \vec{x}, y), (h, \vec{x}, y)) \mid (h, \vec{x}, y) \in \mathcal{S}\}$ be the subset of inputs where player's inputs are identical. First, we need to notice that if Alice and Bob are given inputs from D, then they perform only classical communication. Consider the first $n-k-3$ rounds of communication. There are at most 2^{n-k-3} different transcripts of length $n-k-3$, so there is a transcript T that corresponds to at least $|D|/2^{n-k-3} = 8N^m N!$ inputs from D. Let R be the set of all these inputs. ◀

Let us emphasize again, that the set S constructed here is not consolidated in a single node of the protocol. All the elements of S have the same transcript of the first $n-k-3$ rounds but these transcripts do not include the information who sends each of the messages, so in fact the same transcripts can correspond to different nodes of the protocol. Note that any two inputs from S with the same function g necessarily belong to the same node of the protocol as all the rounds are classical.

The last thing that we need for the proof of Lemma 15 is the "technical lemma". The following combinatorial object is useful for understanding the structure of subsets of inputs.

▶ **Definition 18.** *For a subset of inputs $S \subseteq \mathcal{X}_m$ we define* a domain graph *to be a bipartite graph $G_S = (U_S, V_S, E_S)$, such that $U_S \subseteq \mathcal{P}_n$, $V_S \subseteq \{0,1\}^{n(m+1)}$, and $(h,(\vec{x},y)) \in E_S \iff (h,\vec{x},y) \in S$.*

The following "technical lemma" is a generalization of [18, Lemma 39].

▶ **Lemma 19.** *Let $S \subseteq \mathcal{X}_m$ be a subset of inputs such that $|S| \geq N^m \cdot N!$, and let $G_S = (U_S, V_S, E_S)$ be a domain graph of S. If $\min_{h \in U_S}\{\deg_{G_S}(h)\} \geq 4N^m$ and*

$$\forall h \in \mathcal{P}_n,\ \forall y \in \{0,1\}^n,\ \left|\{\vec{x} \mid (h,(\vec{x},y)) \in E_S\}\right| \leq N^{m-\alpha} \tag{1}$$

for some $\alpha > 0$, then there is a set $H \subseteq U_S$ of size $2^{\Omega(N^\alpha)}$ such that for all distinct $h_1, h_2 \in H$, there exist $(\vec{x},y)$: $(h_1,\vec{x},y) \in S$, $(h_2,\vec{x},y) \in S$, and $h_1(y) \neq h_2(y)$.

The proof of Lemma 19 repeats almost verbatim the proof of the original lemma [18, Lemma 39]. We present it in Appendix A.

Now we are ready to prove Lemma 15 by showing that if $\mathrm{Id}_n \boxplus (g_1,\dots,g_m,\mathrm{Mux})$ has a short protocol then we can either contradict induction hypothesis by extracting a short protocol for $\mathrm{Id}_n \boxplus (g_1,\dots,g_m)$ or non-deterministically solve non-equality more efficiently than it is possible.

▶ **Lemma 15.** *For all $m,k \in \mathbb{N}$, $k \leq n-3$, there exist functions $g_1,\dots,g_m \in \mathcal{P}_n$ such that for any set $S \in \mathcal{X}_m$ of size $2^{-k}N^{m+1}N!$,*

$$\mathrm{CC}^{phd}_{S\times S}(\mathrm{Id}_n \boxplus (g_1,\dots,g_m,\mathrm{Mux})) \geq \big(2-2^{-m}\big)n - k - O(\log n).$$

Proof. Let $\alpha = 1-2^{-m}$. Suppose that Π is a partially half-duplex protocol for $\mathrm{Id}_n \boxplus (g_1,\dots,g_m,\mathrm{Mux})$ of depth d. Let R be the set provided by Lemma 17. Let $S' = R \setminus \{(h,\vec{x},y) \mid \deg_{G_S}(h) < 4N^m\}$, so $|S'| > 4N^m N!$. Let $G_{S'} = (U_{S'}, V_{S'}, E_{S'})$ be a domain graph of S'. The minimal degree of the vertices in $U_{S'}$ is at least $4N^m$.

Suppose that there is $h \in \mathcal{P}_n$ and $y \in \{0,1\}^n$ such that $\left|\{\vec{x} \mid (h,(\vec{x},y)) \in E_{S'}\}\right| > N^{m-\alpha}$. Let $S_{h,y} = \{(h,\vec{x},y) \mid (h,(\vec{x},y)) \in E_{S'}\}$. We can extract from Π a classical protocol Π' of depth at most $d-n+k+3$ that solves $\mathrm{Id}_n \boxplus (g_1,\dots,g_m,\mathrm{Mux})$ on $S_{h,y} \times S_{h,y}$. This follows from the fact that Π is partially half-duplex, so it has only classical rounds on inputs from $S_{h,y} \times S_{h,y}$. Let $W = \{x_1,\dots,x_m \mid (h,(x_1,\dots,x_m,y)) \in E_{S'}\}$.

- If $m = 1$ then the protocol Π' can be used to solve an equality problem on a set W. Given inputs $x_a, x_b \in W$, Alice and Bob simulate the protocol for Π' on $S' \times S'$ for inputs (h,x_a,y) and (h,x_b,y). If the protocol outputs $\perp$ then the players output 1, otherwise they output 0. For inputs (h,x_a,y) and (h,x_b,y), the protocol outputs $\perp$ if and only if $x_a = x_b$, so this reduction gives a correct protocol for EQ_W of the same depth. Any protocol for EQ_W has depth at least $\log|W| \geq \log(N^{1/2}) = n/2$. By the reduction, the same lower bound applies for the protocol for Π on $S' \times S'$. Thus, we have $d \geq n-k-3+n/2 = (2-2^{-m})n - k - 3$.
- If $m > 1$ then we can use the protocol Π' to solve $\mathrm{Id}_n \boxplus (g_1,\dots,g_m)$ on the rectangle $W \times W$. By the induction hypothesis (Lemma 14) we know that

$$\begin{aligned}\mathrm{CC}_{W\times W}(\mathrm{Id}_n \boxplus (g_1,\dots,g_m)) &\geq (2-\alpha-2^{-m+1})n - O(\log n)\\ &= (2-1+2^{-m}-2^{-m+1})n - O(\log n) = \alpha n - O(\log n).\end{aligned}$$

Thus, we have $d \geq n-k-3+\alpha n - O(\log n) = (2-2^{-m})n - k - O(\log n)$.

Otherwise, if $\left|\{\vec{x} \mid (h,(\vec{x},y)) \in E_{S'}\}\right| \leq N^{m-\alpha}$ for all $h \in \mathcal{P}_n$ and $y \in \{0,1\}^n$, we apply Lemma 19 to construct a set H of size at least $2^{\Omega(N^\alpha)}$. In this case, the protocol for $\mathrm{Id}_n \boxplus (\vec{g}, \mathrm{Mux})$ on $S' \times S'$ can be used to non-deterministically solve NEQ_H with additive overhead of $O(\log n)$. The reduction from NEQ_H to $\mathrm{Id}_n \boxplus (\vec{g}, \mathrm{Mux})$ is similar to the reduction used in [18].

Let $R_{h_a,h_b} = \{((h_a,\vec{x}_a,y_a),(h_b,\vec{x}_b,y_b)) \in S' \times S' \mid \vec{g} \otimes \vec{x}_a \oplus h_a(y_a) \neq \vec{g} \otimes \vec{x}_b \oplus g_b(y_b)\}$. We consider the following three situations and show that they are the necessary and sufficient conditions for $h_a, h_b \in H$ to be different:

- There are $\vec{x}_a, \vec{x}_b, y_a, y_b$ such that the protocol Π performs non-classical communication in the first $n-k-3$ rounds on input $((h_a,\vec{x}_a,y_a),(h_b,\vec{x}_b,y_b))$. The partial transcript T of the first $n-k-3$ rounds of Π is fixed by Lemma 17, but it does not include an information about who sends each message, so the same transcript can be produced by different rounds. Such a difference can only exists if $h_a \neq h_b$ – for every fixed $h_a = h_b$ the protocol has only classical rounds, and hence a partial transcript uniquely defines who sends in each round.
- The protocol Π performs a non-classical round on some input from R_{h_a,h_b}. If $h_a = h_b$ then Π can only perform classical rounds by the definition of partially half-duplex communication.
- Π performs only classical rounds on some input from R_{h_a,h_b} and outputs $\perp$.

We can argue that one of this conditions is satisfied iff $h_a \neq h_b$. Indeed, suppose that $h_a \neq h_b$. If the first or the second condition is satisfied we are done, so let's assume that it is not. The first $n-k-3$ rounds of Π on inputs from R_{h_a,h_b} are already known, so we can skip them and only consider the rounds of Π after that. We also know that all the next rounds are going to be classical. By construction of H there exists $\vec{x}, y$, such that $(h_a,\vec{x},y)$ and $(h_b,\vec{x},y)$ belong to S', and also $\vec{g} \otimes \vec{x} \oplus h_a(y) \neq \vec{g} \otimes \vec{x} \oplus h_b(y)$. By the definition of $\mathrm{Id}_n \boxplus (g_1,\ldots,g_m,\mathrm{Mux})$ the protocol Π has to output $\perp$, and hence satisfy the third condition.

Now suppose that $h_a = h_b$. Then neither of the conditions could be satisfied. The first condition fails as in this case a partial transcript uniquely defines who sends in each round. The second condition fails by the definition of partially half-duplex protocol. The third one fails by the definition of $\mathrm{Id}_n \boxplus (g_1,\ldots,g_m,\mathrm{Mux})$.

Now we can use this property to solve NEQ_H. Alice and Bob guess which of the condition is satisfied, guess a proof of it, and then verify it.

- To prove the first condition the players guess the difference in the first $n-k-3$ rounds. Verification requires only $\log n$ bits of communication.
- For the second condition the players guess a number $t \in [d-n+k+3]$, a string $s \in \{0,1\}^t$, a number $i \in [n]$, and bits p, q. Then they verify that there exist pairs $(\vec{x}_a, y_a)$ and $(\vec{x}_b, y_b)$ such that:
  - $p = \left(\vec{g} \otimes \vec{x}_a \oplus h_a(y_a)\right)_i \neq \left(\vec{g} \otimes \vec{x}_b \oplus g_b(y_b)\right)_i = 1-p$,
  - both players are consisted with s being an extension of the partial transcript T on inputs $((h_a,\vec{x}_a,y_a),(h_b,\vec{x}_b,y_b))$, meaning that if a player wants to send a bit in some round, this bit is equal to corresponding bit in s,
  - in the next round after the rounds described in s, the protocol Π performs a non-classical round: either both send (in case $q=1$) or both receive (in case $q=0$).

  All together the size of the witness in this case is $d-n+k+O(\log n)$.
- For the third condition the players guess a string $s \in \{0,1\}^{d-n+k+3}$, a number $i \in [n]$, and a bit p. Then they verify that there exist pairs $(\vec{x}_a, y_a)$ and $(\vec{x}_b, y_b)$ such that:
  - $p = \left(\vec{g} \otimes \vec{x}_a \oplus h_a(y_a)\right)_i \neq \left(\vec{g} \otimes \vec{x}_b \oplus g_b(y_b)\right)_i = 1-p$,

- both players are consisted with s being an extension of the partial transcript T on inputs $(h_a, \vec{x}_a, y_a), (h_b, \vec{x}_b, y_b)$, meaning that if a player wants to send a bit in some round, this bit is equal to corresponding bit in s,
- the transcript ends in a leaf labeled with $\perp$.

All together the size of the witness in this case is $d - n + k + O(\log n)$.

This reduction shows that NEQ_H can be non-deterministically solved with a protocol of size $d - n + k + O(\log n)$. Thus, the depth of the protocol Π is at least

$$\begin{aligned} n - k + \mathrm{NCC}(\mathrm{NEQ}_H) - O(\log n) &\geq n - k + \log\log|H| - O(\log n) \\ &\geq n - k + \log N^{\alpha} - O(\log\log(N)) \\ &= n - k + \alpha n - O(\log n), \end{aligned}$$

where NCC stands for non-deterministic communication complexity. ◀

3 Super-cubic lower bound

Now we have all the ingredients to prove a super-cubic lower bound. First of all we need to reformulate the lower bound on $\mathrm{Id}_n \boxplus_m g$ (Theorem 9) in terms of protocol size.

▶ **Lemma 20.** *For all $n, m \in \mathbb{N}$, there exists $g : \{0,1\}^n \to \{0,1\}^n$ such that any communication protocol Π for $\mathrm{Id}_n \boxplus_m g$,*

$$\log_2 \mathrm{L}(\Pi) \geq \frac{(2 - 2^{-m+1})n}{1.73} - O(\log n).$$

To get a protocol size lower bound from a protocol depth lower bound we need a protocol balancing technique. In Appendix B, we show that any De Morgan formula balancing technique that preserves monotonicity can be applied for communication protocols as well. And hence we can use balancing technique by Khrapchenko [16, 12].

For any $m \geq 16$ that gives us $\log_2 L(\Pi) > 1.156n$. So we fix $m = 16$. Now we are going to feed this lower bound into the following variant of Andreev's function.

▶ **Definition 21.** *For all $n, m \in \mathbb{N}$, $n > m \log n$, and functions $f, g : \{0,1\}^{\log n} \to \{0,1\}^{\log n}$* the XOR-composed Andreev's function $\mathrm{Andr}_{n,m}$ *is defined by*

$$\mathrm{Andr}_{n,m}(f, g, x_1, \ldots, x_{m \log n}) = (f \boxplus_m g)\big(\oplus(x_1), \cdots, \oplus(x_{m \log n})\big),$$

where $x_i \in \{0,1\}^{\frac{n}{m \log n}}$ for $i \in [m \log n]$, and $\oplus(x)$ denotes the sum of all bits of x modulo 2.

▶ **Theorem 22.** *Any communication protocol for the generalized Karchmer–Wigderson game for $\mathrm{Andr}_{n,16}$ has size at least $\Omega(n^{3.156}(\log n)^{-7/2}(\log\log n)^{-2})$.*

Note that the input length of $\mathrm{Andr}_{n,m}$ is $\Theta(n \log n)$. It is also important that there is a natural polynomial time algorithm for $\mathrm{Andr}_{n,m}$, so it is an explicit function. The proof of this theorem is almost identical to the original proof of Håstad [7, Theorem 8.1] with only difference that we now hard-wire functions $\mathrm{Id}_{\log n}$ and g provided by Lemma 20. It's not entirely obvious what restrictions mean in terms of communication protocols. In Appendix C, we explain that Håstad's technique, which was used for formulas, can be carried over to the case of communication protocols.

Proof. Assume that we have a protocol of size L for generalized KW game for $\mathrm{Andr}_{n,16}$. We know that there is a function $\mathrm{Id}_{\log n} \boxplus_{16} g$ on $16 \log n$ variables that requires protocol of size at least $n^{1.156}$. We fix the first two inputs to $\mathrm{Andr}_{n,16}$ with the description of $\mathrm{Id}_{\log n}$ and g provided by Lemma 20. This might decrease the size of the protocol, but it is not clear by how much and hence we just note that the resulting protocol is of size at most L.

Apply an R_p-restriction with $p = \frac{32 \log n \log\log n}{n}$ on the protocol. By Theorem 28 the resulting protocol will be of expected size at most $O(n^{-2}(\log n)^{7/2}(\log\log n)^2 L + 1)$. The probability that all variables in a particular group are fixed is bounded by

$$(1-p)^{\frac{n}{16\log n}} \leq e^{-\frac{pn}{16\log n}} \leq (\log n)^{-2}.$$

Since there are only $16 \log n$ groups, with probability $1 - o(1)$ there remains at least one live variable in each group. Now since a positive random variable is at most twice its expected with probability at least $1/2$, it follows that there is a positive probability that we have at most twice the expected remaining size and some live variable in each group. It follows that

$$n^{-2}(\log n)^{7/2}(\log\log n)^2 L \geq \Omega(n^{1.156}).$$

Hence $L \geq \Omega\big(n^{3.156}(\log n)^{-7/2}(\log\log n)^{-2}\big)$. ◀

▶ **Theorem 8.** *There exists a function $f : \{0,1\}^n \to \{0,1\}^{\log n}$ such that any communication protocol for generalized Karchmer–Wigderson game for f has size at least $\tilde{\Omega}(n^{3.156})$.*

Proof. Let $f' = \mathrm{Andr}_{n',16}$ for $n' = \frac{n}{2\log n}$. The input length of f' is $\frac{2n(\log n - \log\log n)+n}{2\log n} < n$. The output length of f' is $\log n - \log\log n - 1 < \log n$. Let $f : \{0,1\}^n \to \{0,1\}^{\log n}$ be a function obtained from f' by adding the appropriate number of dummy input and output bits. By Theorem 22 any protocol for generalized Karchmer–Wigderson game for f has size at least $\Omega\big(n^{3.156}(\log n)^{-6.656}(\log\log n)^{-2}\big) = \tilde{\Omega}(n^{3.156})$. ◀

4 Conclusion

We hope that our approach can provide an insight for proving better lower bounds on the original Karchmer–Wigderson games, and hence for proving new lower bounds on De Morgan formula size. We propose the following list of open problems.

- Show a better lower bound for block-composition of a universal relation and some function. In [18], the special case of Theorem 9 of $m = 2$ was used to show $1.5n - O(\log n)$ lower bound on $\mathrm{U}_n \diamond f_n$. Is it possible to show a better lower bound from Theorem 9 with $m > 2$?
- Can we show nontrivial lower bounds for generalized Karchmer–Wigderson games for functions from $\{0,1\}^n \to \{0,1\}^m$ for all m? For $m = o(\log n)$ we can not prove $n^{3+\varepsilon}$ bound without proving the same kind of bound for formula size, so that might be a bit too ambitious. For $m = \alpha \log n$ for $\alpha \leq 1$ one can adapt the proof from this paper to get a bound of the form $n^{3+O(\alpha)}$. But for $m = \alpha \log n$ for large enough α the best lower bound we know is just m. Is it possible to show a better bound?
- Show n^4 lower bound for generalized Karchmer–Wigderson games for function from $\{0,1\}^n \to \{0,1\}^{\log n}$. The reason for presented lower bound being $\tilde{\Omega}(n^{3.156})$ is that we balance the protocol to get size lower bound from depth lower bound. If one can avoid this step and get lower bounds for size directly, the lower bound will grow up greatly.
- Are there interesting upper and lower bounds for generalized Karchmer–Wigderson outside of the scope of KRW conjecture? It looks that in this setting in might be possible to develop new approaches that might turn to be useful to prove formula lower bounds.

References

1 Yuriy Dementiev, Artur Ignatiev, Vyacheslav Sidelnik, Alexander Smal, and Mikhail Ushakov. New bounds on the half-duplex communication complexity. In *SOFSEM 2021: Theory and Practice of Computer Science - 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25-29, 2021, Proceedings*, volume 12607 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2021. doi:10.1007/978-3-030-67731-2_17.

2 Irit Dinur and Or Meir. Toward the KRW composition conjecture: Cubic formula lower bounds via communication complexity. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 3:1–3:51. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.3.

3 Jeff Edmonds, Russell Impagliazzo, Steven Rudich, and Jirí Sgall. Communication complexity towards lower bounds on circuit depth. *Comput. Complex.*, 10(3):210–246, 2001. doi:10.1007/s00037-001-8195-x.

4 Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 89–98. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.19.

5 Anna Gál. A characterization of span program size and improved lower bounds for monotone span programs. *Comput. Complex.*, 10(4):277–296, 2001. doi:10.1007/s000370100001.

6 Dmitry Gavinsky, Or Meir, Omri Weinstein, and Avi Wigderson. Toward better formula lower bounds: The composition of a function and a universal relation. *SIAM J. Comput.*, 46(1):114–131, 2017. doi:10.1137/15M1018319.

7 Johan Håstad. The shrinkage exponent of de morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. doi:10.1137/S0097539794261556.

8 Johan Håstad and Avi Wigderson. Composition of the universal relation. In Jin-Yi Cai, editor, *Advances In Computational Complexity Theory, Proceedings of a DIMACS Workshop, New Jersey, USA, December 3-7, 1990*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 119–134. DIMACS/AMS, 1990. URL: http://dimacs.rutgers.edu/Volumes/Vol13.html, doi:10.1090/dimacs/013/07.

9 Kenneth Hoover, Russell Impagliazzo, Ivan Mihajlin, and Alexander V. Smal. Half-duplex communication complexity. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 10:1–10:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.10.

10 Artur Ignatiev, Ivan Mihajlin, and Alexander Smal. Super-cubic lower bound for generalized karchmer-wigderson games. *Electron. Colloquium Comput. Complex.*, TR22-016, 2022. URL: https://eccc.weizmann.ac.il/report/2022/016, arXiv:TR22-016.

11 Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n - o(n) for boolean circuits. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002. doi:10.1007/3-540-45687-2_29.

12 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

13 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5(3/4):191–204, 1995. doi:10.1007/BF01206317.

14 Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 539–550. ACM, 1988. doi:10.1145/62212.62265.

15 Valeriy Mihailovich Khrapchenko. Complexity of the realization of a linear function in the class of II-circuits. *Mathematical Notes of the Academy of Sciences of the USSR*, 9(1):21–23, 1971.

16 Valeriy Mihailovich Khrapchenko. On a relation between the complexity and the depth of formula. *Methods of Discrete Analysis in Synthesis of Control Systems*, 32:76–94, 1978.

17 Jiatu Li and Tianqi Yang. 3.1n - o(n) circuit lower bounds for explicit functions. *Electron. Colloquium Comput. Complex.*, page 23, 2021. URL: `https://eccc.weizmann.ac.il/report/2021/023`, `arXiv:TR21-023`.

18 Ivan Mihajlin and Alexander Smal. Toward better depth lower bounds: The XOR-KRW conjecture. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 38:1–38:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CCC.2021.38`.

19 Alexander A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Comb.*, 10(1):81–93, 1990. `doi:10.1007/BF02122698`.

20 Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

21 Bella Abramovna Subbotovskaya. Realization of linear functions by formulas using $\wedge$, $\vee$, $\neg$. In *Doklady Akademii Nauk*, volume 136-3, pages 553–555. Russian Academy of Sciences, 1961.

A Proof of Technical Lemma

▶ **Lemma 19.** *Let $S \subseteq \mathcal{X}_m$ be a subset of inputs such that $|S| \geq N^m \cdot N!$, and let $G_S = (U_S, V_S, E_S)$ be a domain graph of S. If $\min_{h \in U_S}\{\deg_{G_S}(h)\} \geq 4N^m$ and*

$$\forall h \in \mathcal{P}_n,\ \forall y \in \{0,1\}^n,\ \left|\{\vec{x} \mid (h,(\vec{x},y)) \in E_S\}\right| \leq N^{m-\alpha} \tag{1}$$

for some $\alpha > 0$, then there is a set $H \subseteq U_S$ of size $2^{\Omega(N^\alpha)}$ such that for all distinct $h_1, h_2 \in H$, there exist $(\vec{x}, y)$: $(h_1, \vec{x}, y) \in S$, $(h_2, \vec{x}, y) \in S$, and $h_1(y) \neq h_2(y)$.

Proof. We are going to construct a rooted tree $T(S)$ such that

- each leaf ℓ is labeled with a set of functions $F_\ell \subseteq U_S$,
- each internal node v is labeled with a pair $(\vec{x}_v, y_v) \in V_S$,
- for every leaf ℓ labeled with F_ℓ and every its ancestor labeled with $(\vec{x}, y)$ there exists $a \in \{0,1\}^n$ such that $\forall h \in F_\ell$, $h(y) = a$ and $(h, \vec{x}, y) \in S$.
- for every two leaves labeled with F_1 and F_2, and their lowest common ancestor labeled with $(\vec{x}, y)$: $F_1 \cap F_2 = \emptyset$ and for all $h_1 \in F_1$, $h_2 \in F_2$, such that $h_1(y) \neq h_2(y)$,
- the number of leaves is a least $\frac{3^{N^\alpha}}{N}$.

Having such a tree, the set H is constructed by taking one function from every leaf. Indeed, the structure of the tree guarantees that for every $h_1, h_2 \in H$, $h_1 \neq h_2$, there exist $(\vec{x}, y)$, the label of the least common ancestor of corresponding leaves, such that $(h_1, \vec{x}, y) \in S$, $(h_2, \vec{x}, y) \in S$, and $h_1(y) \neq h_2(y)$.

The tree is defined recursively. For a set $Z \subseteq S$, let $T(Z)$ be a (non-empty) rooted tree. Let $G_Z = (U_Z, V_Z, E_Z)$ be a domain graph of Z. If $\min_{h \in U_Z}\{\deg_{G_Z}(h)\} \geq 2N^{m-1}$ then the rooted tree $T(Z)$ consists of a root node labeled with $(\vec{x}_Z, y_Z)$, where $(\vec{x}_Z, y_Z)$ is a vertex of maximal degree in V_Z, and a set of subtrees – for every $a \in \{0,1\}^n$ such that $\exists h \in U_Z : (h, \vec{x}_Z, y_Z) \in Z$, $h(y_Z) = a$ there is a subtree $T(Z_a)$ attached to the root node, where

$$Z_a = \{(h, \vec{x}, y) \mid (h, \vec{x}, y) \in Z, y \neq y_Z, h(y_Z) = a\}$$

Otherwise $T(Z)$ consists of one leaf node labeled with U_Z.

We are going to lower bound the number of leaves in $T(S)$ by lower bounding the number of nodes at depth $N^\alpha+1$. Let z be some node of $T(S)$ at depth $d \le N^\alpha$ labeled with $(\vec{x}_Z, y_Z)$ that corresponds to a root node of a subtree $T(Z)$ for some $Z \subseteq S$. Let $G_Z = (U_Z, V_Z, E_Z)$ be a domain graph of Z. Due to the condition (1) the minimal degree of vertices in U_Z can be lower bounded by $4N^m - dN^{m-\alpha} \ge 3N^m$. At the same time $|V_Z| \le N(N-d)$. Let $T(Z_{a_1}), \dots, T(Z_{a_k})$ – be the subtrees attached to z. Note that $\pi_1(Z_{a_i}) \cap \pi_1(Z_{a_j}) = \emptyset$ for all $i \ne j$, so the number of functions appearing in $Z_{a_1}, \dots, Z_{a_k}$ is exactly the number of functions in Z defined on $(\vec{x}_Z, y_Z)$. Given that (x_Z, y_Z) is a vertex of maximal degree in V_Z, the number of functions in the subtrees can be lower bounded as follows,

$$\left|\pi_1(Z_{a_1}) \sqcup \dots \sqcup \pi_1(Z_{a_k})\right| \ge \frac{|E_Z|}{|V_Z|} \ge \frac{3N^m|U_Z|}{N^m(N-d)} = \frac{3|U_Z|}{N-d}.$$

Thus by induction the total number of functions that appear in the sets at depth $d+1$ is at least

$$\frac{3^d \cdot |U_S|}{N(N-1)\cdots(N-d)} = \frac{3^d \cdot |U_S| \cdot (N-d-1)!}{N!},$$

where the size of U_S is at least $|S|/N^{m+1} \ge N!/N$. Now we are ready to lower bound the number of nodes at depth $d+1$. Note that the number of permutations with k values fixed is $(N-k)!$, and hence a node at depth $d+1$ has at most $(N-d-1)!$ functions in its set. The number of nodes at depth $d+1$ is at least the total number of functions at depth $d+1$ divided by the upper bound on the number of functions in one node, that is

$$\frac{3^d \cdot |U_S| \cdot (N-d-1)!}{N!} / (N-d-1)! \ge \frac{3^d}{N}.$$

For $d = N^\alpha + 1$ we get the desired lower bound $\frac{3^{N^\alpha}}{N} = 2^{\Omega(N^\alpha)}$ on the number of leaves. ◀

B Protocol balancing

In this section we show that any formula balancing technique that preserves monotonicity can be also used for communication protocols. As De Morgan's formulas balancing methods are well studied, it is tempting to apply it to arbitrary communication protocols as a black-box. Let P be an arbitrary communication problem. We start by showing that every communication protocol Π for P can be viewed as a communication protocol solving *the monotone Karchmer–Wigderson game* for some monotone function f_Π defined by Π, so it can be syntactically transformed into a monotone formula ϕ_Π computing f_Π. Due to Karchmer-Wigderson theorem, any *monotone* formula ψ for f_Π can be syntactically transformed into a protocol Π_ψ solving P and having the same underlying tree. Thus we can convert a protocol for P into a monotone formula, balance it using a technique preserving monotonicity, and then convert it back into a new (balanced) protocol for the original problem P.

▶ **Definition 23.** The monotone Karchmer–Wigderson game (monotone KW game) *for monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ is the following communication problem: Alice gets an input $x \in \{0,1\}^n$ such that $f(x) = 0$, and Bob gets as input $y \in \{0,1\}^n$ such that $f(y) = 1$. Their goal is to find a coordinate $i \in [n]$ such that $x_i < y_i$. The monotone KW game corresponds to a communication problem for* the monotone Karchmer–Wigderson relation for *f:*

$$\mathrm{mKW}_f = \{(x, y, i) \mid x, y \in \{0,1\}^n, i \in [n], f(x) = 0, f(y) = 1, x_i < y_i\}.$$

The following transformation of an arbitrary communication problem into a monotone KW relation is folklore [19, 5]. Let $P \subset X \times Y \times Z$ be any communication problem, and let Π be a communication protocol solving P with s leaves $l_1, \dots, l_s$. For every $x \in X$, let $a(x) \in \{0,1\}^s$ be such that $a(x)_i = 0 \iff \exists y \in Y : (x,y) \in R_i$, where $R_i \subset X \times Y$ is the combinatorial rectangle of inputs corresponding to the leaf l_i. Similarly, for every $y \in Y$, let $b(y) \in \{0,1\}^s$ such that $b(y)_i = 1 \iff \exists x \in X : (x,y) \in R_i$.

▶ **Lemma 24** (Folklore). *For every pair of inputs $(x,y) \in X \times Y$ the protocol Π terminates in a leaf l_i if and only if $a(x)_i < b(y)_i$ and $a(x)_j \geq b(y)_j$ for all $j \neq i$.*

Proof. Every $(x,y) \in X \times Y$ belongs to exactly one leaf rectangle of the protocol Π. Transcript $\Pi(x,y)$ terminates in a leaf l_i if and only if $(x,y) \in R_i$. By definition of a and b and since R_i is a combinatorial rectangle, $a(x)_i = 0$ and $b(y)_i = 1$ is equivalent to $(x,y) \in R_i$. For all $j \neq i$, $(x,y) \notin R_j$, and hence $a(x)_j \geq b(y)_j$. ◀

Now consider a De Morgan formula $\phi_\Pi(a_1, \dots, a_s)$ that is syntactically constructed from Π as in Karchmer–Wigderson theorem. Let $f_\Pi : \{0,1\}^s \to \{0,1\}$ be the function computed by ϕ_Π. Function f_Π has the following properties:

- $\forall x \in X,\ f_\Pi(a(x)) = 0$,
- $\forall y \in Y,\ f_\Pi(b(y)) = 1$.

Moreover, by Lemma 24 for any $x \in X$ and $y \in Y$ there is always unique $i \in [s]$ such that $a(x)_i < b(y)_i$, hence the output of any protocol for mKW_{f_Π} on $(a(x), b(y))$ coincides with the output of the protocol Π on (x,y).

Let $\psi(a_1, \dots, a_s)$ be a monotone formula for f_Π. Consider a communication protocol for mKW_{f_Π} obtained from ψ via Karchmer–Wigderson theorem. Let a and b be the inputs of the players following this protocol. Suppose that the players reached a leaf labelled with some i. The monotonicity of ψ guarantees that in this case $a_i < b_i$.

Consider the following protocol Π' for the original communication problem P. Given $(x,y) \in X \times Y$, Alice and Bob simulate the protocol Π_ψ for mKW_f on $(a(x), b(y))$. Let i be the output of Π_ψ on $(a(x), b(y))$. Alice and Bob outputs the label of the leaf l_i of Π.

▶ **Lemma 25.** *Π' is a correct protocol for P.*

Proof. By Lemma 24, for every $(x,y) \in X \times Y$ there is always a unique index i such that $a(x)_i < b(y)_i$. At the same time, any protocol solving mKW_f must output i such that $a(x)_i < b(y)_i$. So, for all $(x,y) \in X \times Y$ the outputs of Π and Π' coincide. ◀

Now we are ready to prove the main theorem of this section.

▶ **Theorem 26.** *For every communication problem P with a protocol of size L*

$$\mathrm{CC}(P) \leq 1.73 \log_2 L.$$

Proof. We start with a protocol Π for P with L leaves, transform it to a formula ϕ_Π of the same size, balance it using the formula balancing technique by Khrapchenko [16, 12], get some formula ψ, and finally construct a protocol Π' of depth at most $1.73 \log_2 L$ based on ψ. By Lemma 25 the protocol Π' is a correct protocol for communication problem P. ◀

Now we can use this theorem to prove Lemma 20.

▶ **Lemma 20.** *For all $n, m \in \mathbb{N}$, there exists $g : \{0,1\}^n \to \{0,1\}^n$ such that any communication protocol Π for $\mathrm{Id}_n \boxplus_m g$,*

$$\log_2 \mathrm{L}(\Pi) \geq \frac{(2 - 2^{-m+1})n}{1.73} - O(\log n).$$

Proof. Apply Theorem 26 to the statement of Theorem 9. ◀

C Restrictions for generalized Karchmer–Wigderson games

In this section we show that we can adapt random restriction technique for communication protocols for generalized Karchmer–Wigderson games. In the seminal paper [7], Håstad analyzes the expected size of a formula after it has been hit with a random restriction. A *restriction* for a formula on n variables is an element of $\{0,1,*\}^n$. For $p \in [0,1]$ a random restriction ρ from R_p is chosen by that we set randomly and independently each variable to $*$ with probability p and $0,\ 1$ with equal probabilities $\frac{1-p}{2}$. The interpretation of giving a value $*$ to a variable is that it remains a variable, while in the other cases the given constant is substituted as the value of the variable. The Main Shrinkage Theorem [7, Theorem 7.1] bounds the expected size of the resulting formula.

▶ **Theorem 27** (Theorem 7.1 in [7]). *Let ϕ be a formula of size L and ρ a random restriction in R_p. Then the expected size of $\phi\restriction_\rho$ is bounded by*

$$O\left(p^2\left(1+(\log(\min(1/p,L)))^{3/2}\right)L + p\sqrt{L}\right).$$

We want to argue that exactly the same reasoning can be applied to general Karchmer–Wigderson games, and hence we get the following theorem.

▶ **Theorem 28.** *Let Π be a protocol for generalized Karchmer–Wigderson game of size L and ρ a random restriction in R_p. Then expected size of $\Pi\restriction_\rho$ is bounded by*

$$O\left(p^2\left(1+(\log(\min(1/p,L)))^{3/2}\right)L + p\sqrt{L}\right).$$

If we were talking about regular Karchmer–Wigderson games then we would be able to say that this theorem is an immediate corollary of Theorem 27 due to Karchmer–Wigderson correspondence between protocols and formulas. For generalized Karchmer–Wigderson games the situation is a little bit trickier: we still can syntactically translate a protocol into a formula, but it is unclear how the resulting formula is related to the (multioutput) function in the protocol. E.g., if we construct a formula in this way for a naive protocol solving generalized Karchmer–Wigderson game for Id_n, then it will be a formula for a constant function.

▶ Remark. A generalized Karchmer–Wigderson game for a function f is a communication problem with *promise* – players are promised that $f(x) \neq f(y)$. It is possibility that there is a leaf in a protocol for KW_f with label i such that for some pair of inputs in this leaf $x_i = 0$ and $y_i = 1$ and for other pair of inputs the situation is opposite, so $x_i = 1$ and $y_i = 0$. That can not happen in non-promise communication problems due to the fact that all inputs corresponding to a node of a protocol form a combinatorial rectangle. Every protocol for KW_f can be modified such that every leaf with label i contains input pairs of only one of these types, and the size of the protocol increases no more than twice. Therefore, we assume that all protocols in this section have this property.

First of all we need to define how restrictions affect communication protocols for generalized Karchmer–Wigderson games. Let $f : \{0,1\}^n \to \{0,1\}^r$ be a non-constant function and Π be a protocol for KW_f. The protocol Π is defined on all pair of inputs in $X = \{(x,y) \mid x,y \in \{0,1\}^n, x \neq y\}$. When a protocol gets hit with a restriction ρ the set of possible input pairs gets narrowed to

$$X\restriction_\rho = \{(x,y) \mid x,y \in \{0,1\}^n, x \neq y, \forall i : \rho(x_i) \neq * \implies x_i = y_i = \rho(x_i)\}.$$

After that some of the nodes of Π become unreachable and can be eliminated. I.e., if $\rho(x_i) \neq *$ for some i then all the leaves labeled with i become unreachable and can be eliminated. In [7], Håstad considers the following list of simplifications.

- If one input to a $\vee$-gate ($\wedge$-gate) is given the value 0 (value 1) we erase this input and let the other input of this gate take the place of the output of the gate.
- If one input to a $\vee$-gate ($\wedge$-gate) is given the value 1 (value 0) we replace the gate by the constant 1 (constant 0).
- If one input of a $\vee$-gate ($\wedge$-gate) is reduced to the single literal $x_i/\bar{x}_i$ then $x_i = 0/x_i = 1$ ($x_i = 1/x_i = 0$) is substituted in the formula giving the other input to this gate. If possible we do further simplifications in this subformula.

All these simplifications of De Morgan formula can be reformulated in terms of the corresponding communication protocol for Karchmer–Wigderson game. We want to say that exactly the same can be done for communication protocols for generalized Karchmer–Wigderson games as they are syntactically indistinguishable (it is important here that every leaf of a protocol correspond to a literal). Thus, we conclude that Theorem 28 holds.

The Dispersive Art Gallery Problem

Christian Rieck
Department of Computer Science, TU Braunschweig, Germany

Christian Scheffer
Faculty of Electrical Engineering and Computer Science, Bochum University of Applied Sciences, Germany

Abstract

We introduce a new variant of the art gallery problem that comes from safety issues. In this variant we are not interested in guard sets of smallest cardinality, but in guard sets with largest possible distances between these guards. To the best of our knowledge, this variant has not been considered before. We call it the Dispersive Art Gallery Problem. In particular, in the dispersive art gallery problem we are given a polygon $\mathcal{P}$ and a real number ℓ, and want to decide whether $\mathcal{P}$ has a guard set such that every pair of guards in this set is at least a distance of ℓ apart.

In this paper, we study the vertex guard variant of this problem for the class of polyominoes. We consider rectangular visibility and distances as geodesics in the L_1-metric. Our results are as follows. We give a (simple) thin polyomino such that every guard set has minimum pairwise distances of at most 3. On the positive side, we describe an algorithm that computes guard sets for simple polyominoes that match this upper bound, i.e., the algorithm constructs worst-case optimal solutions. We also study the computational complexity of computing guard sets that maximize the smallest distance between all pairs of guards within the guard sets. We prove that deciding whether there exists a guard set realizing a minimum pairwise distance for all pairs of guards of at least 5 in a given polyomino is NP-complete.

We were also able to find an optimal dynamic programming approach that computes a guard set that maximizes the minimum pairwise distance between guards in tree-shaped polyominoes, i.e., computes optimal solutions; due to space constraints, details can be found in the full version of our paper [42]. Because the shapes constructed in the NP-hardness reduction are thin as well (but have holes), this result completes the case for thin polyominoes.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Art gallery, dispersion, polyominoes, NP-completeness, r-visibility, vertex guards, L_1-metric, worst-case optimal

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.67

Related Version *Full Version*: `https://arxiv.org/abs/2209.10291` [42]

Acknowledgements We thank Joseph S.B. Mitchell for bringing this problem to our attention.

1 Introduction

How many guards are necessary to guard an art gallery? This question was first posed by Victor Klee in 1973 and opened a flourishing field of research in computational geometry; see for example the book by O'Rourke [41], or the surveys by Shermer [44], and Urrutia [46]. This question states the classic Art Gallery Problem as follows: Given a (simple) polygon $\mathcal{P}$ and an integer k, decide whether there is a guard set of cardinality k such that every point $p \in \mathcal{P}$ is seen by at least one guard, where a point is seen by a guard if and only if the connecting line segment is inside the polygon.

Suppose the following situation: Your art gallery is the victim of a robbery, or there is a fire outbreak and heavy smoke development in one part of the building. Because guards in an optimal solution to instances of the classic art gallery problem can be really close together, many cameras can be affected at the same time, see Figure 1. From safety and

security issues this would be a catastrophic scenario. We want to address these issues, i.e., for a given shape, we are interested in a guard set that realizes preferably large distances between any two guards of the respective set, rather than focusing on the minimum number of guards needed. Problems of this kind are called Dispersion Problems, and are typically stated as follows: Given a set of n objects in the plane and an integer k, decide if there is a subset of k such objects, such that the distances between any pair in this subset is at least as large as a given threshold. We assume that the shortest paths that realize the distances between guards are within the shape, i.e., they do not leave and enter the shape.

In this paper, we introduce the following problem that combines art gallery and dispersion problems and is described as follows.

Dispersive Art Gallery Problem. Given a polygon $\mathcal{P}$ and a real number ℓ, decide whether there exists a guard set $\mathcal{G}$ for $\mathcal{P}$ such that the pairwise geodesic distances between any two guards in $\mathcal{G}$ are at least ℓ.

Note that in this problem we are not interested in the size of a particular guard set, but only in the distances between guards realized by the guard set. To the best of our knowledge, this problem has not been considered before. Additionally, a first intuitive thought might be that solutions to the classic art gallery problem are also solutions to this variant, since small cardinality guard sets should somehow yield larger pairwise distances. However, this is nowhere near the truth, see for example Figure 1 where doubling the size of the guard set results in an arbitrary growth of the dispersion distance.

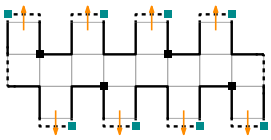


Figure 1 An adaption of the comb-like polyomino. The black vertices realize an optimal guard set for the classic AGP, while the dark cyan set is optimal for the dispersive AGP.

1.1 Our contributions

In this paper, we introduce the dispersive art gallery problem and investigate it for vertex guards in polyominoes, i.e., orthogonal polygons whose vertices have integer coordinates.

- We describe a (simple) thin polyomino where the minimum pairwise distance between any two guards in every feasible guard set is at most 3.
- We give a worst-case optimal algorithm for placing a set of guards at the vertices of a simple polyomino such that the pairwise distances between any two guards are at least 3.
- It is NP-complete to decide whether a pairwise distance of at least 5 can be guaranteed.
- We describe a dynamic programming approach that computes a guard set that maximizes the minimum pairwise distance between any two guards for tree-shaped polyominoes.

1.2 Previous work

The famous question from Klee was answered relatively quickly by Chvátal [17]. Not least because of the beautiful proof from Fisk [28] it is almost common knowledge that $\lfloor n/3 \rfloor$ guards are sufficient but sometimes necessary to monitor a simple polygonal region with n edges. Through their typical orthogonality, "traditional" galleries actually require less

guards, i.e., for orthogonal polygons with n vertices already $\lfloor n/4 \rfloor$ guards are sufficient, but also sometimes necessary [30, 35, 40]. However, finding the optimal solution even in simple polygons is proven to be NP-hard by Lee and Lin [37], and by Schuchardt and Hecker [43] for simple orthogonal polygons. In the special case of r-visibility, computing the minimum guard set is polynomial in orthogonal polygons [11, 47]. More recently, Abrahamsen et al. [2, 3] first showed that irrational guards are sometimes needed in an optimal guard set (in general and orthogonal polygons), and subsequently that the art gallery problem is actually $\exists\mathbb{R}$-complete.

Restricting the class of galleries to polyominoes intuitively makes the problem a lot easier. However, as shown by Biedl et al. [8, 9] the problem remains NP-hard. On the positive side they showed that $\lfloor (m+1)/3 \rfloor$ point guards are always sufficient and sometimes necessary, where m is the number of squares of the polyomino. Additionally, they give an algorithm for computing optimal guard sets in the case of thin polyomino trees.

By now, there are many variations of the classic art gallery problem. At least in two of them the number of placed guards is irrelevant, as it is also the case in our problem setting. These are the Chromatic AGP [21, 22, 25, 33] where guards are associated by a color and no two guards of the same color class are allowed to have overlapping visibility regions, and the Conflict-free chromatic AGP [4, 5, 31] in which the overlapping constraint is relaxed in a way that at every point within the polygon a unique color must be visible. In both of these problems, only the number of used colors in a feasible guard set is of interest.

Other variations regard the region that has to be covered, e.g., the Terrain Guarding Problem [12, 36], or problems that arrive from restricting the visibility of the guards to cones of a certain angle, that can be summarized under the generic term of Floodlight Problems [1, 13, 18, 24, 32, 39, 45].

Dispersion problems are related to packing problems and involve arranging a set of objects "far away" from one another, or choosing a subset of objects that are "far apart". These naturally arrive as obnoxious facility location problems (see, e.g., the surveys by Cappanera [15], or Erkut and Neuman [23]), and as problems of distant representatives [27]. For more recent work in many different settings, e.g., in disks [20, 27], or on intervals [10, 38]; see also [6, 7, 14, 16, 26, 29, 34] for various other settings.

1.3 Preliminaries

We consider *polyominoes*, that are orthogonal polygons formed by joining unit squares edge to edge. These unit squares are called *cells*, and the edges of the cells are denoted as *sides*. The *boundary* $\partial\mathcal{P}$ of the polyomino $\mathcal{P}$ is the sequence of all cell sides each one lying between one cell from $\mathcal{P}$ and one cell not being part of $\mathcal{P}$. The *vertices* of a polyomino $\mathcal{P}$ are the vertices of the boundary of $\mathcal{P}$. A point $p \in \mathcal{P}$ *covers* or *sees* another point $q \in \mathcal{P}$ if there is an axis-aligned rectangle defined by p and q that is a subset of $\mathcal{P}$. In the literature this notion of visibility is called *r-visibility*. The area that is visible from a point p is its *visibility region* $\mathcal{V}(p)$. The *distance* $d(p, q)$ between two points $p, q \in \mathcal{P}$ is given by the L_1 geodesic shortest path connecting these two points, i.e., the distance is measured entirely within the interior of $\mathcal{P}$. A *guard set* $\mathcal{G}$ is a set of points of $\mathcal{P}$ such that every point of $\mathcal{P}$ is covered by at least on point of $\mathcal{G}$. We will restrict ourselves to *vertex guards*, i.e., guards that are placed on vertices of $\mathcal{P}$. The minimum over all pairwise distances between any two guards in a guard set $\mathcal{G}$ is called its *dispersion distance*. The *dual graph* of a polyomino $\mathcal{P}$ has a vertex for every cell of $\mathcal{P}$, and edges between vertices if their corresponding cells share a side. We say that a polyomino is *simple* if it has no holes, *thin* if it does not contain a 2×2 polyomino as a subpolyomino, and *tree-shaped* if its dual graph is a tree. We call a cell a *niche* if it is a degree 1 vertex in the dual graph of $\mathcal{P}$.

2 Worst-case optimality

In this section we prove that a dispersion distance of 3 is worst-case optimal for simple polyominoes. In particular, we describe the construction of thin polyominoes for which no guard set can have larger dispersion distance than 3, and describe an algorithm that constructs such guard sets for any simple polyomino.

▶ **Lemma 1.** *There are (simple) thin polyominoes such that every guard set has dispersion distance at most 3.*

Proof. Consider the dark magenta region in Figure 2. Note that this region has to be guarded by a guard g that is placed on one of the four vertices incident to this region. Let Π be one of the four niches that is closest to g. The guard g' that covers Π has distance at most 3 to g. ◀

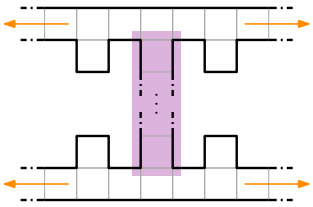


Figure 2 A simple, thin polyomino in that every guard set has dispersion distance at most 3.

Note that the polyomino given in Figure 2 can be used as a "building block", i.e., it can be extended (as indicated by orange arrows) and therefore be used to construct arbitrarily large polyominoes (as well as non-simple ones) in which the same upper bound holds.

In the remainder of this section we show that every *simple* polyomino allows for a guard set with a dispersion distance of at least 3, implying worst-case optimality.

▶ **Theorem 2.** *For every simple polyomino there exists a guard set that has dispersion distance at least 3.*

In particular, we prove Theorem 2 constructively by giving an algorithm that constructs a guard set with dispersion distance of at least 3 in polynomial time. In a nutshell, the algorithm places guards greedily until the whole polyomino is guarded. The algorithm starts with a guard on an arbitrary vertex. Then the region that is visible from this guard is removed from the polyomino. This leads to a set of disjoint subpolyominoes that are guarded recursively, maintaining a distance of at least 3 between any two guards, see Figure 3.

2.1 Preliminaries for the algorithm

Let $\mathcal{P}'$ be a subpolyomino of $\mathcal{P}$, i.e., $\mathcal{P}' \subseteq \mathcal{P}$. The boundary $\partial\mathcal{P}'$ of $\mathcal{P}'$ is the union of all sides being part of exactly one cell from $\mathcal{P}'$. Note that the definition of $\partial\mathcal{P}'$ does not depend on $\mathcal{P}$. Assume that the guard g cannot see the entire polygon $\mathcal{P}$, i.e., $\mathcal{V}(g) \neq \mathcal{P}$. By removing $\mathcal{V}(g)$ from $\mathcal{P}$ we obtain $k \geq 1$ subpolyominoes $\mathcal{P}_1, \ldots, \mathcal{P}_k \subset \mathcal{P}$, being maximal subsets of unit squares such that each subset forms an orthogonal polygon. The *gate* G_i corresponding to $\mathcal{P}_i$ is $\partial\mathcal{V}(g) \cap \partial\mathcal{P}_i$. Without loss of generality we assume $G_1, \ldots, G_k$ to be ordered clockwise on $\partial\mathcal{P}$ starting from g. The *walls* of a gate G_i are the two sides from $\partial\mathcal{P} \setminus G_i$ being adjacent to G_i, see the red segments in Figure 5. Note that the first (second) wall of G_i can lie on $\partial\mathcal{P}_{i-1}$ ($\partial\mathcal{P}_{i+1}$) where from now on the indices $i+1$ and $i-1$ are considered modulo k.

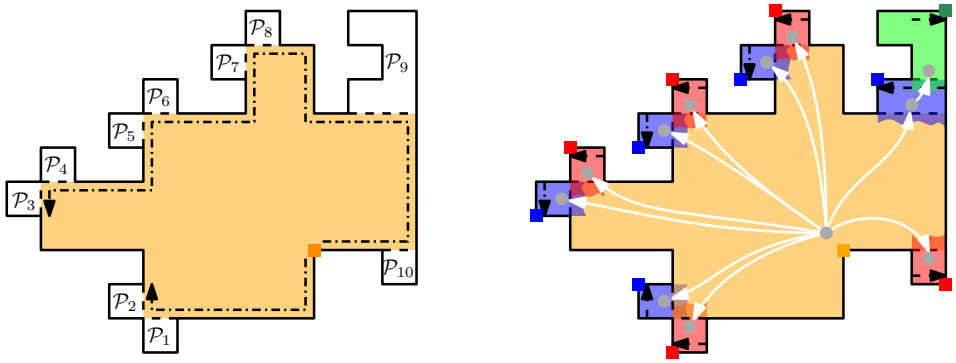


Figure 3 (a) A guard with its visibility region (orange), and the subpolyominoes $\mathcal{P}_1, \ldots, \mathcal{P}_{10}$. The corresponding gates G_1 and G_2 are clockwise, and $G_3, \ldots, G_{10}$ are counterclockwise. (b) The recursion tree T and a guarding computed by our algorithm.

2.2 Description of the algorithm

Based on the preliminaries above we provide the details of our algorithm. As initialization, we consider a guard g placed on an arbitrary vertex of the given polyomino $\mathcal{P}$.

A recursion step. Consider the subshapes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ and the corresponding gates as defined above, see Figure 3(a). Let α and β be the number of sides from $\partial\mathcal{P}$ when walking clockwise along $\partial\mathcal{P}$ from g to G_1, and from G_k to g, respectively. Note that $\alpha, \beta \geq 1$. In the following we declare each gate to be *(oriented) clockwise* or *counterclockwise*. For this, we consider different cases regarding k, see Figure 4.

(1) If $k = 1$:
- **(1.1)** if $\alpha = 1$, we declare G_1 to be clockwise
- **(1.2)** otherwise, we declare G_1 to be counterclockwise.

(2) If $k = 2$:
- **(2.1)** if $\alpha = 1 = \beta$, we declare G_1 to be clockwise and G_2 to be counterclockwise,
- **(2.2)** if $\alpha = 1 < \beta$, we declare G_1 and G_2 to be clockwise,
- **(2.3)** if $\alpha > 1 = \beta$, we declare G_1 and G_2 to be counterclockwise.

(3) If $k \geq 3$, let G_ℓ be the first gate being not adjacent to its successor $G_{\ell+1}$, i.e., G_ℓ and $G_{\ell+1}$ are not sharing an endpoint. We declare $G_1, \ldots, G_\ell$ to be clockwise and $G_{\ell+1}, \ldots, G_k$ to be counterclockwise, see Figure 3(a).

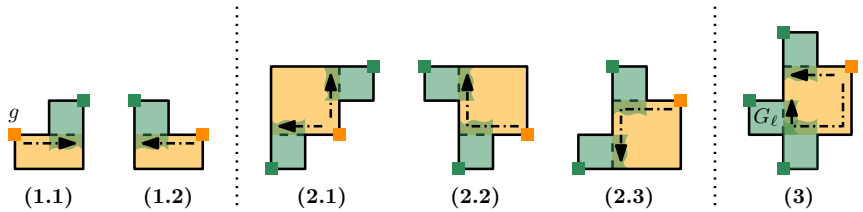


Figure 4 Case distinction for gate orientations (orange: guard g, green: guard $\overline{g}$ placed after g and whose position is influenced by orientation of corresponding gate).

For each $\mathcal{P}_i$ we make a recursive call for covering $\mathcal{P}_i$ separately. In particular, we make the following distinction: A gate is *parallel* (*orthogonal*) when its walls lie parallel (orthogonal) to each other.

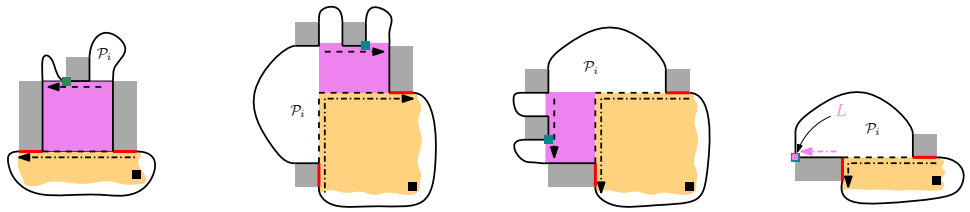


Figure 5 Placements of a guard $\overline{g}$ (green) depending on the gate's type and orientation (dashed line): (a) A parallel counterclockwise gate. (b) An orthogonal clockwise gate. (c) An orthogonal counterclockwise gate. (d) An orthogonal counterclockwise gate, and L degenerates to a single point.

A recursive call for a parallel gate. Without loss of generality, we assume that G_i is horizontal and $\mathcal{V}(g)$ lies below G_i, see Figure 5(a). Let $T \subseteq \mathcal{P}_i$ be the axis aligned rectangle with maximal height and bottom side G_i. If G_i is clockwise (counterclockwise), we choose the guard $\overline{g}$ as an arbitrary vertex on the boundary of T not lying on G_i and not lying on the left (right) side of T. Finally, we recurse on $\mathcal{P} := \mathcal{P}_i \cup \mathcal{V}(\overline{g})$ and $g := \overline{g}$.

A recursive call for an orthogonal gate. Without loss of generality, we assume that $\mathcal{P}_i$ lies above and to the left of G_i, see Figure 5(b)+(c). We distinguish two cases.

(1) G_i is counterclockwise: Let $\ell \subseteq G_i$ be the vertical segment of G_i. Note that ℓ denotes the left endpoint of G_i if it only consists of a horizontal segment, see Figure 5(d). Let $L \subseteq \mathcal{P}_i$ be maximal rectangle with right side ℓ, see Figure 5(c)+(d). We choose $\overline{g}$ as a vertex from the boundary of L not lying on G_i and not lying on the right side of L.

(2) G_i is clockwise: Consider by $t \subset G_i$ the horizontal segment of G_i. Note that t denotes the top endpoint of G_i if it consists only of a vertical segment. Let $T \subseteq \mathcal{P}_i$ be the maximal rectangle with bottom side t, see Figure 5(b). We choose $\overline{g}$ as a vertex from the boundary of T not lying on G_i.

Finally, we again recurse on $\mathcal{P} := \mathcal{P}_i \cup \mathcal{V}(\overline{g})$ and $g := \overline{g}$.

2.3 Analysis of the algorithm

We consider the recursion tree T of our algorithm. In particular, each guard placed in the corresponding recursion step is a node in T. An edge between a father node g and a child node $\overline{g}$ exists if g creates a subpolyomino $\mathcal{P}_i$ causing a recursive call on $\mathcal{P}_i \cup \mathcal{V}(\overline{g})$ and $\overline{g}$. We say that g_2 is an *indirect* child of g_1 if there is a sequence of nodes $g_1 = \overline{g}_1, \ldots, \overline{g}_\ell = g_2$ such that $\overline{g}_i$ is the father of $\overline{g}_{i+1}$ for $i = 1, \ldots, \ell - 1$.

For a clearer presentation, we say that g_1 is a *direct* child of g_2, if $\ell = 1$.

As $\overline{g}$ is chosen from the segment resulting from pushing a vertical or horizontal line of G_i until a vertex of $\mathcal{P}$ is hit for the first time, we obtain the following:

▶ **Observation 3.** *All cells from $\mathcal{P}_i$ sharing at least one point or a side with G_i are seen by $\overline{g}$, see the dark green areas in Figure 6.*

As our algorithm recurses on $\mathcal{P}_i \cup \mathcal{V}(\overline{g})$, we obtain the following as a direct consequence of Observation 3.

▶ **Corollary 4.** *For each recursive call on $\mathcal{P}_i \cup \mathcal{V}(\overline{g})$ and $\overline{g}$ the guard $\overline{g}$ is placed inside $\mathcal{P}_i$ within a distance of at least 1 to the corresponding gate G_i.*

The next lemma addresses neighbored gates and their orientations and is a key in the analysis of the algorithm:

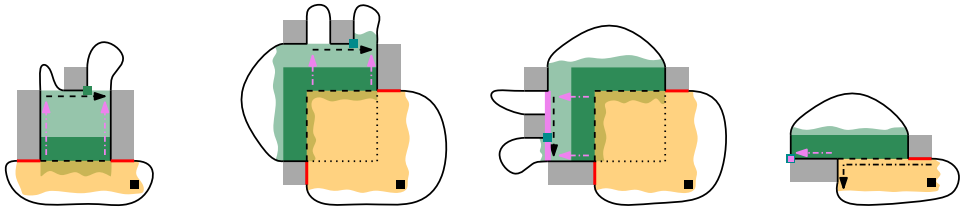


Figure 6 All cells from $\mathcal{P}_i$ that share at least a point or a side with G_i are seen by $\overline{g}$.

▶ **Lemma 5.** *Let G_1 and G_2 be two gates created in the same recursion step. If G_1 and G_2 share an endpoint, they have the same orientation.*

Proof. The proof follows the case distinction of the recursion step, and let k be the number of gates created in the considered recursion step. As $k \geq 2$, Case (1) is not relevant. If $k = 2$, the Cases (2.2) and (2.3) directly imply the same orientation of G_1 and G_2. In Case (2.1) both gates are connected via one side of the boundary of $\mathcal{P}$, i.e., $\gamma_1 = \gamma_2$. Thus, G_1 and G_2 cannot share an endpoint, see Figure 4 (2.1). Finally, let at least $k = 3$ gates be created during the considered recursion step. If G_1 and G_2 share an endpoint, the description of the case ensures that they are oriented in the same direction. ◀

We now consider the geometric form of gates and their positions relative to one another.

▶ **Lemma 6.** *If two gates G_1, G_2 created in the same recursion step share an endpoint, both G_1 and G_2 are parallel gates lying orthogonal to one another.*

Proof. The proof is by contradiction. Assume that at least G_1 is orthogonal. Two adjacent segments from different gates cannot be collinear, because otherwise there is a side from the boundary of $\mathcal{P}$ lying between cells from the polygon, see Figure 7b. As G_1 is orthogonal and adjacent to G_2, there is a sequence of consecutive segments s_1, s_2, s_3 from these gates, where s_1, s_2 and s_2, s_3 are adjacent to one another, see Figure 7c. As the guard g lies to the right of s_1, s_2, and s_3, it sees at least one cell outside of $\mathcal{V}(g)$ being a contradiction. ◀

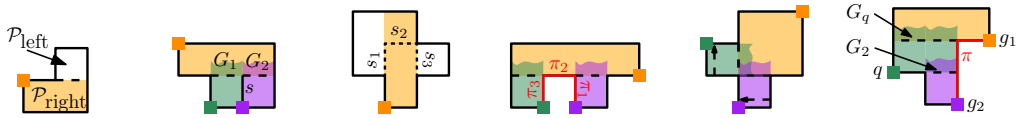


Figure 7 (a) A segment of a gate separates $\mathcal{P}$ into a left and a right polyomino. (b) Two parallel gates lying collinear are not possible. (c) Two gates laying adjacent where at least one is an orthogonal gate is not possible. (d) A shortest path connecting two guards not being (indirect) children of each other where the corresponding gates are not adjacent. (e) Two guards being not (indirect) children of each other where the corresponding gates are adjacent. (f) Sequence of children.

We now analyze the dispersion distance of the guard set constructed by our approach based on Corollary 4 and Lemmas 5 and 6.

▶ **Lemma 7.** *The constructed guard set has a dispersion distance of at least 3.*

Proof. In order to prove the lemma we consider an arbitrary pair of placed guards g_1, g_2 and distinguish three cases: (1) Neither g_1 is an indirect child of g_2 nor vice versa. (2) g_1 is an indirect but not a direct child of g_2 or vice versa. (3) g_1 is a direct child of g_2 or vice versa.

In the following, we consider all three cases separately. The intuitions for the three cases are the following: (1) If g_1 and g_2 have the common father g let G_1, G_2 be the corresponding gates. If G_1, G_2 are not adjacent these gates are within a distance of at least 1. Hence, applying Corollary 4 twice leads to a distance of at least 3, see Figure 7(d). If G_1, G_2 are adjacent, Lemma 5 implies a distance of at least 3, see Figure 7(e). If g_1 or g_2 ist not a direct child of g, similar arguments apply. (2) Applying Observation 3 yields two gates G_1, G_2 between g_1 and g_2 where each path between G_1 and G_2 has a length of 1, see Figure 7(f). Finally, applying Corollary 4 and the observation that g_2 does not lie on a gate caused by g_2 yields a distance of at least 3. (3) Intuitively speaking Figure 6 implies that the same arguments as used in (2) apply to (3).

Neither g_1 is an indirect child of g_2, nor vice versa. First consider the case in which g_1 and g_2 are direct children of the same father g. Let G_1 and G_2 be the gates created by g corresponding to g_1 and g_2. Let π be a shortest path connecting g_1 and g_2. Note that $\pi = (\pi_1, \pi_2, \pi_3)$ contains three subpaths, where π_1 connects g_1 and G_1, π_2 connects G_1 and G_2, and π_3 connects G_2 and g_2. Corollary 4 implies that π_1 and π_3 have a length of at least 1. If G_1 and G_2 do not share an endpoint, we obtain that π_2 has a length of at least 1, implying that π has a length of at least 3, see Figure 7(d).

If G_1 and G_2 share an endpoint, Lemma 5 implies that G_1 and G_2 have the same orientation. Without loss of generality, assume that G_1, G_2 are oriented clockwise. Furthermore, Lemma 6 implies that G_1, G_2 are parallel gates, whose segments lie orthogonal to one another. Without loss of generality, assume that G_1, G_2 are ordered clockwise and that G_1 (G_2) is horizontal (vertical), see Figure 7(e). As G_1 and G_2 are oriented clockwise, applying Corollary 4 simultaneously to g_1 and g_2 implies that the x-coordinate of g_1 is at least one larger than the x-coordinate of g_2 and the y-coordinate of g_1 is at least two smaller than the y-coordinate of g_2. Thus, g_1 and g_2 have a distance of at least 3.

g_1 is an indirect but not a direct child of g_2, or vice versa. Without loss of generality, assume that g_1 is an indirect child of g_2. Thus, there is at least one further guard q being placed between g_1 and g_2, i.e., such that q is a direct or indirect child of g_1 and g_2 is a direct or indirect child of q, see Figure 7(f). This implies that the shortest path π connecting g_1 and g_2 has to cross the visibility region $\mathcal{V}(q)$ of q. Observation 3 implies that this subpath of π has a length of at least 1. Let G_q and G_2 be the gates between g_1, q and q, g_2. Corollary 4 implies that the length of the subpath of π connecting G_2 with g_2 is at least 1. Furthermore, g_1 cannot lie on G_q implying that the length of the subpath of π connecting g_1 and G_q also has a length of 1. Hence, π has a length of at least 3.

g_1 is a direct child of g_2, or vice versa. Without loss of generality, assume that each segment of the gate G_1 corresponding to g_1 has a length of 1, see Figure 8 for the different cases. In the case of a parallel gate, assume without loss of generality that g_2 lies adjacent to an endpoint G_1 resulting in a distance of 3, see Figure 8(a). In the case of an orthogonal gate, and that G_1 consist of two segments, assume without loss of generality that g_2 lies as close as possible to both segments of G_1 resulting in a distance of 4 between g_1 and g_2, see Figure 8(b)+(c). Finally, in the case of an orthogonal gate that consist of a single segment, assume without loss of generality that g_2 lies on the wall collinear with G_1 resulting in a distance of at least 3, see Figure 8(d).

This concludes the proof of Lemma 7. ◀

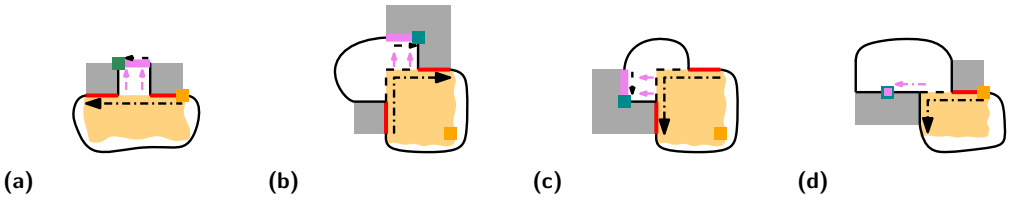


Figure 8 Assuming a position for g decreasing at most its distance to q: (a) A parallel gate. (b) A clockwise oriented orthogonal gate made up of two segments. (c) A counterclockwise oriented orthogonal gate made up of two segments. (d) An orthogonal gate made up of one segment.

As Lemma 1 provides an upper bound on the dispersion distance in simple polyominoes and Lemma 7 the matching lower bound, these lemmata together prove Theorem 2.

3 Computational complexity

In this section we study the computational complexity of computing guard sets that maximize the smallest distance between all pairs of guards within the guard set. In particular, we show the following.

▶ **Theorem 8.** *Deciding whether there exists a guard set with a dispersion distance of 5 for a given polyomino is* NP*-complete.*

3.1 Outline of the NP-hardness reduction

For showing NP-hardness we utilize the problem Planar Monotone 3Sat that is shown to be NP-complete by de Berg and Khosravi [19]. This problem asks to decide the satisfiability of a Boolean 3-CNF formula for which the literals in each clause are either all negated or all unnegated, and the corresponding variable-clause incidence graph is planar.

To this end, we will construct polyominoes that will represent variables and clauses. Because a variable may contribute to multiple clauses, we model a specific shape that duplicates the given assignment. Furthermore, we describe simple shapes that are used to connect different subshapes, while maintaining the given assignment from the variables. Figure 9 gives a high-level overview of the construction and the main gadgets.

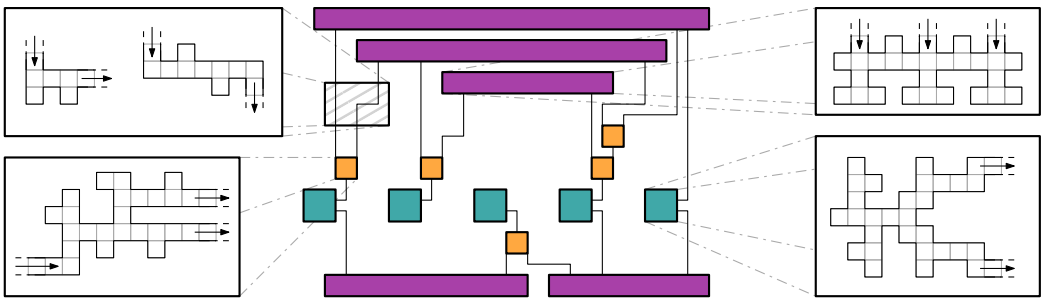


Figure 9 Symbolic overview of the NP-hardness reduction. The depicted instance is due to the Planar Monotone 3Sat formula $\varphi = (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_3} \vee \overline{x_4} \vee \overline{x_5})$. Variables are in dark cyan, clauses in magenta, duplicator gadgets in orange, and connectors as lines.

The idea of the reduction is as follows: As shown in Figure 9, all gadgets have *open ends* (depicted by arrows) where they are connected to one another. These openings are distinguished as input and output depending on how the arrows are oriented. Starting from the

variable gadgets there is a sequence of outputs and inputs that ends in the clauses gadgets. Through intensive use of niches, we force the possible guard sets within gadgets to essentially two sets. In particular, the first set covers the output of a gadget from within the gadget, and therefore also already the input of the next gadget in the sequence. The second set have to cover the input from within the gadget (because it is not already covered from before). We use these constraints and observations to propagate variable assignments through the construction, i.e., to obtain a *guarding direction*.

3.2 Setting up the gadgets

We now give the description of the involved gadgets. In addition we prove several lemma that will then put together to yield a proof of Theorem 8.

Variable gadget The gadget that is depicted in Figure 10 allows exactly two guard sets with a dispersion distance of 5, so it models true and false assignments. In order to do so, we force guards to unique positions within specific subregions, which results in the fact that all other feasible guard positions are restricted to two disjoint sets.

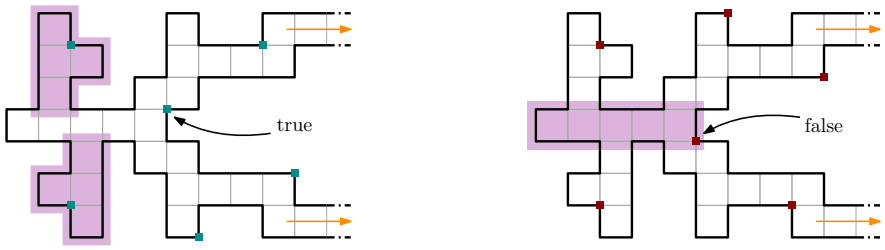


Figure 10 The figure shows the variable gadget, and regions that are used in Lemmas 9 and 10. (a) shows the guard set for a true assignment, while (b) shows the respective set for a false assignment.

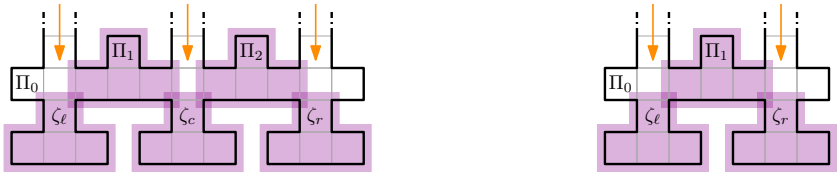
▶ **Lemma 9.** *Within the variable gadget, no guard set has dispersion distance larger than 5.*

Proof. Consider the dark magenta regions ("T-shapes") in Figure 10a. No guard set with at least two guards realizes a dispersion distance larger than 5 within such a region. The only vertex that could partly cover such a T-shape from the outside, is itself a vertex from another T-shape. Therefore, both these regions have to be covered from uniquely from within it. Thus, the largest possible distance between these guards is 5, as shown by cyan squares. ◀

▶ **Lemma 10.** *Within the variable gadget there are exactly two guard sets realizing a dispersion distance of 5.*

Proof. As the guard placement within the T-shapes is unique (see Lemma 9), the only variability lies within the magenta region shown in Figure 10b. However, by maintaining a distance of 5 to the necessary guards placed within the T-shapes, there are exactly two vertices that remain for covering this region. By choosing one, all the other positions follow uniquely. Hence, there are exactly two guard sets with a dispersion distance of 5. ◀

Clause gadget. The clause gadget is depicted in Figure 11. The overall idea of this gadget is that it does not allow for a guard set with a dispersion distance of at least 5 if guards have to be placed only on vertices of this subshape. Hence, some specific cells have to be already covered from outside the shape, what will be related to satisfying the clause.



(a) **(b)**

Figure 11 (a) The depicted shape represents a clause gadget containing three literals, while (b) shows a clause with two literals. The cells labeled with ζ can be covered from outside the gadget.

Note that a clause gadget "contains" basically two types of T-shapes, as shown in Figure 11. We call them *prospects* if they can partly be covered from outside, and *checkers* otherwise.

▶ **Lemma 11.** *There is no guard set with a dispersion distance of at least 5 for the shape representing the clause gadget when placing guards only within this shape.*

Proof. We will only argue this in detail for the case that the clause contains three literals; similar arguments hold for the remaining case. Consider one of the colored T-shapes in Figure 11a. Only a single guard can be placed within such a region if a dispersion distance of at least 5 is required. Consider three consecutive T-shapes, such that two of them are prospects. The shortest path connecting the six potential guard locations has a length of 9. Hence, no guard set with a dispersion distance of 5 exists. ◀

▶ **Lemma 12.** *If at least one cell of the clause gadget is covered from outside the gadget, a feasible guard set with a dispersion distance of 5 exists.*

Proof. Again, we will only argue the more complicated case, i.e., a clause containing three literals. For this, consider the marked region in Figure 11a and distinguish the following.

First assume that the central connector is covered from outside the gadget. Thus, in particular cell ζ_c is already covered, and we can place a guard in a bottom corner of this prospect. This results in two disjoint pairs of uncovered T-shapes, such that each of these pairs share a vertex of the shape. Without loss of generality, consider the left pair. Two guards can be placed at the bottom left vertex of ζ_ℓ and bottom right of Π_1 with a distance of 5. Because the guard that covers Π_1 already covers the bottom part of the other checker, we can place a guard in its niche, i.e., at the top right vertex of Π_2. Placing a guard at the bottom right vertex of ζ_r completes the guard set.

On the other hand, without loss of generality, let the left connector be already covered, i.e., the cell ζ_ℓ is covered. A guard can be placed in the leftmost niche Π_0 to cover the bottom part of both checker. Therefore, we only have to cover Π_1 and Π_2 by placing guards at their respective top vertices. It remains to cover the prospects. Because guards are placed at the top vertices of the checker's niches, we can place guards in appropriate distances to obtain a guard set with dispersion distance 5. ◀

Due to the respective embedding of the overall shape it may be necessary to enlarge the clause gadget, see Figure 12.

▶ **Lemma 13.** *A clause gadget can be enlarged in a way that all functionalities are maintained.*

Proof. If the clause contains three literals, we replace the T-shape checker by the colored region in Figure 12a. Note that this region is mirrored vertically along the center connector, and that the region between Π_2 and Π_3 can be enlarged arbitrarily.

A crucial observation is that the niches Π_1 and Π_3 coincide in the short clause gadget, and therefore apply the same restrictions as before. The additional niche Π_2 guarantees that we cannot place a guard at the bottom right vertex of Π_1, assuming that the clause is not satisfied through an assignment.

If the clause only contains two literals, the T-shape checker will be replaced by the colored region in Figure 12b. The correctness follows analogously. ◀

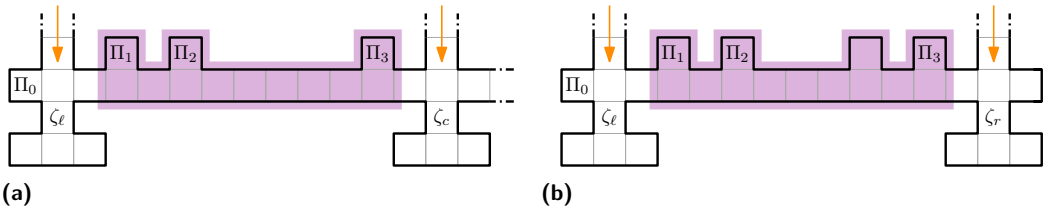


Figure 12 (a) shows the left part of an enlarged clause gadget containing three literals, while (b) shows the respective enlarged shape for clauses containing two literals.

Duplicator gadget. Because a variable may contribute to more than one clause, we need to duplicate the respective assignment. For this purpose, we construct the duplicator gadget that is depicted in Figure 13. It works as follows: if the incoming connector is covered from outside the gadget, both outgoing connectors can be covered from within the gadget. Similarly, if the incoming connector has to be covered from within the gadget, the outgoing connectors must be covered from outside the gadget.

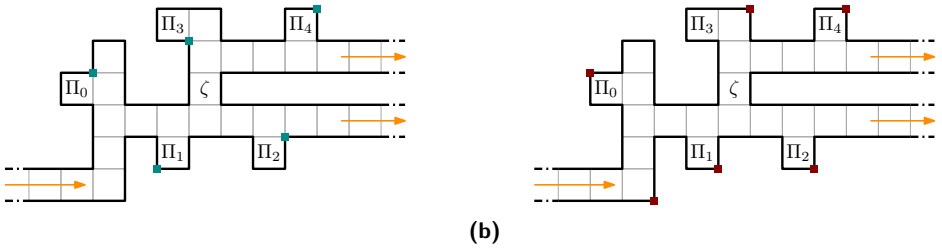


Figure 13 The figure shows the duplicator gadget. (a) shows a set of guards duplicating a true assignment, while (b) shows the respective guard set for a false assignment.

▶ **Lemma 14.** *The duplicator gadget is correct, i.e., any output is equal to the input.*

Proof. First consider the situation given in Figure 13a. Because the incoming connector is covered from the outside, we want to cover the outgoing connectors from the inside. We will argue that the configuration in the marked region is unique and fulfills the requirements. Because of niche Π_3, covering Π_1 by a guard placed at vertices of ζ is not possible, and because of Π_0 and Π_2 the guard covering Π_1 is uniquely defined. Because this position is fixed, all other positions follow.

Now consider the situation in Figure 13b. The incoming connector has to be covered from the inside. Because of Π_0, the position of the guard covering the incoming connector is uniquely defined. Therefore, there are two positions left to cover the niche Π_1; however, because of Π_3, we cannot choose the vertex of ζ. It follows that the positions for guarding Π_1 and Π_2 are uniquely defined. Because ζ cannot be covered from the outside, the guard

covering this square is also uniquely defined and also covers Π_3 simultaneously. This again leaves only a single position to cover Π_4. Overall, this leaves some squares of the outgoing connectors uncovered, so that they have to be covered from outside the gadget. ◀

Because a necessary condition to the problem is that the guard set have to cover the polyomino completely, we have to ensure that visibility regions that are induced by guards from clause gadgets do not interfere the assignment given due to guards from within the variable gadgets. So, if a clause $C_i = (x_j, x_k, x_\ell)$ is satisfied by say x_j, we have to make sure that other clauses containing x_k or x_ℓ but not x_j are not become automatically satisfied by *backward guarding* from guards in C_i. Preventing this is also the job of the duplicator gadget.

▶ **Lemma 15.** *Backward guarding of an output of the duplicator gadget cannot result in covering the other output from within the gadget.*

Proof. Consider without loss of generality that the duplicator gadget propagates false from the respective variable. A critical situation would occur if the assignment could be flipped within a duplicator gadget due to the coverage coming from a clause gadget, i.e., propagating true to the other output.

As argued above, due to the positions of the niches, the positions for guards are highly restricted. The coverage from the outside does not cover any of these niches. Therefore, it does not change the possible set of guard positions. ◀

Connector gadget. Now that we have the main components, we need to connect them. For this, we introduce two different connector gadgets, see Figure 14.

(a) (b)

Figure 14 (a) shows an L-connector, and (b) shows a Z-connector. The dark cyan and red colored guard sets propagate whether a variable is set to true or false, respectively.

▶ **Lemma 16.** *All connector gadgets fulfill the property that either the input, or the output can be guarded from within the gadgets by a guard set with a dispersion distance of 5.*

Proof. As these gadgets connect the previous ones, we distinguish between the cases that their input is already covered or not. Remark that if the input is already covered, we want to cover the output within the connector, and vice versa.

We prove this by providing specific sets of guards, regarding the different settings. For the case that the input is already covered, consider the dark cyan placed guards. The distance between guards is at least 5, and everything is covered. For the case that the input has to be covered, consider the red guarding positions. The placement of the niches force the position of the guard that covers the input. All other positions follow uniquely. It is easy to see that no more guards can be placed, so the output remains uncovered. ◀

3.3 Completing the NP-hardness proof

We described several gadgets that will now be used to construct an instance $\mathcal{P}_\varphi$ of the Dispersive Art Gallery Problem from any Boolean formula φ that is an instance of Planar Monotone 3SAT; this yields a proof for Theorem 8 that we restate here.

▶ **Theorem 8.** *Deciding whether there exists a guard set with a dispersion distance of 5 for a given polyomino is* NP*-complete.*

Proof. Note that the problem is obviously in NP. It is easy to verify whether a potential set of vertices is in fact a guard set for the given polyomino. Furthermore we can compute the dispersion distance of a guard set in polynomial time.

To show that the problem is NP-hard, we reduce from Planar Monotone 3Sat. For a given formula φ we create an instance $\mathcal{P}_\varphi$ of dispersive AGP as follows; again, see Figure 9 for the high-level idea of the construction.

Consider the rectilinear embedding of the graph given by φ. For every variable, we place a variable gadget horizontally in a row. Each clause is represented by a clause gadget. Due to the rectilinear embedding, we can place them vertically behind one another, and expand them appropriately if necessary as shown above. Without loss of generality, we place the clauses containing only unnegated literals above the variables, and below otherwise. If a literal x_i occurs in m_i many clauses, we construct $m_i - 1$ duplicator gadgets between vertically between clauses and variables. We properly place a set of connector gadgets to connect variables to duplicator gadgets, as well as the outputs of duplicator gadgets to respective inputs, and duplicator gadgets to the respective clauses. Note that variables are connected to clauses if they contribute only to a single clause.

If φ is satisfiable, then there is a guard set with dispersion distance 5 for $\mathcal{P}_\varphi$.

Consider a satisfying assignment of φ. A guard set with a dispersion distance of 5 for $\mathcal{P}_\varphi$ can be constructed as follows: From the given assignment of the variable x_i the respective set of guards within the variable gadget is chosen. For every connector and duplicator gadget, there is a set of guards that maintains the assignment. Because we propagate the satisfying assignment through the gadgets, at least one literal satisfies each clause. Hence, we can choose guards within each clause gadget that has dispersion distance of 5, because in each of these gadgets at least one of the cells are covered from the outside.

If there is a guard set with dispersion distance 5 for $\mathcal{P}_\varphi$, then φ is satisfiable.

Consider a guard set for $\mathcal{P}_\varphi$ that has a dispersion distance of 5. As argued above, at least one cell of each clause gadget are covered from outside of the respective gadget, because otherwise there is no such desired guard set. Furthermore, there is no guard set for the variable gadget that has a dispersion distance larger than 5, and there are only two sets that realize this pairwise minimum distance. For every path from variables to clauses, the duplicator and connector gadgets provide specific locations for guards that maintain a dispersion distance of 5. Hence, the guards within the variable gadget of $\mathcal{P}_\varphi$ realize a satisfying assignment for φ.

This concludes the proof. ◀

4 Trees

While computing guard sets with maximum dispersion distance is NP-hard in general, we present a linear-time algorithm to compute optimal solutions in tree-shaped polyominoes. Recall that a polyomino $\mathcal{P}$ is tree-shaped if the dual graph of $\mathcal{P}$ is a tree. In particular, these polyominoes do not contain a 2×2 subpolyomino.

▶ **Theorem 17.** *Given a tree-shaped polyomino $\mathcal{P}$ with n vertices, there is an $O(n)$ dynamic programming approach for computing guard sets of maximum dispersion distance.*

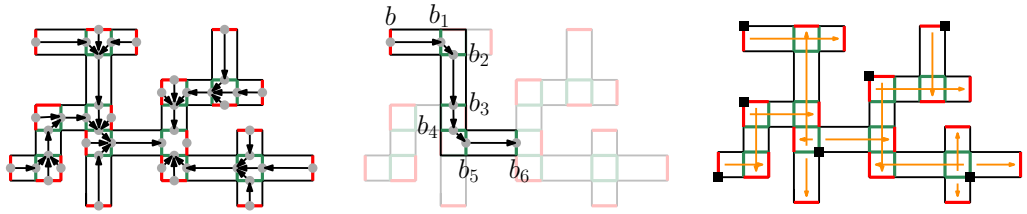


Figure 15 (a) The directed tree (gray dots and black directed edges) and borders (red and green segments) of a tree-shaped polyomino. (b) The unique path from an outer border b to the inner border b_6 of the "root cell". (c) An optimal guard set (black squares) generated by our approach, with "screening" directions (orange arrows).

The main structure used in our algorithm are *borders*, which are defined as follows: Let $R \subseteq \mathcal{P}$ be the set of all maximal rectangles, see Figure 15. Note that R covers $\mathcal{P}$. We refer to the part of the boundary of a rectangle to be an *inner border* if it is not part of the boundary of the polyomino, see the green segments in Figure 15a. The two boundary parts of each rectangle having length 1 are called *outer borders*, see the red segments in Figure 15a. Note that every outer border is on the boundary of $\mathcal{P}$, and every border is either an inner or an outer border.

For every border we consider *states* describing (1) which of its endpoints are chosen as guards, (2) weather there is already a guard placed "behind" this border, and (3) the ratio of the shortest distances of its endpoints to a placed guard.

Applying a straightforward dynamic programming approach on a directed tree structure induced by the above-mentioned borders yields an algorithm for computing guard sets of maximum dispersion distance for the class of tree-shaped polyominoes. The observation that (3) does not cause $\Omega(n)$ but only a constant many states, results in the fact that the runtime of this algorithm is linear in the number of vertices of the input polyomino.

For the detailed description of the algorithm we refer to the full version [42].

5 Conclusion and future work

In this paper we introduced the dispersive AGP and investigated it for vertex guards in polyominoes. We described an algorithm that constructs worst-case optimal solutions of dispersion distance 3, and showed that it is NP-complete to decide whether a dispersion distance of 5 can be achieved. We were also able to find a linear-time dynamic programming approach to compute guard sets of maximum dispersion distance for tree-shaped polyominoes, see the full version [42].

Several open questions remain. Is it possible to close the gap to the worst-case, i.e., is deciding whether a dispersion distance of 4 can be achieved NP-hard as well? Is it possible to compute worst-case solutions in the case of non-simple polyominoes? It seems very promising that our method can be extended. Other open questions concern approximation algorithms. Is there a constant-factor approximation for this problem?

What can be said about the ratio between the cardinality of guard sets in optimal solutions for the dispersive and the classic art gallery problem? As shown in Figure 1 this ratio is at least 2, while the ratio between the dispersion distances increases arbitrarily.

What can be said about the dispersive art gallery problem in terrains, or general polygons?

References

1 James Abello, Vladimir Estivill-Castro, Thomas C. Shermer, and Jorge Urrutia. Illumination of orthogonal polygons with orthogonal floodlights. *International Journal of Computational Geometry and Applications*, 8(1):25–38, 1998. doi:10.1142/S0218195998000035.

2 Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. Irrational guards are sometimes needed. In *Symposium on Computational Geometry (SoCG)*, pages 3:1–3:15, 2017. doi:10.4230/LIPIcs.SoCG.2017.3.

3 Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. *Journal of the ACM*, 69(1):4:1–4:70, 2022. doi:10.1145/3486220.

4 Andreas Bärtschi, Subir Kumar Ghosh, Matús Mihalák, Thomas Tschager, and Peter Widmayer. Improved bounds for the conflict-free chromatic art gallery problem. In *Symposium on Computational Geometry (SoCG)*, pages 144–153, 2014. doi:10.1145/2582112.2582117.

5 Andreas Bärtschi and Subhash Suri. Conflict-free chromatic art gallery coverage. *Algorithmica*, 68(1):265–283, 2014. doi:10.1007/s00453-012-9732-5.

6 Christoph Baur and Sándor P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001. doi:10.1007/s00453-001-0022-x.

7 Marc Benkert, Joachim Gudmundsson, Christian Knauer, René van Oostrum, and Alexander Wolff. A polynomial-time approximation algorithm for a geometric dispersion problem. *International Journal of Computational Geometry and Applications*, 19(3):267–288, 2009. doi:10.1142/S0218195909002952.

8 Therese C. Biedl, Mohammad T. Irfan, Justin Iwerks, Joondong Kim, and Joseph S. B. Mitchell. Guarding polyominoes. In *Symposium on Computational Geometry (SoCG)*, pages 387–396, 2011. doi:10.1145/1998196.1998261.

9 Therese C. Biedl, Mohammad T. Irfan, Justin Iwerks, Joondong Kim, and Joseph S. B. Mitchell. The art gallery theorem for polyominoes. *Discrete Computational Geometry*, 48(3):711–720, 2012. doi:10.1007/s00454-012-9429-1.

10 Therese C. Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Dispersion for intervals: A geometric approach. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 37–44, 2021. doi:10.1137/1.9781611976496.4.

11 Therese C. Biedl and Saeed Mehrabi. On r-guarding thin orthogonal polygons. In *Symposium on Algorithms and Computation (ISAAC)*, pages 17:1–17:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.17.

12 Édouard Bonnet and Panos Giannopoulos. Orthogonal terrain guarding is NP-complete. *Journal of Computational Geometry*, 10(2):21–44, 2019. doi:10.20382/jocg.v10i2a3.

13 Prosenjit Bose, Leonidas J. Guibas, Anna Lubiw, Mark H. Overmars, Diane L. Souvaine, and Jorge Urrutia. The floodlight problem. *International Journal of Computational Geometry and Applications*, 7(1/2):153–163, 1997. doi:10.1142/S0218195997000090.

14 Sergio Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62(2):49–73, 2007. doi:10.1016/j.jalgor.2004.06.009.

15 Paola Cappanera. A survey on obnoxious facility location problems. Technical report, Università di Pisa, 1999.

16 Barun Chandra and Magnús M. Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, 38(2):438–465, 2001. doi:10.1006/jagm.2000.1145.

17 Vasek Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, 18(1):39–41, 1975. doi:10.1016/0095-8956(75)90061-1.

18 Jurek Czyzowicz, Eduardo Rivera-Campo, and Jorge Urrutia. Optimal floodlight illumination of stages. In *Canadian Conference on Computational Geometry (CCCG)*, pages 393–398, 1993.

19 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012. doi:10.1142/S0218195912500045.

20 Adrian Dumitrescu and Minghui Jiang. Dispersion in disks. *Theory of Computing Systems*, 51(2):125–142, 2012. doi:10.1007/s00224-011-9331-x.

21 Lawrence H. Erickson and Steven M. LaValle. A chromatic art gallery problem. Technical report, University of Illinois, 2010.

22 Lawrence H. Erickson and Steven M. LaValle. An art gallery approach to ensuring that landmarks are distinguishable. In *Robotics: Science and Systems VII*, 2011. doi:10.15607/RSS.2011.VII.011.

23 Erhan Erkut and Susan Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40(3):275–291, 1989. doi:10.1016/0377-2217(89)90420-7.

24 Vladimir Estivill-Castro, Joseph O'Rourke, Jorge Urrutia, and Dianna Xu. Illumination of polygons with vertex lights. *Information Processing Letters*, 56(1):9–13, 1995. doi:10.1016/0020-0190(95)00129-Z.

25 Sándor P. Fekete, Stephan Friedrichs, Michael Hemmer, Joseph S. B. Mitchell, and Christiane Schmidt. On the chromatic art gallery problem. In *Canadian Conference on Computational Geometry (CCCG)*, 2014. URL: http://www.cccg.ca/proceedings/2014/papers/paper11.pdf.

26 Sándor P. Fekete and Henk Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38(3):501–511, 2004. doi:10.1007/s00453-003-1074-x.

27 Jirí Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306–316, 2005. doi:10.1016/j.dam.2004.02.018.

28 Steve Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory*, 24(3):374, 1978. doi:10.1016/0095-8956(78)90059-X.

29 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Symposium on Computational Geometry (SoCG)*, pages 281–288, 1991. doi:10.1145/109648.109680.

30 Frank Hoffmann. On the rectilinear art gallery problem. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 717–728, 1990. doi:10.1007/BFb0032069.

31 Frank Hoffmann, Klaus Kriegel, Subhash Suri, Kevin Verbeek, and Max Willert. Tight bounds for conflict-free chromatic guarding of orthogonal art galleries. *Computational Geometry*, 73:24–34, 2018. doi:10.1016/j.comgeo.2018.01.003.

32 Hiro Ito, Hideyuki Uehara, and Mitsuo Yokoyama. NP-completeness of stage illumination problems. In *Japanese Conference on Discrete and Computational Geometry (JCDCG)*, pages 158–165, 1998. doi:10.1007/978-3-540-46515-7_12.

33 Chuzo Iwamoto and Tatsuaki Ibusuki. Computational complexity of the chromatic art gallery problem for orthogonal polygons. In *Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 146–157, 2020. doi:10.1007/978-3-030-39881-1_13.

34 Minghui Jiang, Sergey Bereg, Zhongping Qin, and Binhai Zhu. New bounds on map labeling with circular labels. In *Symposium on Algorithms and Computation (ISAAC)*, pages 606–617, 2004. doi:10.1007/978-3-540-30551-4_53.

35 Jeff Kahn, Maria Klawe, and Daniel Kleitman. Traditional galleries require fewer watchmen. *SIAM Journal on Algebraic Discrete Methods*, 4(2):194–206, 1983. doi:10.1137/0604020.

36 James King and Erik Krohn. Terrain guarding is NP-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011. doi:10.1137/100791506.

37 D. T. Lee and Arthur K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986. doi:10.1109/TIT.1986.1057165.

38 Shimin Li and Haitao Wang. Dispersing points on intervals. *Discrete Applied Mathematics*, 239:106–118, 2018. doi:10.1016/j.dam.2017.12.028.

39 Bengt J. Nilsson, David Orden, Leonidas Palios, Carlos Seara, and Pawel Zylinski. Illuminating the x-axis by α-floodlights. In *Symposium on Algorithms and Computation (ISAAC)*, pages 11:1–11:12, 2021. doi:10.4230/LIPIcs.ISAAC.2021.11.

40 Joseph O'Rourke. An alternate proof of the rectilinear art gallery theorem. *Journal of Geometry*, 21(1):118–130, 1983. doi:10.1007/BF01918136.

41 Joseph O'Rourke. *Art gallery theorems and algorithms*. Oxford New York, NY, USA, 1987.

42 Christian Rieck and Christian Scheffer. The dispersive art gallery problem, 2022. arXiv:2209.10291.

43 Dietmar Schuchardt and Hans-Dietrich Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41:261–267, 1995. doi:10.1002/malq.19950410212.

44 Thomas C. Shermer. Recent results in art galleries (geometry). *Proceedings of the IEEE*, 80(9):1384–1399, 1992.

45 William L. Steiger and Ileana Streinu. Illumination by floodlights. *Computational Geometry*, 10(1):57–70, 1998. doi:10.1016/S0925-7721(97)00027-8.

46 Jorge Urrutia. Art gallery and illumination problems. In *Handbook of Computational Geometry*, pages 973–1027, 2000. doi:10.1016/b978-044482537-7/50023-1.

47 Chris Worman and J. Mark Keil. Polygon decomposition and the orthogonal art gallery problem. *International Journal on Computational Geometry and Applications*, 17(2):105–138, 2007. doi:10.1142/S0218195907002264.